49

Visual Studio Tools for Office

В ЭТОЙ ГЛАВЕ...

- Типы проектов, создаваемые с помощью VSTO, и возможности, доступные для проектов
- Фундаментальные приемы, применимые ко всем типам решений VSTO
- Использование хост-компонентов и хост-элементов управления
- Построение решений VSTO со специальным пользовательским интерфейсом

Visual Studio Tools for Office (VSTO) — это технология, позволяющая настраивать и расширять приложения Microsoft Office и документы, используя .NET Framework. Она также включает инструменты, которые можно применять для облегчения настройки в Visual Studio — например, визуальный конструктор для ленты элементов управления офиса (office ribbon controls).

VSTO — последняя и самая крупная линейка продуктов, которые выпустила Microsoft, чтобы обеспечить возможность настройки приложений Office. Объектная модель, используемая для доступа к приложениям Office, со временем эволюционировала. Если вы применяли ее в прошлом, то ее части будут выглядеть знакомо. Если вы программировали дополнения VBA для приложений Office, то вы хорошо подготовлены для изучения приемов, рассматриваемых в этой главе. Однако классы, которые предоставляет VSTO для взаимодействия с Office, являются расширением, выходящим за рамки объектной модели Office. Например, классы VSTO включают функциональность привязки данных .NET.

До появления Visual Studio 2008 продукт VSTO был доступен в виде отдельной загрузки, которую вы могли получить, если собирались разрабатывать решения Office. Начиная с версии Visual Studio 2008, инструмент VSTO интегрирован в IDE-среду. Эта версия VSTO, известная как VSTO 3, включает полную поддержку Office 2007 и имеет множество новых средств. Сюда входит способность взаимодействовать с элементами управления содержимым Word, визуальный конструктор ленты управления, интеграция с VBA и многое другое.

В Visual Studio 2010 появилась версия VSTO 4, которая была расширена и усовершенствована по сравнению с предыдущими версиями. Она облегчает развертывание и не требует установки первичных сборок взаимодействия (Primary Interop Assemblies — PIA) на клиентских ПК — теперь это достигается встраиванием типов CLR 4. В VSTO 4 также включена поддержка Office 2010.

Никаких предварительных знаний VSTO и его предшественников для изучения этой главы не требуется.

Обзор VSTO

VSTO состоит из следующих компонентов:

- набор шаблонов проектов, которые можно использовать для создания различных типов решений VSTO;
- поддержка конструктора для визуальной компоновки инструментальной ленты, панелей действий и специальных панелей задач;
- классы, построенные на базе сборок Office Primary Interop Assemblies (PIA), обеспечивающие расширенные возможности.

VSTO поддерживает следующие версии Office — 2003, 2007 и 2010. Библиотека классов VSTO поставляется в двух вариантах, по одной для каждой из версий Office с использованием разных наборов сборок. Для простоты внимание в этой главе будет сосредоточено на версии 2007. Общая архитектура решений VSTO показана на рис. 49.1.

Обратите внимание, что сборка PIA, показанная на рис. 49.1, требуется только во время проектирования, ориентированного на .NET 4, поскольку при развертывании у клиента используется встраивание типов. Это не касается решений VSTO, нацеленных на .NET 3.5, где сборка PIA необходима как на машине разработки, так и на целевой машине.

Типы проектов

На рис. 49.2 можно видеть шаблоны проектов, доступные в Visual Studio для Office 2007 (похожий список доступен для Office 2010).

В этой главе мы сосредоточимся на Office 2007.

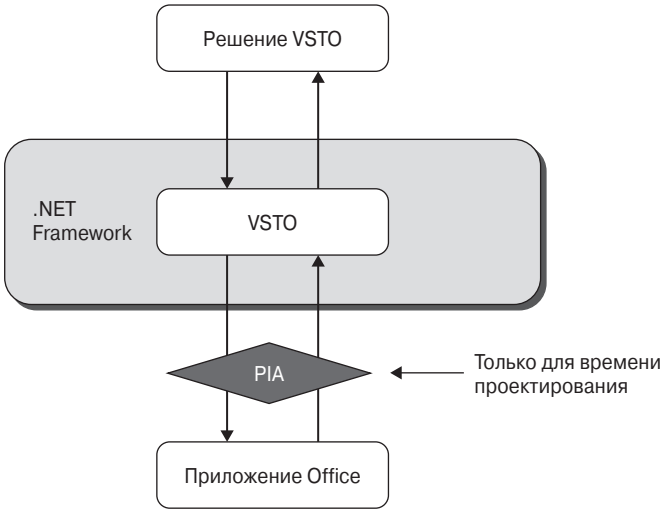

Рис. 49.1. *Общая архитектура VSTO*

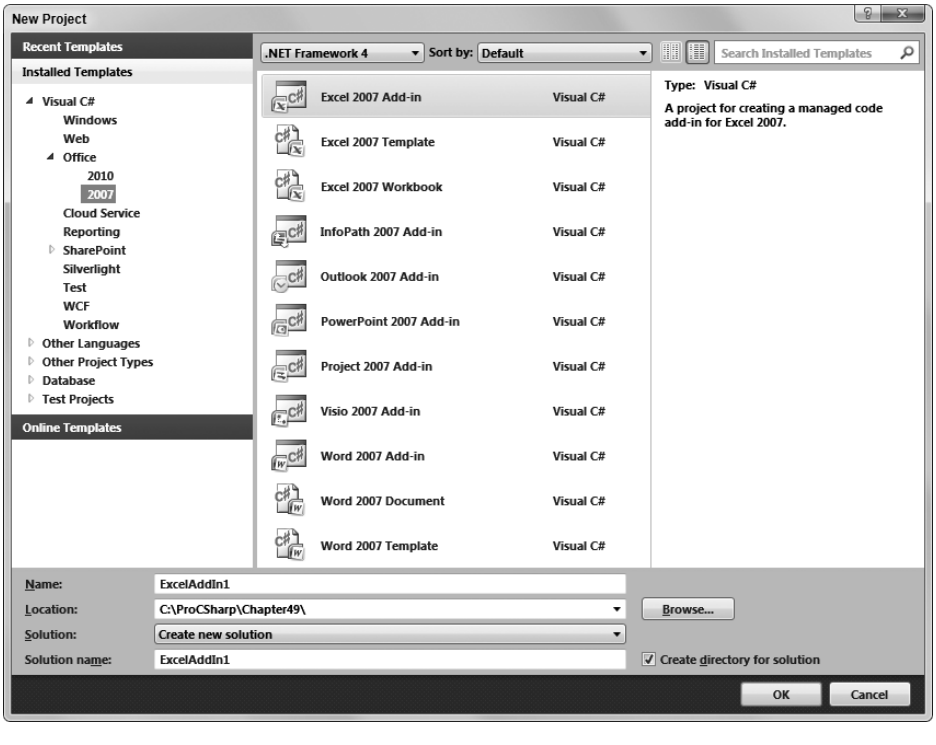

Рис. 49.2. *Шаблоны проектов VSTO*

Шаблоны проектов VSTO подразделяются на следующие категории:

- настройки (customization) уровня документа;
- дополнения уровня приложения;
- шаблоны рабочего потока SharePoint (на рис. 49.2 не показаны, поскольку являются отдельной категорией шаблонов).

Следует отметить, что предыдущая версия VSTO содержала и четвертую категорию — шаблоны форм InfoPath, которая в VSTO 4 уже не доступна.

В этой главе мы сосредоточим внимание на наиболее часто используемых типах проектов, которыми являются настройки уровня документа и дополнения уровня приложения.

Настройки уровня документа

Когда вы создаете проект этого типа, то генерируете сборку, которая будет привязана к индивидуальному документу, например, документу Word, шаблону Word или рабочей книге Excel. После загрузки документа ассоциированное приложение Office обнаружит настройку, загрузит сборку и сделает настройку VSTO доступной.

Проект этого типа можно использовать для обеспечения дополнительной функциональности конкретного бизнес-документа или целого класса документов, добавляя настройку к шаблону документа. Вы можете включить код, манипулирующий документом и его содержимым, в том числе любые встроенные объекты. Вы можете также предоставлять специальные меню, включая меню ленты инструментов, которые создаются с использованием VS Ribbon Designer.

При построении проекта уровня документа есть возможность либо создать новый документ, либо скопировать существующий в качестве начальной точки для разработки. Можно также выбрать тип создаваемого документа. Например, для документа Word можно выбрать создание документов `.docx` (по умолчанию), `.doc` или `.docm` (документ с макросами). Соответствующее диалоговое окно мастера создания нового проекта показано на рис. 49.3.

Дополнения уровня приложения

Дополнения уровня приложения отличаются от настройки уровня документа в том, что доступны всему целевому приложению Office. Обращаться к коду дополнения, который может включать меню, манипуляции документом и т.п., допускается независимо от загруженного в данный момент документа.

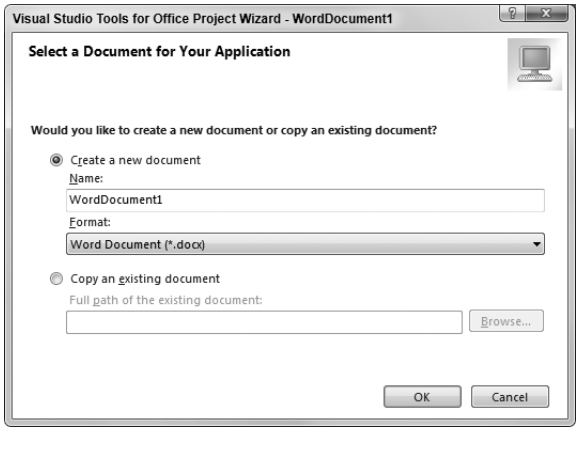

Рис. 49.3. *Выбор типа создаваемого документа*

После запуска такое приложение Office, как Word, ищет ассоциированные дополнения, которые имеют элементы в реестре и загружает любые необходимые сборки.

Шаблоны рабочих потоков SharePoint

Эти проекты предоставляют шаблон для создания приложений рабочего потока SharePoint. Они используются для управления потоком документов внутри процессов SharePoint. Создавая проект такого типа, вы можете выполнять специальный код в ключевые моменты жизненного цикла документа.

Средства проектов

Существует несколько средств, которые можно использовать в разных типах проектов VSTO, такие как интерактивные панели и элементы управления. Доступные средства определяются выбранным типом проекта. В табл. 49.1, 49.2 и 49.3 перечислены эти средства в соответствие с проектами, в которых они доступны.

Таблица 49.1. Средства настройки уровня документа

Средство	Описание
Панель действий	Панели действий — это диалоговые окна, располагаемые внутри панели действий Word или Excel. Здесь можно отображать любые желаемые элементы управления, что делает эту панель исключительно гибким средством расширения документов и приложений.
Кэш данных	Кэширование данных позволяет сохранять данные, используемые в документах, вне этих документов — в кэшированных островах данных (data island). Эти острова данных могут обновляться автоматически по источникам данных или вручную, позволяя документам Office обращаться к данным, когда исходный источник данных недоступен.
Конечная точка кода VBA	Как упоминалось ранее, VSTO обеспечивает возможность взаимодействия с VBA. В настройках уровня документа можно предоставить методы конечной точки, которые могут быть вызваны из кода VBA.
Хост-элементы управления	Хост-элементы управления — это расширенные оболочки существующих элементов управления в объектной модели Office. Этими объектами можно манипулировать и связывать с ними данные.
Смарт-теги	Смарт-теги — это объекты, которые встраиваются в документы Office и имеют типизированное содержимое. Они автоматически обнаруживаются внутри содержимого документов Office; например, смарт-тег биржевого индекса добавляется автоматически, когда приложение обнаруживает соответствующий текст. Можно создавать собственные типы смарт-тегов и определять операции, которые должны выполняться на них.
Визуальный конструктор документов	При работе с проектами настройки документов объектная модель Office используется для создания поверхности визуального конструктора, которую можно применять для интерактивного размещения компонентов. Как будет показано далее в этой главе, панели инструментов и меню, отображаемые в визуальном конструкторе, являются полностью функциональными.

Таблица 49.2. Средства дополнений уровня приложения

Средство	Описание
Специальная панель задач	Панели задач обычно стыкуются с одной стороной окна приложения Office и предоставляют широкий выбор функциональности. Например, Word имеет панель задач, используемую для манипулирования стилями. Как и с панелями действий, это обеспечивает значительную гибкость.
Межпрограммные коммуникации	Создав дополнение для одного приложения Office, эту функциональность легко распространить на другие дополнения. Можно, например, создать службу финансовых вычислений в Excel и затем использовать ее в Word, не создавая отдельное дополнение.
Области форм Outlook	Можно создать области форм, используемые в Outlook.

Таблица 49.3. Средства, полезные для всех проектов

Средство	Описание
Развертывание ClickOnce	Любой проект VSTO, создаваемый для конечных пользователей, может распространяться с помощью методов развертывания ClickOnce. Это позволяет пользователям получать обновления решений уровня документа и уровня приложения за счет обнаружения изменений в манифесте приложения.
Ленточное меню	Ленточные меню (ribbon menu) используются во всех приложениях Office, и VSTO включает два способа создания собственных ленточных меню. Можно либо применять XML для определения ленты, либо использовать Ribbon Designer. Обычно имеет смысл применять Ribbon Designer, поскольку это гораздо проще, хотя для обратной совместимости может быть выбран XML.

Основы проектов VSTO

Теперь, когда было показано, что включено в VSTO, стоит обратиться к рассмотрению практической стороны вопроса и узнать, как строятся проекты VSTO. Приемы, продемонстрированные в этом разделе, являются общими, и они применимы ко всем типам проектов VSTO. В этом разделе рассматриваются следующие темы:

- объектная модель Office;
- пространства имен VSTO;
- хост-элементы и хост-элементы управления;
- базовая структура проекта VSTO;
- класс `Globals`;
- обработка событий.

Объектная модель Office

Наборы приложений Office 2007 и Office 2010 обеспечивают свою функциональность через объектную модель COM. Эту объектную модель можно использовать непосредственно из VBA для управления практически любым аспектом функциональности Office. Объектная модель Office была предложена в Office 97 и затем развивалась по мере изменений функциональности Office.

В объектной модели Office существует громадное количество классов, причем некоторые из них используются во всем наборе приложений Office, а некоторые специфичны для индивидуальных приложений. Например, объектная модель Word 2007 включает коллекцию `Documents`, представляющую текущие загруженные объекты, каждый из которых представлен экземпляром объекта `Document`. В коде VBA можно обращаться к документам по имени или индексу и вызывать методы для выполнения операций над ними. Например, следующий код VBA закрывает документ по имени `My Document`, не сохраняя изменений:

```
Documents("My Document").Close SaveChanges:=wdDoNotSaveChanges
```

Объектная модель Office включает именованные константы (вроде `wdDoNotSaveChanges` в предыдущем коде) и перечисления, облегчающие их использование.

Пространства имен VSTO

В VSTO имеется коллекция пространств имен, содержащих типы, предназначенные для программирования с применением объектной модели Office. Многие типы в этих пространствах имен отображаются непосредственно на типы в объектной модели Office. Они доступны через сборки Office PIA во время проектирования и через встроенную информацию о типах — когда решение развернуто. Из-за встраивания типов для доступа к модели Office в основном будут использоваться интерфейсы. VSTO также содержит типы, которые не отображаются напрямую, либо не связаны с объектной моделью Office. Например, есть множество классов, которые служат для поддержки визуального конструктора в Visual Studio.

Типы, которые упаковывают или взаимодействуют с объектами в объектной модели Office, подразделяются на пространства имен. Пространства имен, которые используемые при разработке для Office, перечислены в табл. 49.4.

Таблица 49.4. Пространства имен для Office

Пространство имен	Описание
`Microsoft.Office.Core`, `Microsoft.Office.Interop.*`	Эти пространства имен содержат тонкие оболочки вокруг классов PIA, обеспечивая базовую функциональность для работы с классами Office. Для каждого продукта Office предусмотрено несколько вложенных пространств внутри `Microsoft.Office.Interop`.
`Microsoft.Office.Tools`	Это пространство имен содержит общие типы, предоставляющие функциональность VSTO, и базовые классы для многих классов во вложенных пространствах имен. Например, это пространство включает классы, необходимые для реализации панелей действий в настройках уровня документа, и базовый класс для дополнений уровня приложения.
`Microsoft.Office.Tools.Excel`, `Microsoft.Office.Tools.Excel.*`	Эти пространства имен содержат типы, необходимые для взаимодействия с приложением Excel и документами Excel.
`Microsoft.Office.Tools.Outlook`	Это пространство имен содержит типы, необходимые для взаимодействия с приложением Outlook.
`Microsoft.Office.Tools.Ribbon`	Это пространство имен содержит типы, необходимые для работы с ленточными меню и для создания собственных ленточных меню.

Окончание табл. 49.4

Пространство имен	Описание
`Microsoft.Office.Tools.Word`, `Microsoft.Office.Tools.Word.*`	Эти пространства имен содержат типы, необходимые для взаимодействия с приложением Word и документами Word.
`Microsoft.VisualStudio.Tools.*`	Это пространство имен обеспечивает инфраструктуру VSTO, с которой вы имеете дело при разработке решений VSTO в VS.

Хост-элементы и хост-эелементы управления

Хост-элементы (host items) и хост-элементы управления (host controls) — это интерфейсы, расширенные для облегчения настройкам уровня документа взаимодействовать с документами Office. Эти классы упрощают код, представляя события в стиле .NET, и являются полностью управляемыми. Часть "хост" в их наименовании указывает на тот факт, что классы упаковывают и расширяют встроенные объекты Office.

Часто при использовании хост-элементов вы обнаружите, что вместе с ними необходимо также применять лежащие в основе interop-типы. Например, если вы создаете новый документ Word, то принимаете ссылку на interop-тип документа Word вместо создания нового документа хост-элемента Word. Об этом следует помнить и писать код соответственно.

Существуют хост-элементы и хост-элементы управления для настройки уровня документа, как для Word, так и для Excel.

Word

Для Word предусмотрен единственный хост-элемент — `Microsoft.Office.Tools.Word.Document`. Он представляет документ Word. Как и следовало ожидать, этот класс имеет невероятно большое количество методов и свойств, которые можно использовать для взаимодействия с документами Word.

Существует 12 хост-элементов управления для Word, которые перечислены в табл. 49.5, и все они находятся в пространстве имен `Microsoft.Office.Tools.Word`.

Таблица 49.5. Хост-элементы управления для Word

Элемент управления	Описание
`Bookmark`	Этот элемент управления представляет местоположение внутри документа Word. Это может быть единичное местоположение или диапазон символов.
`XMLNode`, `XmlNodes`	Эти элементы управления используются, когда документ имеет присоединенную XML-схему. Это позволяет ссылаться на содержимое документа через местоположение узла XML содержимого. С помощью этих элементов управления можно также манипулировать XML-структурой документа.
`ContentControl`	Данный класс является базовым для остальных восьми элементов управления в настоящей таблице и позволяет иметь дело с элементами управления содержимым Word. Элемент управления содержимым — это элемент, представляющий содержимое в качестве элемента управления или обеспечивающий функциональность вдобавок к той, что предлагается простым текстом в документе.

Окончание табл. 49.5

Элемент управления	Описание
`BuildingBlockGallery-ContentControl`	Этот элемент управления позволяет добавлять и манипулировать строительными блоками документа, такими как форматированные таблицы, страницы обложки и т.п.
`ComboBoxContentControl`	Этот элемент управления представляет содержимое, сформатированное как комбинированный список.
`DatePickerContentControl`	Этот элемент управления представляет содержимое, сформатированное как указатель даты.
`DropDownListContentControl`	Этот элемент управления представляет содержимое, сформатированное как раскрывающийся список.
`GroupContentControl`	Этот элемент управления представляет содержимое, являющееся сгруппированной коллекцией других элементов содержимого, включая текст и прочие элементы управления содержимым.
`PictureContentControl`	Этот элемент управления представляет графическое изображение.
`RichTextContentControl`	Этот элемент управления представляет блок форматированного текстового содержимого.
`PlainTextContentControl`	Этот элемент управления представляет блок простого текстового содержимого.

Excel

Для Excel предусмотрены три хост-элемента и четыре хост-элемента управления, и все они содержатся в пространстве имен `Microsoft.Office.Tools.Excel`.

Хост-элементы Excel перечислены в табл. 49.6.

Таблица 49.6. Хост-элементы Excel

Хост-элемент	Описание
`Workbook`	Этот хост-элемент представляет целую рабочую книгу Excel, которая может содержать множество рабочих таблиц и панелей графиков.
`Worksheet`	Этот хост-элемент используется для индивидуальных рабочих таблиц внутри рабочей книги.
`Chartsheet`	Этот хост-элемент используется для индивидуальных панелей графиков внутри рабочей книги.

Хост-элементы управления Excel описаны в табл. 49.7.

Таблица 49.7. Хост-элементы управления Excel

Элемент управления	Описание
`Chart`	Этот элемент управления представляет график, встроенный в рабочую таблицу.
`ListObject`	Этот элемент управления представляет список в рабочей таблице.
`NamedRange`	Этот элемент управления представляет именованный диапазон в рабочей таблице.
`XmlMappedRange`	Этот элемент управления используется, когда электронная таблица Excel имеет присоединенную схему, и служит для манипуляций диапазонами, отображаемыми на элементы схемы XML.

Базовая структура проекта VSTO

Когда вы впервые создаете проект VSTO, файлы, с которых вы начинаете, варьируются в зависимости от типа проекта, хотя обладают некоторыми общими свойствами. В этом разделе будет показано, из чего состоит проект VSTO.

Структура проекта настройки уровня документа

При создании проекта настройки уровня документа в Solution Explorer появляется элемент, представляющий тип документа. Это может быть:

- файл `.docx` для документа Word;
- файл `.dotx` для шаблона Word;
- файл `.xlsx` для рабочей книги Excel;
- файл `.xltx` для шаблона Excel.

Каждый из них имеет представление визуального конструктора и файл кода, которые можно увидеть, развернув элемент в Solution Explorer. Шаблоны Excel также включают подэлементы, представляющие рабочую книгу в целом и каждую электронную таблицу в этой книге по отдельности. Такая структура позволяет обеспечить специальную функциональность на уровне отдельной электронной таблицы или рабочей книги в целом.

Если просмотр скрытых файлов в одном из этих проектов включен, можно увидеть также и несколько файлов визуального конструктора, в которых находится сгенерированный шаблоном код. Каждый элемент документа Office имеет ассоциированный класс из пространства имен VSTO, и классы в файлах кода наследуются от этих классов. Эти классы определены как частичные, так что специальный код отделен от кода, сгенерированного визуальным конструктором, подобно структуре приложений Windows Forms.

Например, шаблон документа Word представляет класс, унаследованный от класса `Microsoft.Office.Tools.Word.Document`. Этот класс предоставляет хост-элемент `Document` через свойство `Base`; код содержится в `ThisDocument.cs`, как показано ниже.

```
using System;
using System.Collections.Generic;
using System.Data;
using System.Linq;
using System.Text;
using System.Windows.Forms;
using System.Xml.Linq;
using Microsoft.VisualStudio.Tools.Applications.Runtime;
using Office = Microsoft.Office.Core;
using Word = Microsoft.Office.Interop.Word;

namespace WordDocument1
{
  public partial class ThisDocument
  {
    private void ThisDocument_Startup(object sender, System.EventArgs e)
    {
    }
    private void ThisDocument_Shutdown(object sender, System.EventArgs e)
    {
    }
    #region VSTO Designer generated code
    /// <summary>
    /// Required method for Designer support - do not modify
    /// the contents of this method with the code editor.
    /// </summary>
```

```
    private void InternalStartup()
    {
      this.Startup += new System.EventHandler(ThisDocument_Startup);
      this.Shutdown += new System.EventHandler(ThisDocument_Shutdown);
    }
    #endregion
  }
}
```

Сгенерированный шаблоном код включает псевдонимы для двух главных пространств имен, которые используются при создании настройки уровня документа для Word, `Microsoft.Office.Core` — для главных классов VSTO Office и `Microsoft.Office.Interop.Word` — для классов, специфичных для Word. Обратите внимание, что если вы хотите применять хост-элементы управления Word, то должны также добавить оператор `using` для пространства имен `Microsoft.Office.Tools.Word`. Сгенерированный шаблоном код также определяет два обработчика событий, которые можно использовать для выполнения кода, когда документ загружается или выгружается — `ThisDocument_Startup()` и `ThisDocument_Shutdown()`.

Все типы проектов настройки уровня документа обладают сходной структурой в файле кода (или файлах кода в случае Excel). В них определяются псевдонимы пространств имен и обработчики разнообразных событий `Startup` и `Shutdown`, определенных в классах VSTO. Начиная отсюда, вы добавляете диалоговые окна, панели действий, ленточные элементы управления, обработчики событий и специальный код, определяющий поведение настройки.

В настройке уровня документа можно также настраивать документ или документы с помощью визуального конструктора документов. В зависимости от типа создаваемого решения, это может предусматривать добавление стандартного содержимого в шаблоны, интерактивного содержимого в документы или чего-либо еще. Визуальные конструкторы — это на самом деле хост-версии приложений Office, и их можно использовать для ввода содержимого, как это делается в самих приложениях. Однако можно также добавлять в документы элементы управления, такие как хост-элементы управления и элементы управления Windows Forms, а также код, сопровождающий эти элементы управления.

Структура проекта дополнения уровня приложений

В случае создания дополнения уровня приложения в Solution Explorer документа или документов не существует. Вместо этого вы увидите там элемент, представляющий приложение, для которого создается дополнение. Раскрыв этот элемент, вы увидите файл по имени `ThisAddIn.cs`. Этот файл содержит частичное определение класса по имени `ThisAddIn`, которое предоставляет точку входа в дополнение. Этот класс унаследован от `Microsoft.Office.Tools.AddInBase`, который предоставляет код функциональности дополнения.

Например, код, сгенерированный шаблоном дополнения Word, выглядит следующим образом:

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using System.Xml.Linq;
using Word = Microsoft.Office.Interop.Word;
using Office = Microsoft.Office.Core;
using Microsoft.Office.Tools.Word;
namespace WordAddIn1
{
  public partial class ThisAddIn
  {
```

```
    private void ThisAddIn_Startup(object sender, System.EventArgs e)
    {
    }
    private void ThisAddIn_Shutdown(object sender, System.EventArgs e)
    {
    }
    #region VSTO generated code
    /// <summary>
    /// Required method for Designer support - do not modify
    /// the contents of this method with the code editor.
    /// </summary>
    private void InternalStartup()
    {
      this.Startup += new System.EventHandler(ThisAddIn_Startup);
      this.Shutdown += new System.EventHandler(ThisAddIn_Shutdown);
    }
    #endregion
  }
}
```

Как видите, эта структура очень похожа на структуру, используемую в настройке уровня документа. Она включает псевдонимы для тех же пространств имен `Microsoft.Office.Core` и `Microsoft.Office.Interop.Word` и предоставляет обработчики событий `Startup` и `Shutdown` (`ThisAddIn_Startup()` и `ThisAddIn_Shutdown()`). Эти события слегка отличаются от таких же у документов, поскольку возникают, когда загружается и выгружается дополнение, а не открывается и закрывается индивидуальный документ.

Настройка дополнений уровня приложения во многом подобна настройке уровня документа и предусматривает добавление ленточных элементов управления, панелей задач и дополнительного кода.

Класс `Globals`

Во всех типах проектов VSTO определен класс по имени `Globals`, который обеспечивает глобальный доступ к следующим вещам.

- Для настройки уровня документа — ко всем документам решения. Это обеспечивается членами с именами, соответствующими именам классов документов — например, `Globals.ThisWorkbook` и `Globals.Sheet1`.
- Для дополнений уровня приложения — к объекту дополнения. Это обеспечивает `Globals.ThisAddIn`.
- Для проектов дополнений Outlook — ко всем областям Outlook.
- Ко всем лентам в решении — через свойство `Globals.Ribbons`.
- К интерфейсу, унаследованному от `Microsoft.Office.Tools.Factory`, который представляет служебные методы, специфичные для типа проекта — через свойство `Factory`.

"За кулисами" класс `Globals` создается через серию частичных определений в различных файлах кода решения, обслуживаемых визуальным конструктором. Например, рабочая таблица по умолчанию `Sheet1` в проекте Excel Workbook включает следующий сгенерированный визуальным конструктором код:

```
internal sealed partial class Globals
{
  private static Sheet1 _Sheet1;
  internal static Sheet1 Sheet1
  {
    get
```

```
    {
      return _Sheet1;
    }
    set
    {
      if ((_Sheet1 == null))
      {
        _Sheet1 = value;
      }
      else
      {
        throw new System.NotSupportedException();
      }
    }
  }
}
```

Этот код добавляет член `Sheet1` в класс `Globals`.

Класс `Globals` также позволяет трансформировать interop-тип в VSTO, используя метод `Globals.Factory.GetVstoObject()`. Он получает VSTO-интерфейс `Workbook`, например, от интерфейса `Microsoft.Office.Interop.Excel.Workbook`. Имеется также метод `Glbal.Factory.HasVstoObject()`, который служит для проверки возможности выполнения такой трансформации.

Обработка событий

Ранее в этой главе было показано, каким образом классы хост-элемента и хост-элемента управления предоставляют события, которые можно обработать. К сожалению, это не касается классов interop. Существует несколько событий, которые можно использовать, но для большей части вы сочтете трудным создание управляемого событиями решения, применяющего эти события. Чаще всего для того, чтобы реагировать на события, необходимо сосредоточиться на событиях, предоставленных хост-элементами и хост-элементами управления.

Очевидная проблема здесь состоит в том, что не существует хост-элементов или хост-элементов управления для проектов дополнений уровня приложения. К сожалению, во время использования VSTO с этим придется мириться. Однако наиболее распространенные события, которые, скорее всего, будут прослушиваться в дополнениях — это те, что ассоциированы с ленточным меню и взаимодействием с панелями задач. Ленты проектируются интегрированным конструктором лент, и можно предусмотреть реагирование на любые события, сгенерированные лентой, чтобы сделать этот элемент управления интерактивным. Панели задач обычно реализованы как пользовательские элементы управления Windows Forms (хотя можете применять WPF), и здесь можно работать с событиями Windows Forms. Это значит, что вы не часто столкнетесь с ситуациями, в которых не найдется доступного события для нужной функциональности.

Когда необходимо использовать какое-то interop-событие Office, вы обнаружите, что такие события доступны через интерфейсы этих объектов. Рассмотрим проект Word Add-In. Класс `ThisAddIn` в этом проекте предоставляет свойство по имени `Application`, через которое можно получить ссылку на приложение Office. Это свойство имеет тип `Microsoft.Office.Interop.Word.Application` и представляет события через интерфейс `Microsoft.Office.Interop.Word.ApplicationEvents4_Event`. В данном интерфейсе имеется всего 29 событий (что на самом деле не так уж и много для сложного приложения вроде Word). Вы можете обработать, например, событие `DocumentBeforeClose`, возникающее в ответ на запрос закрытия документа Word.

Построение решений VSTO

В предыдущих разделах объяснялось, что собой представляют проекты VSTO, как они структурированы и какие средства можно использовать в различных типах проектов. В этом разделе мы рассмотрим реализацию решений VSTO.

На рис. 49.4 показан набросок структуры решения настройки уровня документа.

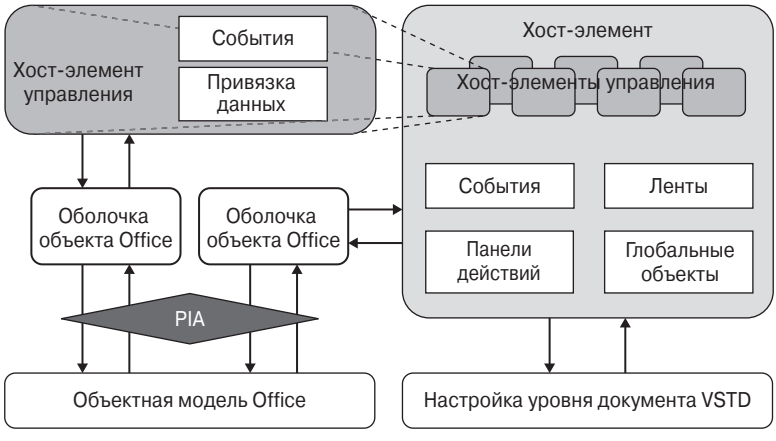

Рис. 49.4. *Структура решения настройки уровня документа*

При настройке уровня документа вы будете взаимодействовать как минимум, с одним хост-элементом, который обычно содержит несколько хост-элементов управления. Оболочку объекта Office можно использовать непосредственно, но большей частью вы будете получать доступ к объектной модели Office и ее функциональности через хост-элементы и хост-элементы управления.

В коде будут применяться события хост-элементов и хост-элементов управления, привязка данных, ленточные меню, панели действий и глобальные объекты.

На рис. 49.5. представлен эскиз структуры решения дополнений уровня приложения.

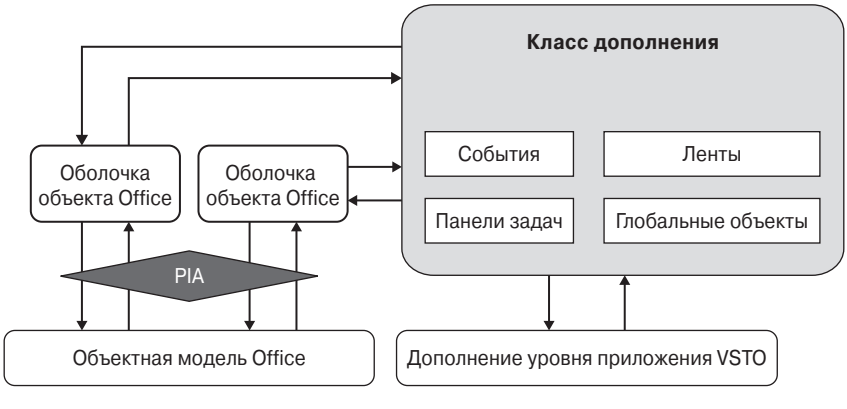

Рис. 49.5. *Структура решений дополнений уровня приложения*

В этой несколько упрощенной модели, скорее всего, будут напрямую использоваться тонкие оболочки вокруг объектов Office — по крайней мере, через класс дополнения, ин-

капсулирующий решение. Также в коде применяются события, представленные классом дополнения, ленточные меню, панели задач и глобальные объекты.

В этом разделе мы рассмотрим оба эти типа приложений, а также раскроем следующие темы:

- управление дополнениями уровня приложений;
- взаимодействие с приложениями и документами;
- настройка пользовательского интерфейса.

Управление дополнениями уровня приложений

Одна из первых вещей, с которыми вы столкнетесь при создании дополнения уровня приложения — это то, что Visual Studio заботится о выполнении всех шагов, необходимых для регистрации дополнения в приложении Office. Это значит, что добавляются необходимые записи в реестр, так что приложение Office после запуска автоматически находит и загружает вашу сборку. Если впоследствии понадобится добавить или удалить дополнение, то для этого нужно будет обратиться к настройкам приложения Office или же непосредственно манипулировать реестром.

Например, в Word понадобится открыть меню Office Button, щелкнуть на кнопке Word Options (Параметры Word) и перейди на вкладку Add-Ins (Дополнения), которая показана на рис. 49.6.

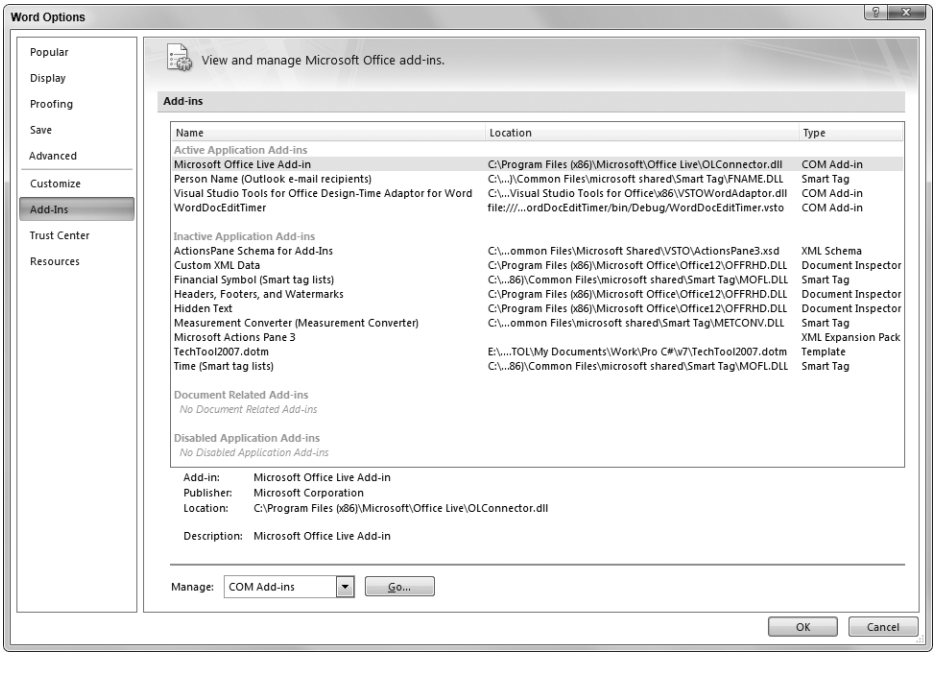

Рис. 49.6. *Настройки дополнений для Word*

На рис. 49.6. показано дополнение, которое было создано с помощью VSTO — `WordDocEditTimer`. Чтобы добавить или удалить дополнение, необходимо выбрать пункт COM Add-Ins (Дополнения COM) в раскрывающемся списке Manage (Управление) и щелкнуть на кнопке Go (Перейти). Появится диалоговое окно COM Add-Ins (Дополнения COM), показанное на рис. 49.7.

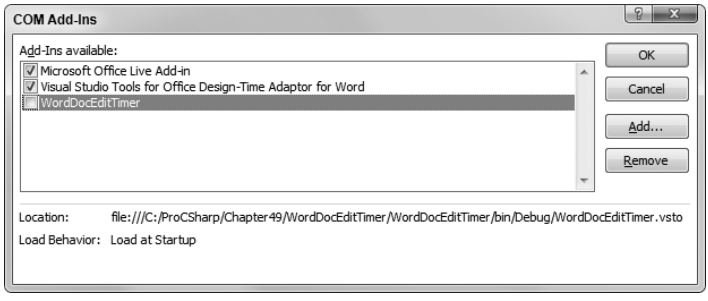

Рис. 49.7. *Диалоговое окно* COM Add-Ins

В этом окне можно выгрузить дополнения, сняв отметки с соответствующих им флажков, а также добавить новые или удалить старые с помощью кнопок Add (Добавить) и Remove (Удалить).

Взаимодействие с приложениями и документами

Какой бы тип приложения ни создавался, вы всегда захотите взаимодействовать с приложением-хостом и/или документами, загруженными в него. Частично это включает использование настройки пользовательского интерфейса, о которой вы узнаете в следующем разделе. Однако может также понадобиться отслеживать документы внутри приложения, что означает необходимость обработки некоторых событий объектной модели Office. Например, для отслеживания документов в Word необходимы обработчики следующих событий интерфейса `Microsoft.Office.Interop.Word.ApplicationEvents4_Event`:

1. `DocumentOpen` — возникает при открытии документа;
2. `NewDocument` — возникает при создании нового документа;
3. `DocumentBeforeClose` — возникает, когда документ сохраняется.

Кроме того, когда Word стартует вначале, он имеет загруженный документ, которым будет либо пустой новый документ, либо документ, который был загружен.

В код для настоящей главы входит пример по имени `WordDocEditTimer`, который поддерживает список времен редактирования документов Word. Часть функциональности этого приложения заключается в мониторинге загруженных документов — по причинам, объясняемым ниже. Поскольку в этом примере также применяется специальная панель задач и ленточное меню, мы рассмотрим его после раскрытия этих тем.

Обращаться к текущему активному документу в Word можно через свойство `ThisAddIn.Application.ActiveDocument`, а к коллекции открытых документов — через свойство `ThisAddIn.Application.Documents`. Аналогичные свойства есть и в других приложениях Office с многодокументным интерфейсом (MDI). Для манипулирования различными свойствами документов применяются свойства, предоставляемые, например, классом `Microsoft.Office.Interop.Word.Document`.

Здесь следует отметить один момент — объем классов и членов классов, с которыми придется иметь дело во время разработки решений VSTO, в действительности огромен. Пока вы не приобрели опыт работы с ними, часто бывает трудно найти нужное средство. Например, совсем не очевидно, по какой причине в Word текущий активный выбор доступен не через активный документ, а через приложение (через свойство `ThisAddIn.Application.Selection`).

Выделение удобно для вставки, чтения или замены текста через свойство `Range`, например:

```
ThisAddIn.Application.Selection.Range.Text = "Inserted text";
```

К сожалению, в этой главе недостаточно места, чтобы полностью описать библиотеку объектов. Вы узнаете о ней лишь то, что будет существенно по ходу дискуссии.

Настройка пользовательского интерфейса

Наиболее важный аспект последнего выпуска VSTO связан с гибкостью настройки пользовательского интерфейса. Вы получаете возможность добавлять содержимое в любое из существующих ленточных меню, добавлять совершенно новые ленточные меню, настраивать панели задач, добавляя к ним новые задачи, добавлять новые панели задач и интегрировать формы и элементы управления Windows Forms и WPF.

Ниже рассматриваются все перечисленные возможности.

Ленточные меню

Ленточные меню могут быть добавлены к любому проекту VSTO, рассматриваемому в этой главе. Для добавления ленточного меню используется визуальный конструктор, показанный на рис. 49.8.

Рис. 49.8. *Визуальный конструктор ленточного меню*

Этот конструктор позволяет настраивать ленту, добавляя элементы управления к кнопочному меню Office (на рис. 49.8 вверху слева), а также к группам ленты. Можно также добавлять дополнительные группы.

Классы, используемые в лентах, находятся в пространстве имен `Microsoft.Office.Tools.Ribbon`. Оно включает класс ленты, от которого наследуется создаваемая лента –`RibbonBase`. Этот класс может содержать объекты `RibbonTab`, каждый из которых включает содержимое одной вкладки. Вкладки содержат объекты `RibbonGroup`, такие как группа `group1` на рис. 49.8. Эти вкладки могут содержать широкое разнообразие элементов управления.

Группы на вкладке можно позиционировать на совершенно новой вкладке или на одной из существующих вкладок целевого приложения Office. Где именно появится группа, определяется свойством `RibbonTab.ControlId`. Это свойство имеет свойство `ControlIdType`, которое можно установить в `RibbonControlIdType.Custom` или `RibbonControlIdType.Office`. В случае `Custom` можно также установить `RibbonTab.ControlId.CustomId` в строковое значение — идентификатор вкладки. Идентификатор может быть произвольным. Однако если для `ControlType` выбрано значение `Office`, то `RibbonTab.ControlId.OfficeId` должно быть установлено в строковое значение, соответствующее одному из идентификаторов, которые используются в целевом продукте Office. Например, для Excel это свойство можно установить в `TabHome`, чтобы добавлять группы на вкладку Home (Домашняя), в `TabInsert` — для добавления на вкладку Insert (Вставка), и т.д. Значением по умолчанию для дополнений является `TabAddIns`, которое разделяется всеми дополнениями.

Доступно много вкладок, особенно в Outlook; полный список идентификаторов доступен для загрузки по адресу `http://www.microsoft.com/downloads/details.aspx?familyid=4329D9E9-4D11-46A5-898D-23E4F331E9AE&displaylang=en`.

Приняв решение относительно того, куда помещать группы ленты, к ним можно добавлять любые элементы управления, перечисленные в табл. 49.8.

Таблица 49.8. Элементы управления для помещения на ленты

Элемент управления	Описание
`RibbonBox`	Это контейнерный элемент управления, который может использоваться для размещения других элементов в группе. Вы можете располагать их в `RibbonBox` горизонтально или вертикально, изменяя свойство `BoxStyle` на `RibbonBoxStyle.Horizontal` или `RibbonBoxStyle.Vertical`.
`RibbonButton`	Этот элемент управления может использоваться для добавления в группу маленькой или большой кнопки с текстовой меткой или без нее. Управление размером осуществляется установкой свойства `ControlSize` в `RibbonControlSize.RibbonControlSizeLarge` или `RibbonControlSize.RibbonControlSizeRegular`. Кнопка имеет обработчик события `Click`, который служит для реагирования на взаимодействие. Можно также установить специальное графическое изображение или одно из изображений, доступных в Office (рассматривается далее в главе).
`RibbonButtonGroup`	Это контейнерный элемент управления, представляющий группу кнопок. Он может содержать элементы `RibbonButton`, `RibbonGallery`, `RibbonMenu`, `RibbonSplitButton` и `RibbonToggleButton`.
`RibbonCheckBox`	Элемент управления типа флажка с событием `Click` и свойством `Checked`.
`RibbonComboBox`	Элемент управления типа комбинированного списка (сочетание текстового поля с раскрывающимся списком элементов). Используйте свойство `Items` для доступа к элементам, свойство `Text` — для доступа к введенному тексту и событие `TextChanged` — для реагирования на изменение.
`RibbonDropDown`	Контейнер, который может содержать в свойствах `Items` и `Buttons` элементы `RibbonDropDownItem` и `RibbonButton`. Кнопки и элементы форматируются в раскрывающийся список. Для реагирования на взаимодействие используется событие `SelectionChanged`.
`RibbonEditText`	Текстовое поле, которое пользователь может применять для ввода и редактирования текста в свойстве `Text`. Этот элемент имеет событие `TextChanged`.
`RibbonGallery`	Как и `RibbonDropDown`, этот элемент управления может содержать в свойствах `Items` и `Buttons` элементы `RibbonDropDownItem` и `RibbonButton`. Этот элемент управления использует события `Click` и `ButtonClick` вместо события `SelectionChanged`, применяемого в `RibbonDropDown`.
`RibbonLabel`	Простая текстовая метка, устанавливается с помощью свойства `Label`.
`RibbonMenu`	Всплывающее меню, которое в представлении визуального конструктора можно наполнять другими элементами управления, такими как `RibbonButton` и вложенными элементами управления `RibbonMenu`. Для реагирования на взаимодействие служат события элементов меню.

Окончание табл. 49.8

Элемент управления	Описание
`RibbonSeparator`	Простой разделитель, используемый для настройки компоновки элементов управления в группах.
`RibbonSplitButton`	Кнопка, комбинирующая `RibbonButton` или `RibbonToggleButton` с `RibbonMenu`. Стиль кнопки устанавливается в `ButtonType` и может быть `RibbonButtonType.Button` или `RibbonButtonType.ToggleButton`. Для реагирования на взаимодействие служит событие `Click` главной кнопки или события `Click` индивидуальных кнопок в меню.
`RibbonToggleButton`	Кнопка, которая может быть в выбранном или невыбранном состоянии, что определяется свойством `Checked`. Этот элемент управления также поддерживает событие `Click`.

Если установить свойство `DialogBoxLauncher` группы, в нижнем правом углу группы появится значок. Его можно использовать для отображения диалогового окна, для открытия панели задач или для выполнения любого другого необходимого действия. Значок добавляется и удаляется через меню GroupView Tasks (Задачи группы), как показано на рис. 49.9, где также видны другие элементы управления из предыдущей таблицы, как они выглядят на ленте в представлении визуального конструктора.

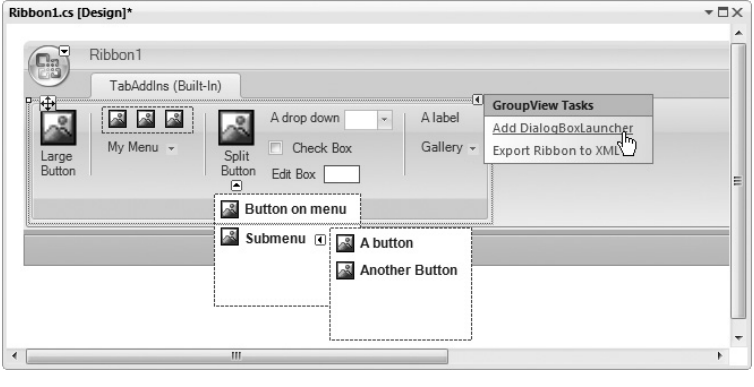

Рис. 49.9. *Меню GroupView Tasks*

Чтобы задать изображение для элемента управления, например, `RibbonButton`, необходимо либо установить в свойстве `Image` специальное изображение, а в `ImageName` — имя изображения (чтобы можно было оптимизировать загрузку изображений в обработчике событий `OfficeRibbon.LoadImage`), либо воспользоваться одним из встроенных в Office изображений. Для этого укажите в свойстве `OfficeImageId` идентификатор нужного изображения.

Для использования доступно множество изображений; галерею значков можно загрузить по адресу `http://www.microsoft.com/downloads/details.aspx?familyid=12b99325-93e8-4ed4-8385-74d0f7661318&displaylang=en`. На рис. 49.10 показаны примеры.

На рис. 49.10 можно видеть вкладку ленты Developer (Разработчик), которая включается на вкладке Popular (Популярные) диалогового окна Excel Options (Параметры Excel), доступного через меню Office Button.

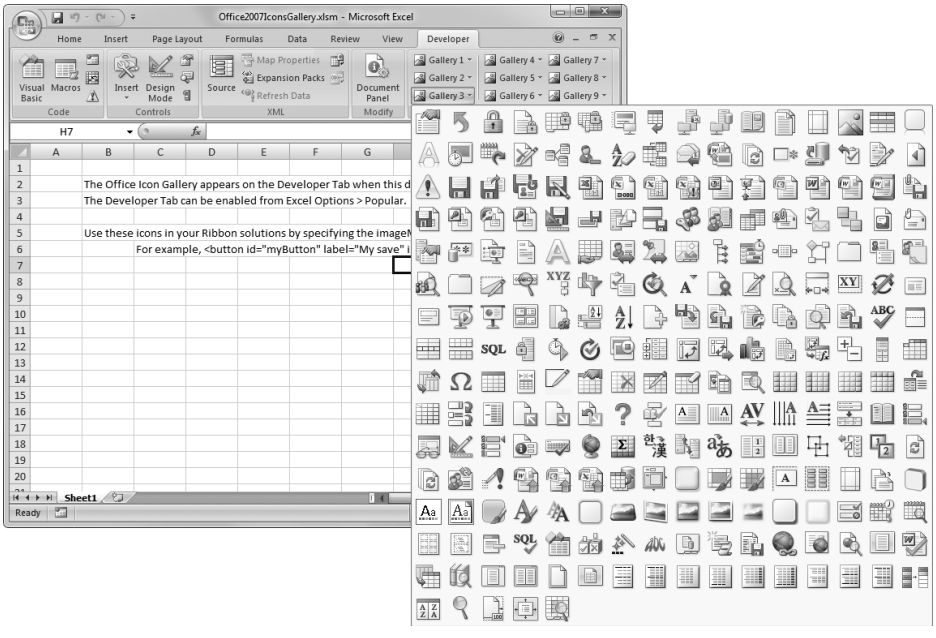

***Рис. 49.10**. Галерея значков*

Щелчок на изображении приводит к открытию диалогового окна, сообщающего его идентификатор, как показано на рис. 49.11.

***Рис. 49.11**. Диалоговое окно с идентификатором изображения*

Визуальный конструктор ленты исключительно гибок, и с его помощью можно предоставить почти любую функциональность для ленты Office. Тем не менее, для дальнейшей настройки пользовательского интерфейса потребуется использовать панели действий и задач.

Панели действий и специальные панели задач

Панели действий и задач служат для отображения содержимого, стыкованного с областью панели задач в интерфейсе приложения Office. Панели задач используются в дополнениях уровня приложения, а панели действий — при настройке уровня документа. И панели действий, и панели задач должны наследоваться от объектов `UserControl`, а это значит, что пользовательский интерфейс создается с помощью Windows Forms. Можно также использовать WPF, если разместить форму WPF в элементе управления `ElementHost` на `UserControl`.

Для добавления панели действий к документу в настройке уровня документа необходимо добавить экземпляр класса панели действия в коллекцию `Controls` свойства `ActionsPane` документа.

Например:

```
public partial class ThisWorkbook
{
  private UserControl1 actionsPane;
  private void ThisWorkbook_Startup(object sender, System.EventArgs e)
  {
    Workbook wb = Globals.Factory.GetVstoObject(this.Application.ActiveWorkbook);
    wb.AcceptAllChanges();
    actionsPane = new UserControl1();
    this.ActionsPane.Controls.Add(actionsPane);
  }
  ...
}
```

Этот код добавляет панель действий при загрузке документа (в данном случае — рабочей книги Excel). Это также можно сделать, например, в обработчике события кнопки ленты.

В проектах дополнений уровня приложения специальные панели задач добавляются методом `ThisAddIn.CustomTaskPanes.Add()`. Этот метод также позволяет назначить имя окну задачи. Например:

```
public partial class ThisAddIn
{
  Microsoft.Office.Tools.CustomTaskPane taskPane;
  private void ThisAddIn_Startup(object sender, System.EventArgs e)
  {
    taskPane = this.CustomTaskPanes.Add(new UserControl1(), "My Task Pane");
    taskPane.Visible = true;
  }
  ...
}
```

Обратите внимание, что метод `Add()` возвращает объект типа `Microsoft.Office.Tools.CustomTaskPane`. Обращение к самому элементу управления производится через свойство `Control` этого объекта. Для управления панелью задач можно также использовать другие свойства, предоставленные этим типом — например, свойство `Visible`, как показано в предыдущем коде.

И здесь стоит упомянуть о слегка необычной особенности приложений Office, в частности, о разнице между Word и Excel. По историческим причинам, хотя и Word, и Excel являются приложениями с многодокументным интерфейсом (MDI), способ размещения документов в них отличается. В Word каждый документ имеет уникальное родительское окно. В Excel все документы разделяют одно и то же родительское окно.

При вызове метода `CustomTaskPanes.Add()` поведение по умолчанию заключается в добавлении панели задач к текущему активному окну. В Excel это означает, что каждый документ будет отображать панель задач, поскольку все они имеют общее родительское окно. В Word ситуация иная. Если вы хотите, чтобы панель задач появлялась в каждом документе, то должны добавить ее в каждое окно, содержащее документ.

Чтобы добавить панель задач к определенному документу, методу `Add()` в третьем параметре передается экземпляр класса `Microsoft.Office.Interop.Word.Window`. С помощью свойства `Microsoft.Office.Interop.Word.Document.ActiveWindow` можно получить окно, ассоциированное с документом.

В следующем разделе будет показано, как это реализовать на практике.

Пример приложения

Как упоминалось в предыдущем разделе, в состав примеров для этой главы включено приложение по имени `WordDocEditTimer`, которое поддерживает список моментов редактирования документов Word. В этом разделе мы детально рассмотрим его код, поскольку он иллюстрирует все показанное до сих пор и содержит ряд полезных советов.

Главная операция в этом приложении заключается в том, что когда документ создается или загружается, то запускается таймер, привязанный к имени документа. Если вы закрываете документ, то таймер для данного документа приостанавливается. Если вы открываете документ, для которого уже был создан таймер, он возобновляется. Также, если используется операция Save As (Сохранить как) для сохранения документа под другим именем, таймер обновляется с учетом нового имени файла.

Это приложение представляет собой дополнение уровня приложения и использует специальную панель задач и ленточное меню. Ленточное меню содержит кнопку, которую можно применять для включения и выключения панели задач, и флажок, позволяющий приостановить таймер для текущего активного документа. Группа, содержащая эти элементы управления, добавляется на вкладку Home ленты. Панель задач отображает список активных таймеров.

Пользовательский интерфейс показан на рис. 49.12.

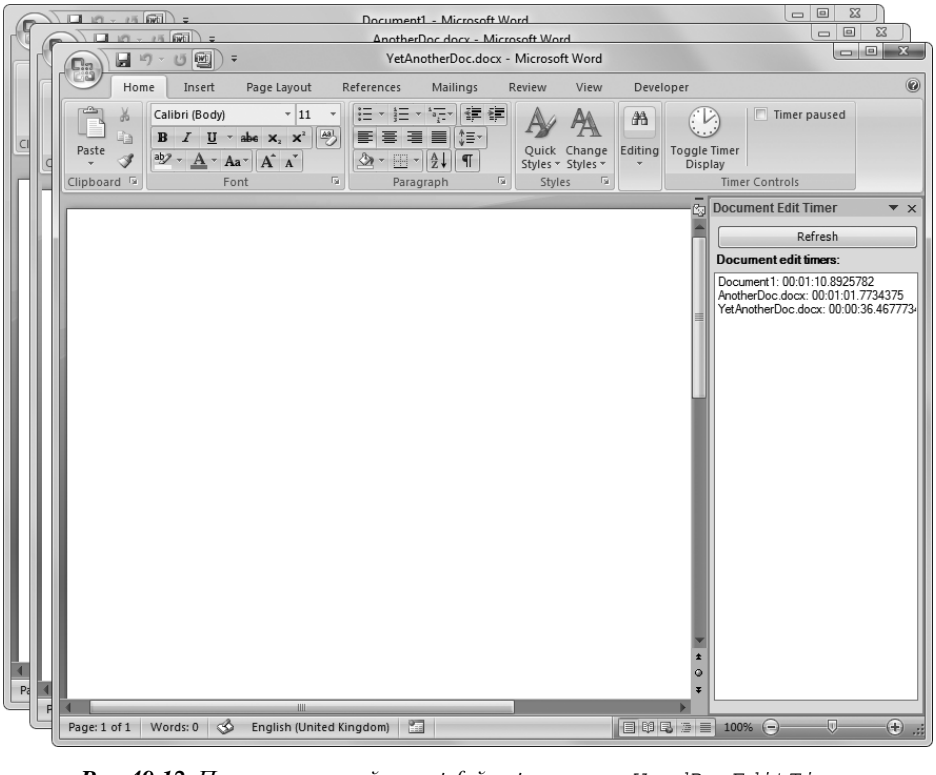

Рис. 49.12. *Пользовательский интерфейс приложения* `WordDocEditTimer`

Таймер поддерживается классом DocumentTimer:

```
public class DocumentTimer
{
  public Word.Document Document { get; set; }
  public DateTime LastActive { get; set; }
  public bool IsActive { get; set; }
  public TimeSpan EditTime { get; set; }
}
```

Фрагмент кода ***DocumentTimer.cs***

Здесь сохраняется ссылка на интерфейс Microsoft.Office.Interop.Word.Document, а также общее время редактирования, признак активности таймера и время его последней активизации. Класс ThisAddIn поддерживает коллекцию этих объектов, ассоциированных с именами документов:

```
public partial class ThisAddIn
{
  private Dictionary<string, DocumentTimer> documentEditTimes;
```

Фрагмент кода ***ThisAddin.cs***

Таким образом, каждый таймер может быть найден по ссылке на документ или по имени документа. Это необходимо, поскольку ссылка на документ позволяет отслеживать изменения имени документа (не существует события, которое можно было бы использовать для этого), а имена документов позволяют отслеживать закрытые и вновь открытые документы.

Класс ThisAddIn также поддерживает список объектов CustomTaskPane (как упоминалось ранее, он необходим каждому окну Word):

```
private List<Tools.CustomTaskPane> timerDisplayPanes;
```

При запуске дополнения метод ThisAddIn_Startup() решает несколько задач. Сначала он инициализирует две коллекции:

```
private void ThisAddIn_Startup(object sender, System.EventArgs e)
{
  // Инициализация таймеров и панелей отображения
  documentEditTimes = new Dictionary<string, DocumentTimer>();
  timerDisplayPanes = new List<Microsoft.Office.Tools.CustomTaskPane>();
```

Затем он добавляет несколько обработчиков событий через интерфейс Application Events4_Event:

```
  // Добавить обработчики событий
  Word.ApplicationEvents4_Event eventInterface = this.Application;
  eventInterface.DocumentOpen += new Microsoft.Office.Interop.Word
    .ApplicationEvents4_DocumentOpenEventHandler(eventInterface_DocumentOpen);
  eventInterface.NewDocument += new Microsoft.Office.Interop.Word
    .ApplicationEvents4_NewDocumentEventHandler(eventInterface_NewDocument);
  eventInterface.DocumentBeforeClose += new Microsoft.Office.Interop.Word
    .ApplicationEvents4_DocumentBeforeCloseEventHandler(
      eventInterface_DocumentBeforeClose);
  eventInterface.WindowActivate += new Microsoft.Office.Interop.Word
    .ApplicationEvents4_WindowActivateEventHandler(
      eventInterface_WindowActivate);
```

Эти обработчики событий используются для мониторинга документов — их открытия, создания и закрытия — а также для гарантии того, что флажок Pause (Пауза) сохраняется в актуальном состоянии в ленте. Последняя функциональность обеспечивается отслеживанием активизации окон в событии WindowActivate.

Последняя задача, которая выполняется в обработчике событий — это запуск мониторинга текущего документа и добавление специальной панели задач в окно, содержащее документ:

```
  // Запуск мониторинга активного документа
  MonitorDocument(this.Application.ActiveDocument);
  AddTaskPaneToWindow(this.Application.ActiveDocument.ActiveWindow);
}
```

Вспомогательный метод `MonitorDocument()` добавляет таймер для документа:

```
internal void MonitorDocument(Word.Document Doc)
{
  // Мониторинг документа
  documentEditTimes.Add(Doc.Name, new DocumentTimer
  {
    Document = Doc,
    EditTime = new TimeSpan(0),
    IsActive = true,
    LastActive = DateTime.Now
  });
}
```

Этот метод просто создает новый `DocumentTimer` для документа. `DocumentTimer` ссылается на документ, имеет нулевое время редактирования, активен и был активизирован в текущее время. Затем он добавляет этот таймер в коллекцию `documentEditTimes` и ассоциирует его с именем документа.

Метод `AddTaskPaneToWindow()` добавляет в окно специальную панель задач. Этот метод начинается с проверки существования панели задач, чтобы удостовериться, что ее еще нет в окне. Еще одна странная особенность Word состоит в том, что если вы немедленно откроете старый документ после загрузки приложения, то новый документ исчезнет, не инициируя события закрытия. Это может привести к генерации исключения при обращении к окну панели задач, поэтому метод также проверяет `ArgumentNullException`, которое указывает на это.

```
private void AddTaskPaneToWindow(Word.Window Wn)
{
  // Проверка панели задач в окне
  Tools.CustomTaskPane docPane = null;
  Tools.CustomTaskPane paneToRemove = null;
  foreach (Tools.CustomTaskPane pane in timerDisplayPanes)
  {
    try
    {
      if (pane.Window == Wn)
      {
        docPane = pane;
        break;
      }
    }
    catch (ArgumentNullException)
    {
      // pane.Window равно null, поэтому document1 выгружен.
      paneToRemove = pane;
    }
  }
```

Если сгенерировано исключение, то сомнительная панель задач удаляется из коллекции:

```
  // Удалить панель при необходимости
  timerDisplayPanes.Remove(paneToRemove);
```

Если панель задач не найдена в окне, то метод завершается ее добавлением:

```
  // Добавить панель задач к документу
  if (docPane == null)
  {
    Tools.CustomTaskPane pane = this.CustomTaskPanes.Add(
      new TimerDisplayPane(documentEditTimes),
      "Document Edit Timer",
      Wn);
    timerDisplayPanes.Add(pane);
    pane.VisibleChanged +=
      new EventHandler(timerDisplayPane_VisibleChanged);
  }
}
```

Добавленная панель задач представляет собой экземпляр класса `TimerDisplayPane`. Чуть позже мы рассмотрим этот класс. Он добавляется с именем "Document Edit Timer". Также добавляется обработчик для события `VisibleChanged` объекта `CustomTaskPane`, который вы получаете после вызова метода `CustomTaskPanes.Add()`. Это позволяет обновить отображение в первый раз.

```
private void timerDisplayPane_VisibleChanged(object sender, EventArgs e)
{
  // Получить панель задач и переключить видимость
  Tools.CustomTaskPane taskPane = (Tools.CustomTaskPane)sender;
  if (taskPane.Visible)
  {
    TimerDisplayPane timerControl = (TimerDisplayPane)taskPane.Control;
    timerControl.RefreshDisplay();
  }
}
```

Класс `TimerDisplayPane` определяет метод `RefreshDisplay()`, который вызывается в предыдущем коде. Этот метод, как следует из его имени, обновляет отображение объекта `timerControl`.

Затем идет код, который обеспечивает мониторинг всех объектов. Сначала, когда создается новый документ, вызывается обработчик событий `eventInterface_NewDocument()`, и документ подвергается мониторингу вызовом методов `MonitorDocument()` и `AddTaskPaneToWindow()`, которые вы уже видели.

```
private void eventInterface_NewDocument(Word.Document Doc)
{
  // Мониторинг нового документа
  MonitorDocument(Doc);
  AddTaskPaneToWindow(Doc.ActiveWindow);
```

Этот метод также очищает флажок Pause в ленточном меню, поскольку новый документ запускается с работающим таймером. Это достигается служебным методом `SetPauseStatus()`, который определен в ленте:

```
  // Установить флажок
  Globals.Ribbons.TimerRibbon.SetPauseStatus(false);
}
```

Непосредственно перед закрытием документа вызывается обработчик события `eventInterface_DocumentBeforeClose()`. Этот метод "замораживает" таймер документа, обновляет общее время редактирования, очищает ссылку `Document` и удаляет панель задач из окна документа (с помощью метода `RemoveTaskPaneFromWindow()`, описанного ниже), прежде чем документ закрывается.

```
private void eventInterface_DocumentBeforeClose(Word.Document Doc,
   ref bool Cancel)
{
  // Заморозить таймер
  documentEditTimes[Doc.Name].EditTime += DateTime.Now
    - documentEditTimes[Doc.Name].LastActive;
  documentEditTimes[Doc.Name].IsActive = false;
  documentEditTimes[Doc.Name].Document = null;
  // Удалить панель задач
  RemoveTaskPaneFromWindow(Doc.ActiveWindow);
}
```

Когда документ открывается, вызывается метод `eventInterface_DocumentOpen()`. Здесь нужно выполнить еще небольшой кусок работы, так как перед мониторингом документа метод должен определить, существует ли уже таймер для документа, проверив это по его имени:

```
private void eventInterface_DocumentOpen(Word.Document Doc)
{
  if (documentEditTimes.ContainsKey(Doc.Name))
  {
    // Мониторинг старого документа
    documentEditTimes[Doc.Name].LastActive = DateTime.Now;
    documentEditTimes[Doc.Name].IsActive = true;
    documentEditTimes[Doc.Name].Document = Doc;
    AddTaskPaneToWindow(Doc.ActiveWindow);
  }
```

Если документ еще не подвергался мониторингу, то конфигурируется новый монитор, как для нового документа:

```
  else
  {
    // Мониторинг нового документа
    MonitorDocument(Doc);
    AddTaskPaneToWindow(Doc.ActiveWindow);
  }
}
```

Метод `RemoveTaskPaneFromWindow()` служит для удаления панели задач из окна. Код этого метода сначала проверяет существование панели задач в указанном окне:

```
private void RemoveTaskPaneFromWindow(Word.Window Wn)
{
  // Проверить наличие панели задач в окне
  Tools.CustomTaskPane docPane = null;
  foreach (Tools.CustomTaskPane pane in timerDisplayPanes)
  {
    if (pane.Window == Wn)
    {
      docPane = pane;
      break;
    }
  }
```

Если окно задач найдено, оно удаляется вызовом метода `CustomTaskPanes.Remove()`. Оно также удаляется из локальной коллекции ссылок в панели задач.

```
  // Удалить панель задач документа
  if (docPane != null)
  {
    this.CustomTaskPanes.Remove(docPane);
    timerDisplayPanes.Remove(docPane);
  }
}
```

Последний обработчик событий в этом классе, `eventInterface_WindowActivate()`, вызывается, когда активизируется окно. Этот метод получает таймер активного документа (сначала проверяя, существует ли таковой, поскольку на момент вызова этого метода новый документ может быть еще не добавлен) и устанавливает флажок в ленточном меню, так чтобы он был обновлен для документа:

```
private void eventInterface_WindowActivate(Word.Document Doc,
   Word.Window Wn)
{
   if (documentEditTimes.ContainsKey(this.Application.ActiveDocument.Name))
   {
     // Проверить правильность установки флажка в ленте, начать с получения таймера
     DocumentTimer documentTimer =
        documentEditTimes[this.Application.ActiveDocument.Name];

     // Установить флажок
     Globals.Ribbons.TimerRibbon.SetPauseStatus(!documentTimer.IsActive);
   }
}
```

Код `ThisAddIn` также содержит два вспомогательных метода. Первый из них, `ToggleTaskPaneDisplay()`, используется для отображения или сокрытия панели задач для текущего активного документа, посредством установки свойства `CustomTaskPane.Visible`.

```
internal void ToggleTaskPaneDisplay()
{
  // Проверить наличие панели задач в окне
  AddTaskPaneToWindow(this.Application.ActiveDocument.ActiveWindow);

  // Переключить панель задач документа
  Tools.CustomTaskPane docPane = null;
  foreach (Tools.CustomTaskPane pane in timerDisplayPanes)
  {
    if (pane.Window == this.Application.ActiveDocument.ActiveWindow)
    {
      docPane = pane;
      break;
    }
  }
  docPane.Visible = !docPane.Visible;
}
```

Метод `ToggleTaskPaneDisplay()`, показанный в этом коде, как вы вскоре увидите, вызывается обработчиками событий элемента управления лентой.

И, наконец, класс имеет другой метод, вызываемый из ленточного меню, который позволяет его элементам управления приостанавливать или возобновлять работу таймера документа:

```
internal void PauseOrResumeTimer(bool pause)
{
  // Получить таймер
  DocumentTimer documentTimer =
  documentEditTimes[this.Application.ActiveDocument.Name];

  if (pause && documentTimer.IsActive)
  {
    // Заморозить таймер
    documentTimer.EditTime += DateTime.Now - documentTimer.LastActive;
    documentTimer.IsActive = false;
  }
```

```
    else if (!pause && !documentTimer.IsActive)
    {
      // Возобновить таймер
      documentTimer.IsActive = true;
      documentTimer.LastActive = DateTime.Now;
    }
  }
}
```

Единственный оставшийся код в определении этого класса — это пустой обработчик события `Shutdown`, а также сгенерированный VSTO код для подключения обработчиков событий `Startup` и `Shutdown`.

Далее лента проекта `TimerRibbon` компонуется, как показано на рис. 49.13.

***Рис. 49.13**. Компоновка ленты `TimerRibbon`*

Лента содержит элементы управления `RibbonButton`, `RibbonSeparator`, `RibbonCheckBox` и `DialogBoxLauncher`. Кнопка использует крупный стиль отображения и имеет `OfficeImageId`, равный `StartAfterPrevious`, что отображает поверхность кнопки, как показано на рис. 40.13. (Эти изображения невидимы во время проектирования.) Лента использует тип вкладки `TabHome`, что приводит к добавлению ее содержимого на вкладку Home.

Лента имеет три обработчика событий, каждый из которых вызывает по одному из служебных методов `ThisAddIn`, описанных ранее:

```
private void group1_DialogLauncherClick(object sender,
  RibbonControlEventArgs e)
{
  // Показать или скрыть панель задач
  Globals.ThisAddIn.ToggleTaskPaneDisplay();
}

private void pauseCheckBox_Click(object sender, RibbonControlEventArgs e)
{
  // Приостановить таймер
  Globals.ThisAddIn.PauseOrResumeTimer(pauseCheckBox.Checked);
}

private void toggleDisplayButton_Click(object sender, RibbonControlEventArgs e)
{
  // Показать или скрыть панель задач
  Globals.ThisAddIn.ToggleTaskPaneDisplay();
}
```

*Фрагмент кода **`TimerRibbon.cs`***

Лента также включает собственный служебный метод `SetPauseStatus()`, который вызывается кодом `ThisAddIn` для установки или снятия метки флажка:

```
internal void SetPauseStatus(bool isPaused)
{
  // Привести флажок в соответствие
  pauseCheckBox.Checked = isPaused;
}
```

Другой компонент этого решения — пользовательский элемент управления `TimerDisplayPane`, используемый в панели задач. Компоновка этого элемента показана на рис. 49.14.

***Рис. 49.14**. Компоновка пользовательского элемента управления `TimerDisplayPane`*

Элемент включает кнопку, метку и окно списка — не слишком впечатляющий набор, но его достаточно просто заменить, например, более симпатичным элементом WPF.

Код для этого элемента сохраняет локальную ссылку на таймеры документа, устанавливаемые в конструкторе:

```
public partial class TimerDisplayPane : UserControl
{
  private Dictionary<string, DocumentTimer> documentEditTimes;
  public TimerDisplayPane()
  {
    InitializeComponent();
  }
  public TimerDisplayPane(Dictionary<string, DocumentTimer>
    documentEditTimes) : this()
  {
    // Сохранить ссылку на время редактирования
    this.documentEditTimes = documentEditTimes;
  }
```

*Фрагмент кода **`TimerDisplayPane.cs`***

Обработчик события кнопки вызывает метод `RefreshDisplay()`, чтобы обновить отображение таймера:

```
private void refreshButton_Click(object sender, EventArgs e)
{
  RefreshDisplay();
}
```

Метод `RefreshDisplay()` также вызывается из `ThisAddIn`, как было показано ранее. Это неожиданно сложный метод, если учесть то, что он делает. Он также сверяет список наблюдаемых документов со списком загруженных документов и исправляет обнаруженные проблемы. Такого рода код часто необходим в приложениях VSTO, поскольку интерфейс с объектной моделью COM Office иногда работает не так хорошо, как должен. Эмпирическое правило, которого здесь следует придерживаться — применяйте приемы защитного программирования.

Метод начинается с очистки текущего списка таймеров в окне списка `timerList`:

```
internal void RefreshDisplay()
{
   // Очистить существующий список
   this.timerList.Items.Clear();
```

Затем проверяются мониторы. Метод проходит по всем документам коллекции `Globals.ThisAddIn.Application.Documents` и определяет, подвергается ли документ мониторингу, не подвергается или же подвергается, но его имя изменилось с момент последнего обновления.

Поиск подвергаемых мониторингу документов включает простую проверку имени документа по именам документов в коллекции ключей documentEditTimes:

```
// Проверка мониторинга всех документов
foreach (Word.Document doc in Globals.ThisAddIn.Application.Documents)
{
  bool isMonitored = false;
  bool requiresNameChange = false;
  DocumentTimer oldNameTimer = null;
  string oldName = null;
  foreach (string documentName in documentEditTimes.Keys)
  {
    if (doc.Name == documentName)
    {
      isMonitored = true;
      break;
    }
```

Если имя не совпадает, сверяются ссылки на документы, что позволяет обнаружить документы с измененными именами, как показано в следующем коде:

```
    else
    {
      if (documentEditTimes[documentName].Document == doc)
      {
        // Подвергается мониторингу, но имя изменилось!
        oldName = documentName;
        oldNameTimer = documentEditTimes[documentName];
        isMonitored = true;
        requiresNameChange = true;
        break;
      }
    }
  }
```

Для документов, не подверженных мониторингу, создается новый монитор:

```
  // Добавить новый монитор, если нужно
  if (!isMonitored)
  {
    Globals.ThisAddIn.MonitorDocument(doc);
  }
```

Документы с измененными именами заново ассоциируются с мониторами, использованными для документов со старыми именами:

```
  // Переименовать при необходимости
  if (requiresNameChange)
  {
    documentEditTimes.Remove(oldName);
    documentEditTimes.Add(doc.Name, oldNameTimer);
  }
}
```

После согласования таймеров редактирования документов генерируется список. Этот код также проверяет, загружены ли все еще документы, на которые установлены ссылки, и приостанавливает таймер для документов, которые не загружены, устанавливая свойство IsActive в false. Опять-таки, это — защитное программирование.

```
// Создать новый список
foreach (string documentName in documentEditTimes.Keys)
{
  // Проверить, загружен ли все еще документ
```

```
        bool isLoaded = false;
        foreach (Word.Document doc in
          Globals.ThisAddIn.Application.Documents)
        {
          if (doc.Name == documentName)
          {
            isLoaded = true;
            break;
          }
        }
        if (!isLoaded)
        {
          documentEditTimes[documentName].IsActive = false;
          documentEditTimes[documentName].Document = null;
        }
```

Для каждого монитора добавляется элемент в окно списка, включающий имя документа и его общее время редактирования:

```
        // Добавить элемент
        this.timerList.Items.Add(string.Format("{0}: {1}", documentName,
          documentEditTimes[documentName].EditTime +
          (documentEditTimes[documentName].IsActive ?
          (DateTime.Now - documentEditTimes[documentName].LastActive) :
           new TimeSpan(0))));
      }
    }
  }
```

На этом код данного примера завершен. Было продемонстрировано использование элементов управления типа ленты и панели задач, а также поддержка панели задач в нескольких документах Word. Также были проиллюстрированы многие приемы, описанные ранее в этой главе.

Резюме

В этой главе было показано, как использовать VSTO для создания управляемых решений для продуктов Office.

В первой части этой главы описывалась общая структура проектов VSTO и типов этих проектов, которые можно создавать. Также вы видели средства, которые доступны для облегчения программирования VSTO.

В главе подробно рассматривались некоторые средства, доступные в решениях VSTO, и было показано, как взаимодействовать с объектной моделью Office. Также вы узнали о пространствах имен и типах, доступных в VSTO, и научились использовать эти типы для реализации разнообразной функциональности. Затем было проведено исследование некоторых характеристик кода проектов VSTO и их применение для получения нужного эффекта.

После этого мы обратились к более практическим вещам. Вы узнали, как управляются дополнения в приложениях Office, и как осуществляется взаимодействие с объектной моделью Office. Также вы видели, как настроить пользовательский интерфейс приложений, оснастив его ленточными меню, панелями задач и панелями действий.

И, наконец, был рассмотрен пример приложения, иллюстрирующего приемы построения пользовательского интерфейса и реализации взаимодействия, изученные ранее. В примере были продемонстрированы полезные приемы, включая управление панелями задач во множестве окон документов Word.