Алексей Федоров

Microsoft® Visual Studio® 2008

Краткий обзор ключевых новинок

Алексей Федоров

Microsoft® Visual Studio® 2008

Краткий обзор ключевых новинок

Содержание

Введение

В этом году Visual Studio отмечает свое десятилетие. Планируемая к выпуску версия Microsoft Visual Studio 2008 включает в себя лучшее из появившегося в продукте за последнюю декаду. Первая версия Visual Studio была выпущена в 1997 году и состояла из отдельных средств разработки на языках Visual C++, Visual Basic и Visual J++, а также включала продукт под названием Visual InterDev, предназначенный для создания веб-приложений с использованием JavaScript.

Версия Visual Studio 6.0, в которой появился самый популярный до последнего времени язык разработки Visual Basic 6.0, стала платформой, поддерживавшей унифицированный набор сервисов для различных языков программирования и заложила основу новой архитектуры средств разработки компании Microsoft.

С появлением Visual Studio .NET 2002 и Visual Studio .NET 2003 разработчикам стала доступна технология на базе CRL и .NET Framework. Впервые за всю историю существования средств разработки программисты смогли использовать единый набор средств, включающий визуальные средства дизайна, визуальные компоненты с поддержкой drag-and-drop и технологию IntelliSense. В то же время стали появляться и средства, позволявшие упрощать коллективную работу на программными проектами.

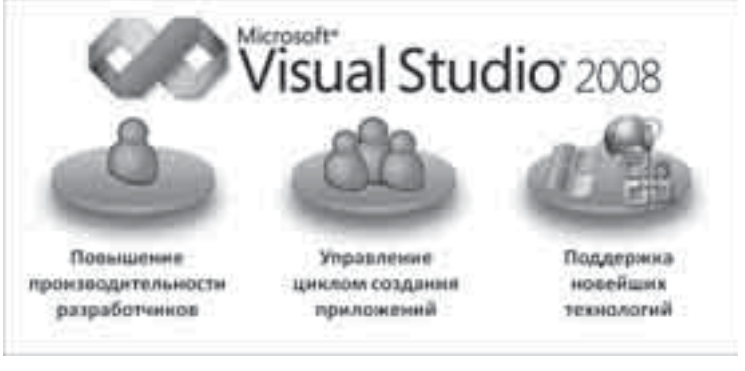
В Visual Studio 2005 средства, облегчающие коллективную, командную работу в рамках основных проектных ролей (архитектор, разработчик, тестировщик) увидели свет в виде нескольких специальных ролевых изданий, объединенных названием Visual Studio Team System и компонента, обеспечивающего фундамент для совместной работы — Visual Studio 2005 Team Foundation Server.

Visual Studio 2008 продолжает традиции, заложенные предыдущими версиями продукта и содержит большое число новинок, предназначенных как для упрощения создания приложений для различных платформ отдель-

ными разработчиками, так и для повышения производительности всей проектной команды.

Если кратко охарактеризовать новинки, появившиеся в Microsoft Visual Studio 2008, то их можно будет разделить на три категории — повышение производительности разработчиков, поддержка новейших технологий и управление всем циклом создания приложений.

К новинкам в области повышения производительности разработчиков можно отнести расширения в языках программирования, поддержку написания кода для нескольких версий .NET Framework (multi-targeting), улучшенные средства интеграции между данными и языками программирования в Visual Basic .NET и C# (технология Language Integrated Query, LINQ), а также существенные улучшения в поддержке создания веб-приложений, включающие расширения в дизайнере HTML/CSS и поддержку технологии Intellisense для JavaScript и расширенные средства отладки.



Группы ключевых новинок в Visual Studio 2008

В области поддержки новейших технологий мы найдем следующие новинки — поддержку создания приложений для Windows Server 2008, Windows Vista (включая Common Dialogs) и Microsoft Office 2007 (включая новую версию Visual Studio Tools for Office и развертывание через ClickOnce), а также средства для работы с SQL Server 2008. В частности, в Visual Studio 2008 полностью поддерживается создание приложений для платформы .NET Framework 3.0 и 3.5, а также обеспечивается интегрированная поддержка таких технологий, как ASP.NET AJAX и Microsoft Silverlight.

Средства управления циклом создания приложений (Visual Studio Team System) обогатились расширенным набором поддерживаемых ролей, включая издание для работы с базами данных — Team Edition for Database Professionals. Помимо этого расширена функциональность модульного тестирования, реализована поддержка нагрузочного тестирования для корпоративных сценариев и реализованы новые средства измерения производительности и диагностики.

В основной массе (а в Visual Studio 2008 можно насчитать более 200 новых и расширенных функций), эти новинки применимы к разработке веб-приложений, разработке Windows-приложений, а также созданию решений на платформе Microsoft Office, хотя не забыты и приложения для мобильных устройств, а также поддержка создания сервисов. Ниже мы поговорим об этом более подробно.

Варианты поставки Visual Studio 2008

Варианты поставки Visual Studio 2008 практически не отличаются от вариантов поставки Visual Studio 2005 (за исключением, наверное, издания для работы с базами данных — Team Edition for Database Professionals) и закрывают потребности всего сообщества разработчиков — от студентов и энтузиастов (издания семейства Express) до профессиональных разработчиков, входящих в состав проектной группы (Visual Studio Team System). Некоторые расширения, которые мы обсудим в данном обзоре — поддержка технологии LINQ, расширения в дизайнере HTML/CSS, дизайнер для Windows Presentation Foundation и O-R дизайнер, доступны в изданиях семейства Express, в то время как другие, в первую очередь Visual Studio Tools for Office, требуют использования Visual Studio Professional Edition.

Издания Visual Studio 2008

По сравнению с Visual Studio 2005, ряд изданий продуктов семейства Visual Studio получил новые названия, которые показаны в следующей таблице.

Название в Visual Studio 2005	Название в Visual Studio 2008
Visual Studio Team System	Visual Studio Team System 2008
Visual Studio 2005 Team Suite	Visual Studio Team System 2008 Team Suite
Visual Studio 2005 Team Edition for Software Architects	Visual Studio Team System 2008 Architecture Edition
Visual Studio 2005 Team Edition for Software Developers	Visual Studio Team System 2008 Development Edition
Visual Studio 2005 Team Edition for Software Testers	Visual Studio Team System 2008 Test Edition
Visual Studio 2005 Team Edition for Database Professionals	Visual Studio Team System 2008 Database Edition
Visual Studio 2005 Team Foundation Server	Visual Studio Team System 2008 Team Foundation Server
Visual Studio 2005 Team Test Load Agent	Visual Studio Team System 2008 Test Load Agent

Названия остальных изданий продуктов, входящих в семейство Visual Studio, остались прежними — изменилась лишь версия продукта.

Visual Studio 2005	Visual Studio 2008
Visual C# 2005 Express Edition	Visual C# 2008 Express Edition
Visual C++ 2005 Express Edition	Visual C++ 2008 Express Edition
Visual Basic 2005 Express Edition	Visual Basic 2008 Express Edition
Visual Web Developer 2005 Express Edition	Visual Web Developer 2008 Express Edition
Visual Studio 2005 Standard Edition	Visual Studio 2008 Standard Edition
Visual Studio 2005 Professional Edition	Visual Studio 2008 Professional Edition

С помощью средств, включенных в состав Visual Studio 2008, можно создавать приложения для широкого спектра платформ — от т. н. «разумных» устройств (smart personal objects) до сотовых телефонов, планшетных компьютеров и настольных компьютеров и серверов.

Платформы, поддерживаемые в Visual Studio 2008

Требования к системе

В данном разделе мы рассмотрим основные требования к системе для Visual Studio 2008 и Visual Studio Team System 2008. Visual Studio 2008 может быть установлена на следующие операционные системы:

- Windows Vista (x86 и x64) — все издания за исключением издания Starter Edition;
- Windows XP (x86 и x64) с установленным пакетом обновлений Service Pack 2 или более поздней версии — все издания за исключением издания Starter Edition;
- Windows Server 2003 (x86 и x64) с установленным пакетом обновлений Service Pack 1 или более поздней версии;
- Windows Server 2003 R2 (x86 или x64) или более поздней версии;
- Windows Server 2008 (x86 и x64).

Для установки Visual Studio 2008 потребуется компьютер со следующими характеристиками:

- Процессор с частотой не ниже 1.6 ГГц;
- Минимум 384 Мбайт оперативной памяти (768 Мбайт или более для работы под Windows Vista);
- 2.2 Гбайт свободного пространства на жестком диске со скоростью не менее 5400 об/мин;
- Дисплей с разрешением не менее 1024 x 768;
- Привод DVD.

Издание Microsoft Visual Studio 2008 Team System Team Foundation Server представляет собой многозвенное приложение, компоненты которого могут быть установлены на различных компьютерах. Для установки Visual Studio Team Foundation Server на одном компьютере, вам потребуется машина со следующими характеристиками:

- Процессор с частотой не ниже 2.2 ГГц;
- Минимум 2 Гбайт оперативной памяти;
- 8 Гбайт свободного пространства на жестком диске со скоростью не менее 5400 об/мин;
- Дисплей с разрешением не менее 1024 x 768;
- Привод DVD.

Пакет Visual Studio Team Foundation Server может быть установлен на следующие операционные системы:

- Windows Vista (x86 и x64) — все издания за исключением издания Starter Edition;
- Windows XP (x86 и x64) с установленным пакетом обновлений Service Pack 2 или более поздней версии — все издания за исключением издания Starter Edition;
- Windows Server 2003 (x86 и x64) с установленным пакетом обновлений Service Pack 1 или более поздней версии;
- Windows Server 2003 R2 (x86 или x64) или более поздней версии;
- Windows Server 2008 (x86 и x64).

После того как мы познакомились с общими новинками в Visual Studio 2008, давайте перейдем к деталям. Начнем наше обсуждение с новинок в языках программирования — C# 3.0 и Visual Basic .NET 9.0.

Глава 1

Новинки в языках программирования

Наше знакомство с ключевыми новинками в Microsoft Visual Studio 2008 начнем с обсуждения языков программирования, а именно новых версий C# — C# 3.0 и Visual Basic .NET — VB 9.0, которые пополнились рядом синтаксических расширений и поддержкой технологии LINQ.

При работе над новой версией языка C# его создатели ставили перед собой следующие задачи — обеспечить на уровне языка интеграцию механизмов работы с объектами, реляционными данными и XML, а также добавить в язык ряд новых программных конструкций и при этом сделать его полностью совместимым с предыдущими версиями. Команда, отвечающая за развитие языка Visual Basic, решала схожие задачи — упростить механизмы запросов к данным, включить поддержку на уровне языка таких операций, как запрос к данным и их преобразование, унифицировать механизмы запросов к объектам, реляционным данным и данным в формате XML. Помимо этого в новую версию Visual Basic были встроены расширенные механизмы работы с XML, включая поддержку документов без схемы, возможность построения XML-документов и упрощенный доступ к элементам таких документов.

Таким образом, интеграция и унификация механизмов запросов к объектам, реляционным данным и данным в формате XML вылилась в технологию LINQ. Об этом механизме мы поговорим чуть позже, а сейчас обратим внимание на синтаксические расширения, появившиеся в новых версиях языков C# и Visual Basic .NET.

C# 3.0

Начнем с C# 3.0. Расширения, появившиеся в этой версии языка C# показаны в следующей таблице.

Расширение	Описание
Автоматическое создание методов для свойств объектов	Используется для автоматизации процесса создания свойств со стандартными функциями
Типизация локальных переменных	Позволяет определить тип локальной переменной по выражению, которое используется для ее инициализации
Типизация массивов	Форма создания и инициализации массива, в которой тип элемента определяется по выражению, используемому для его инициализации
Методы-расширения	Способ расширения существующих типов, а также типов, создаваемых пользователями

Расширение	Описание
Лямбда-выражения	Расширение механизмов анонимных методов, которое существенно упрощает автоматическое определение типов и преобразование как в делегаты, так и в деревья выражений
Деревья выражений	Механизм поддержки лямбда-выражений, позволяющий представлять их в виде данных (деревьев выражений), а не кода (делегатов)
Инициализаторы объектов и коллекций	Позволяют задавать значения одного или более полей или свойств вновь созданных объектов и объединять в одном шаге создание и инициализацию
Запросы	Интегрированный в язык синтаксис запросов, схожих с языком запросов к реляционным и иерархическим данным типа SQL и XQuery
Анонимные типы	Типы, которые автоматически создаются на основе инициализаторов объектов

Рассмотрим эти более подробно. Начнем с автоматического создания свойств.

Автоматическое создание методов для свойств объектов

Часто методы доступа к свойствами объектов — get и set — имеют очень простую реализацию и служат для получения значения соответствующего поля внутри объекта (get) или присвоения этому полю нового значения (set). Создание свойств со стандартными методами доступа может существенно упростить написание кода и сделать его более понятным, т. к. в этом случае компилятор C# автоматически генерирует весь необходимый код. Рассмотрим следующий пример. Пусть у нас есть класс Point, содержащий два свойства:

```
public class Point
{
    private int _x;
    private int _y;

    public int X { get { return _x; } set { _x = value; } }
    public int Y { get { return _y; } set { _y = value; } }
}
```

Для упрощения описания этого класса мы можем воспользоваться автоматическим созданием свойств и в этом случае описание класса Point будет выглядеть так:

```
public class Point

{
    public int X {get ; set ;}
    public int Y {get ; set ;}
}
```

Так как теперь компилятор берет на себя всю работу по реализации методов get и set, мы можем инициализировать наш объект как обычно:

```
GroundZero p = new Point();
p.X = 0;
p.Y = 0;
```

Если мы переопределим метод ToString() для нашего объекта следующим образом:

```
public override string ToString()
{
    return X + ";" + Y;
}
```

то сможем воспользоваться методом WriteLine() для вывода содержимого полей объекта на экран консоли:

```
Console.WriteLine(c);
```

Создание свойств описанным выше методом требует, чтобы у свойств были методы доступа get и set, но если мы захотим добавить собственные методы, мы можем воспользоваться подходом, описанным ниже.

Предположим, что у нас есть класс Customer, описанный следующим образом:

```
public class Customer
{

    public string Name { get; set; }
    public string City { get; set; }

    public override string ToString()
    {
        return Name + "\t" + City;
   }
}
```

Теперь предположим, что мы хотим добавить еще одно свойство, которое вне класса будет доступно только для чтения. Метод автоматического создания свойств позволяет решить эту задачу путем использования модификатора — в нашем примере это будет модификатор private для метода set:

```
public class Customer
{
    public int CustomerID { get; private set; }
    public string Name { get; set; }
         public string City { get; set; }
}
```

Инициализация объектов и коллекций

Следующее расширение синтаксиса языка C#, которое мы рассмотрим, связано с упрощением инициализации объектов и коллекций. При объявлении объекта или коллекции в C# имеется возможность использования т. н. инициализатора, который будет задавать начальные значения членов вновь создаваемых объектов или коллекций. Новый синтаксис позволяет объединить в одном шаге и создание и инициализацию объекта или коллекции. Рассмотрим пример использования инициализатора для объекта. Пусть у нас есть класс Point, содержащий два свойства:

```
public class Point
{
    private int _x;
    private int _y;

    public int X { get { return _x; } set { _x = value; } }
    public int Y { get { return _y; } set { _y = value; } }
}
```

Инициализатор объекта состоит из последовательности инициализаторов членов класса, заключенных в фигурные скобки { } и разделенные запятыми. Каждый инициализатор члена класса присваивает значение полю или свойству объекта. Для нашего класса Point инициализация с использованием нового синтаксиса будет выглядеть так:

```
Point p = new Point { X = 3, Y = 99 };
```

Такой компактный синтаксис инициализации объекта семантически эквивалентен вызову конструктора класса и присвоению значения каждому члену класса.

Отметим, что при таком способе инициализации не обязательно указывать в списке все поля объекта — не указанные поля получат значения, присваиваемые по умолчанию.

В языке C# объекты, которые реализуют интерфейс System.Collections.Generic.IEnumerable<T> и имеют метод Add(), могут быть инициализированы используя инициализатор коллекций. Используя наш класс Point, мы можем создать коллекцию точек, описывающую какую-то геометрическую фигуру:

```
List<Point> Square = new List<Point>
{
    new Point { X=0, Y=5 },
    new Point { X=5, Y=5 },
    new Point { X=5, Y=0 },
    new Point { X=0, Y=0 }
};
```

Типизация переменных и массивов

Следующее расширение синтаксиса языка C#, которое мы рассмотрим, связано с типизацией переменных и массивов и позволяет определить тип локальной переменной или элемента массива по выражению, которое используется для инициализации. Для задания переменных, тип которых может быть определен компилятором автоматически, используется конструкция var, а для использования аналогичных возможностей для массивов — синтаксис new[]{...} — обратите внимание на отсутствие указания типа. Сначала приведем «стандартный» синтаксис, который мы использовали для задания и инициализации переменных и массивов в предыдущих версиях языка:

```
int i = 43;
string s = "...This is only a test...";
int[] numbers = new int[] { 4, 9, 16};
```

Используя механизмы типизации для переменных и массивов, мы можем инициализировать переменные и массивы более простым способом:

```
var i = 43;
var s = "...This is only a test...";
var numbers = new [] { 4, 9, 16 };
```

Механизм типизации для переменных и массивов имеет ряд ограничений. Например, так как тип переменной определяется по способу ее инициализации, при объявлении переменной с помощью конструкции var всегда должен присутствовать инициализатор. Например, при выполнении следующего кода мы получим ошибку:

```
var x;  // Ошибка: тип не определен
x = new int[] { 1, 2, 3 };
```

Также мы всегда должны использовать синтаксис new[]{...} при создании инициализируемых массивов — при выполнении следующего кода мы получим ошибку:

```
var x = {1, 2, 3};
```

для исправления которой код нужно переписать следующим образом:

```
var x = new [] {1, 2, 3};
```

Используя конструкцию var мы можем упростить код некоторых комплексных выражений. Например, вместо использования в механизме итерации foreach четкого указания типа, как показано в следующем примере,

```
Console.WriteLine("Customers");

foreach (Customer c in customers)
 Console.WriteLine(c);
```

мы можем использовать конструкцию var:

```
Console.WriteLine("Customers:");

foreach (var c in customers)
 Console.WriteLine(c);
```

Методы-расширения

Методы-расширения представляют собой способ расширения существующих типов, а также типов, создаваемых пользователями. Используя этот способ разработчики могут добавлять к существующим типам новые методы, которые будут вызываться с использованием стандартного синтаксиса. Методы-расширения — это статические методы, объявляемые с использованием ключевого слова this в качестве модификатора первого параметра метода. Приведем следующий пример. Предположим, что нам нужна функция сравнения двух объектов типа Customer. В C# 2.0 мы могли бы написать следующий код:

```
public static class Extensions
{
  public static bool Compare(Customer customer1, Customer
   customer2)
    {
        if (customer1.Name == customer2.Name &&
            customer1.City == customer2.City )
```

```
        {
            return true;
        }

        return false;
    }
}
```

Вызов этого метода может выглядеть так:

```
foreach (var c in customers)
    {
        if (Extensions.Compare(newCusomter, c))
        {
            Console.WriteLine("Already in the list");

            return;
        }
    }
```

В C# 3.0 можно реализовать ту же функциональность с помощью метода-расширения, который будет в контексте объекта вызываться с использованием стандартного синтаксиса. Добавим ключевое слово this в качестве модификатора первого параметра метода Compare();

```
public static class Extensions
{
 public static bool Compare(this Customer customer1, Customer
  customer2)
 {
   ...
 }
}
```

Тогда наш код проверки двух объектов будет выглядеть следующим образом:

```
foreach (var c in customers)
{

    if (newOrder.Compare(c))
    {
        Console.WriteLine("Already in the list");
        return;
    }
...
```

Запомним следующее простое правило: методы-расширения доступны только в том случае, когда они объявлены в статическом классе и находятся в области видимости соответствующего пространства имен. Эти методы будут доступны в качестве дополнительных методов для типов, указанных в качестве первого параметра метода.

Методы-расширения могут быть добавлены к любому типу, включая такие встроенные типы, как List<T> и Dictionary<K, V>. Рассмотрим пример расширения функциональности стандартного типа List<T> — добавим к нему метод Append(), который будет объединять два элемента типа List<T> в один:

```
public static class Extensions
{
    public static List<T> Append<T>(this List<T> a, List<T> b)
    {
        var newList = new List<T>(a);
        newList.AddRange(b);
        return newList;
    }
...
}
```

Вызов нового метода-расширения для стандартного типа может выглядеть так:

```
{
...
    var addedCustomers = new List<Customer>
    {
     new Customer { Name = "Paolo Accorti", City = "Torino" },
     new Customer { Name = "Diego Roel",    City = "Madrid" }
    };

    customers.Append(addedCustomers);
...
}
```

Методы-расширения предоставляют в распоряжение разработчиков элегантный способ расширения функциональности типов таким образом, что добавленные функции становятся частью типа. Используя методы-расширения можно добавлять новую функциональность к уже откомпилированным классам, включая классы, созданные пользователем и стандартные классы .NET Framework.

Лямбда-выражения

Лямбда-выражения — это расширение механизмов анонимных методов, которое существенно упрощает автоматическое определение типов и преобразование как в делегаты, так и в деревья выражений.

В C# 2.0 появились анонимные методы, позволявшие вставлять блоки кода в тех местах, где было возможно использование делегатов. Например:

```
var innerPoints = points.FindAll(delegate(Point p)
 { return (p.X > 0 && p.Y > 0); });
```

Метод FindAll() ожидает параметр в виде делегата. В нашем примере делегат определяет, являются ли координаты x и y положительными, т. е. относится ли точка с заданными координатами к первому квадранту картезианской поверхности.

В C# 3.0 появились лямбда-выражения, которые позволяют использовать более простой синтаксис для написания анонимных методов. Таким образом, предыдущий пример можно переписать так:

```
var innerPoints = points.FindAll( p => p.X > 0 && p.Y > 0);
```

Лямбда-выражение пишется как список параметров, за которым следует символ =>, а за ним – код самого выражения. Например:

```
(int x) => { return x + 1; }
```

Параметры лямбда-выражения могут быть непосредственно или опосредованно типизованы:

```
(int x) => x + 1
```

В списке опосредованно типизованных параметров типы параметров получаются из контекста, в котором используется данное лямбда-выражение. Помимо этого, если в лямбда-выражении задан один типизованный параметр, список параметров можно не заключать в скобки:

```
x => x + 1

(x,y) => x * y;
```

Предположим, что мы хотим найти в списке объектов Customer все объекты с определенным значением поля City. Для этого мы можем использовать метод FindAll() класса List<T>. Напишем метод FindCustomersByCity(), в котором будем использовать анонимные методы и делегаты используя синтаксис C# 2.0:

```
public static List<Customer> FindCustomersByCity(
    List<Customer> customers,
    string city)
```

```
{
    return customers.FindAll(
        delegate(Customer c){
            return c.City == city;
    });
}
```

Вызов этого метода будет выглядеть следующим образом:

```
{
    var customers = CreateCustomers();

    foreach (var c in FindCustomersByCity(customers, "London"))
        Console.WriteLine(c);
}
```

Теперь воспользуемся возможностями C# 3.0 и заменим анонимный метод эквивалентным лямбда-выражением:

```
public static List<Customer> FindCustomersByCity(
    List<Customer> customers, string city)
{
    return customers.FindAll(c => c.City == city);
}
```

Как мы увидели выше, использование лямбда-выражений в некоторых случаях упрощает написание кода. В приведенном выше примере список параметров состоит из нетипизованного параметра 'c' и, таким образом, явного указания типа не требуется. В качестве альтернативы мы можем написать следующий код:

```
return customers.FindAll ((Customer c) => c.City == city);
```

На этом мы закончим наше знакомство с некоторыми новыми возможностями, появившимися в C# 3.0. Желающие более подробно познакомиться со всеми новинками, могут обратиться к спецификации C# 3.0, которая доступна на сайте MSDN, а также к соответствующим разделам документации по .NET Framework 3.0 SDK и Visual Studio 2008. Теперь посмотрим, что нового появилось в Visual Basic .NET 9.0.

Visual Basic .NET 9.0

Как мы отметили во введении к этой главе, задача новой версии VB.NET — упростить механизмы запросов к данным, включить поддержку на уровне языка таких операций, как запрос к данным и их преобразование, унифицировать механизмы запросов к объектам, реляционным данным и дан-

ным в формате XML. Помимо этого в новую версию Visual Basic встроены расширенные механизмы работы с XML, включая поддержку документов без схемы, возможность построения XML-документов и упрощенный доступ к элементам таких документов.

В качестве примера будем использовать класс Country, который будет содержать информацию о стране — название, площадь и численность населения:

```
Class Country
  Public Property Name As String
  Public Property Area As Long
  Public Property Population As Integer
End Class
```

Типизация локальных переменных

Начнем наше знакомство с новинками в Visual Basic 9.0 типизации локальных переменных. Как мы помним из нашего обсуждения языка C# 3.0, этот механизм позволяет определить тип локальной переменной по выражению, которое используется для ее инициализации. Например, компилятор может легко определить типы для следующих локальных переменных:

```
Dim Name = "Monaco"
Dim Area = 1.9
Dim Population = 31719

Dim country = New Country With { .Name = "Monaco", …}
```

Приведенный выше код является эквивалентом следующей группы объявлений:

```
Dim Name As String = "Monaco"
Dim Area As Float = 1.9
Dim Population As Integer = 31719

Dim country As Country = New Country With { .Name = "Monaco", …}
```

Так как типы локальных переменных определяются компилятором по выражению, которое используется для их инициализации — это возможно благодаря новой опции Option Infer On, которая включена по умолчанию для всех новых проектов; в независимости от значения опции Option Strict доступ к таким переменным всегда осуществляется на основе раннего связывания. Для включения позднего связывания всегда нужно указывать тип переменных как Object, например:

```
Dim country As Object = New Country With { .Name = "Monaco", … }
```

Определение типов данных при их инициализации предотвращает использование позднего связывания и позволяет реализовывать такие расширения, как связывание с новыми типами данных, например, XML – мы обсудим эти возможности ниже.

Инициализация массивов и объектов

В Visual Basic ключевое слово With упрощает доступ к нескольким членам агрегатного типа данных без необходимости в указании этого типа для каждого члена. Внутри блока With выражение, используемое для доступа к члену типа, начинается с точки и при его обработке компилятором такое выражение «разворачивается» в полное, как если бы слева от точки было указано название типа. Например:

```
Dim palau As New Country()
With palau
  .Name = “Monaco”
  .Area = 1.9
  .Population = 31719
End With
```

Новые в Visual Basic 9.0 инициализаторы объектов позволяют существенно упростить создание и инициализацию объектов, объединив их в один шаг:

```
Dim palau New Country With {
  .Name = “Monaco”, _
  .Area = 1.9, _
  .Population = 31719 _
}
```

Как мы увидели выше, инициализаторы объектов достаточно удобны для создания коллекций комплексных объектов. Массивы могут быть инициализированы и типы их элементов автоматически определены с помощью инициализаторов массивов. Например, если есть класс City, описывающий город:

```
Class City
  Public Property Name As String
  Public Property Country As String
  Public Property Longitude As Long
  Public Property Latitude As Long
End Class
```

мы можем создать массив, описывающий столицы некоторых стран, следующим образом:

```
Dim Capitals = { _
  New City With { _
    .Name = "Antanarivo", _
    .Country = "Madagascar", _
    .Longitude = 47.4, _
    .Latitude = -18.6 }, _
  New City With { _
    .Name = "Belmopan", _
    .Country = "Belize", _
    .Longitude = -88.5, _
    .Latitude = 17.1 }, _
  New City With { _
    .Name = "Monaco", _
    .Country = "Monaco", _
    .Longitude = 7.2, _
    .Latitude = 43.7 }, _
  New City With { _
    .Country = "Palau", _
    .Name = "Koror", _
    .Longitude = 135, _
    .Latitude = 8 } _
}
```

«Пустые» типы

В новой версии Visual Basic появилась поддержка пустых типов, реализованная через стандартный тип данных Nullable(Of T As Structure), реализованный в .NET Framework 2.0. Такая поддержка необходима для обеспечения более удобной интеграции с данными, получаемыми из баз данных и полноценной реализации механизма LINQ. Используя такой тип данных, мы можем объявлять «пустые» версии переменных таких типов, как Integer, Boolean, Date и т. п. Синтаксис для пустых типов в Visual Basic выглядит как T.

В нашем примере со странами, можно ввести поле, содержащее, например, День независимости той или иной страны. Но так как не все страны в мире являются независимыми, мы должны определить это поле как поле, которое может содержать пустые значения:

```
Partial Class Country
  Public Property Independence As Date?
End Class
```

День независимости Республики Палау — 1 октября 1994 года, но Британские Виргинские острова являются частью Великобритании, поэтому для них День независимости будет иметь значение Nothing:

```
Dim palau = _
  New Country With { _
    .Name = "Palau", _
    .Area = 458, _
    .Population = 16952, _
    .Independence = #10/1/1994# }

Dim virginIslands = _
  New Country With { _
    .Name = "Virgin Islands", _
    .Area = 150, _
    .Population = 13195, _
    .Independence = Nothing }
```

В Visual Basic 9.0 поддерживаются следующие правила — если один из операндов арифметической операции, операций сравнения, логической или бинарной операции, операции сдвига, работы со строками или типизации имеет значение Nothing, результат операции будет Nothing.

Так как поля объектов Palau.Independence и VirginIslands.Independence имеют тип Date?, компилятор использует описанный выше подход для следующего вычисления, так как при автоматическом определении типов обе локальные переменные — PLength и VILength будут иметь тип TimeSpan?:

```
' 3980.00:00:00
Dim pLength = #8/24/2005# - Palau.Independence
```

Значение переменной PLength будет 3980.00:00:00, так как оба операнда операции имеют типы, отличные от Nothing. С другой стороны, так как значение поля VirginIslands.Independence имеет значение Nothing, результат опять будет иметь тип TimeSpan?, но значение переменной VILength будет Nothing, так как действуют описанные выше правила:

```
Dim vILength = #8/24/2005# - virginIslands.Independence ' Nothing
```

Как и в языке SQL, в операторах If и While значение Nothing интерпретируется как False и в следующем фрагменте кода будет задействована ветвь Else:

```
If vILength < TimeSpan.FromDays(10000)
  …
Else
  …
End If
```

Упрощенные делегаты

При создании делегатов, используя ключевые слова AddressOf или Handles в Visual Basic 8.0, один из методов, обеспечивающих связывание с идентификатором делегата, должен иметь точную сигнатуру типа делегата. В приведенном ниже примере сигнатура метода OnClick должна точно соответствовать сигнатуре делегата обработчика события Delegate Sub EventHandler(sender As Object, e As EventArgs), который объявлен для класса Button:

```
Dim WithEvents btn As New Button()

Sub OnClick(sender As Object, e As EventArgs) Handles B.Click
  MessageBox.Show("Hello World from" & btn.Text)
End Sub
```

Но при вызове функций и подпрограмм, не являющихся делегатами, Visual Basic не требует, чтобы аргументы в точности соответствовали вызываемым методам. Как показано ниже, мы можем вызвать подпрограмму OnClick используя аргументы типа Button и типа MouseEventArgs, которые, соответственно, являются подтипами формальных параметров Object и EventArgs:

```
Dim m As New MouseEventArgs(MouseButtons.Left, 2, 47, 11,0)
OnClick(btn, m)
```

Предположим, что мы можем определить подпрограмму — назовем ее RelaxedOnClick, которая получает два параметра типа Object, и что мы можем вызвать ее с аргументами типа Object и EventArgs:

```
Sub RelaxedOnClick(sender As Object, e As Object) Handles btn.Click
  MessageBox.Show("Hello World from" & btn.Text))
End Sub
Dim e As EventArgs = m
Dim s As Object = btn
RelaxedOnClick(btn,e)
```

В Visual Basic 9.0 связь делегатов упрощена и поддерживается соответствие вызываемому методу. Таким образом, если появляется возможность вызова функции или подпрограммы с параметрами и возвращаемыми типами, которые в точности соответствуют делегату, мы можем связать эту функцию или подпрограмму с делегатом. Другими словами, связь с делегатом и его определением следует той же логике перегрузки, что и вызов обычных методов.

В Visual Basic 9.0 мы можем связать подпрограмму RelaxedOnClick, которая имеет два аргумента типа с Object событием Click класса Button:

```
Sub RelaxedOnClick(sender As Object, e As Object) Handles btn.Click
  MessageBox.Show((“Hello World from” & btn.Text)
End Sub
```

В данном случае два аргумента обработчика события — sender и EventArgs особого значения не имеют. Вместо этого, обработчик обращается непосредственно к компоненту, для которого зарегистрировано событие и игнорирует эти аргументы. Для поддержки этого сценария делегаты могут вообще не использовать аргументов — в том случае, если не возникает неразрешимых противоречий. Другими словами, можно написать следующий код:

```
Sub RelaxedOnClick Handles btn.Click
  MessageBox.Show(“Hello World from” & btn.Text)
End Sub
```

Такое упрощение работы с делегатами распространяется и на конструирование делегатов, используя ключевое слово AddressOf, или на выражения, создающие делегаты, даже в тех случаях, когда используется позднее связывание:

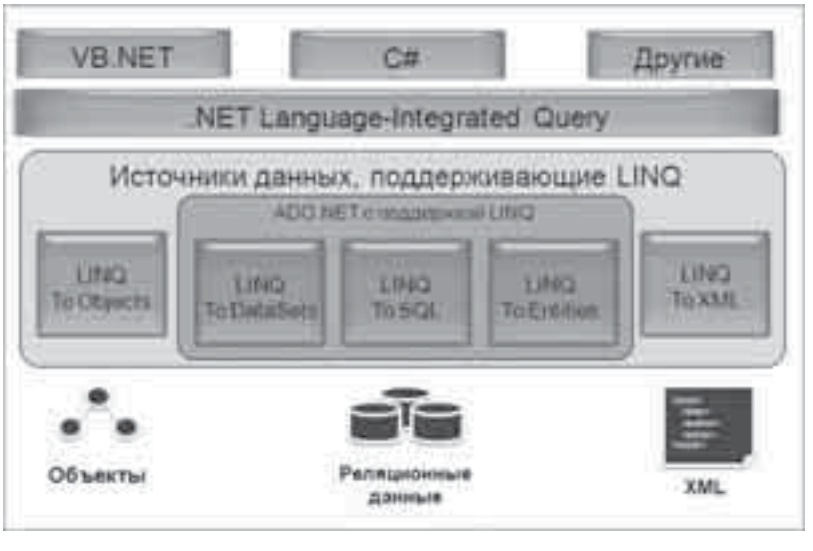
```
Dim F As EventHandler = AddressOf RelaxedOnClick
Dim G As New EventHandler(AddressOf btn.Click)
```

На этом мы завершим наше знакомство с рядом новинок в Visual Basic .NET 9.0; желающие более подробно познакомиться со всеми новинками могут обратиться к спецификации языка, которая доступна на сайте MSDN, а также к соответствующим разделам документации по .NET Frame-work 3.0 SDK и Visual Studio 2008.

Глава 2

LINQ — встроенные механизмы запросов

Помимо рассмотренных выше синтаксических расширений, языки C# 3.0 и Visual Basic .NET 9.0 обогатились встроенной поддержкой механизмов запросов на уровне языка — поддержкой технологии Language Integrated Query (LINQ). Если отложить в сторону теоретические рассуждения о важности и полезности наличия унифицированного механизма доступа к объектам, реляционным данным и XML-документам и кратко описать данную технологию, то LINQ позволяет решить две задачи — во-первых, унифицировать механизм запросов к различным объектам, пусть это будут объекты типа массивов, коллекций и т. п., находящиеся в памяти, базам данных и XML-документам и, во-вторых, интегрировать средства написания таких запросов непосредственно в язык программирования. LINQ представляет собой набор расширений для языков C# 3.0 и Visual Basic .NET 9.0, а также унифицированную программную модель на уровне библиотеки классов .NET Framework. Ниже мы посмотрим, как LINQ поддерживается в C# 3.0 — изучение аналогичных возможностей языка Visual Basic .NET 9.0 мы оставим читателям для самостоятельного упражнения.



Технология LINQ

Механизмы доступа к объектам

Наше знакомство с технологией LINQ мы начнем с рассмотрения механизмов доступа к объектам (т. н. OLINQ или Object LINQ). Любая коллекция, поддерживающая интерфейс System.Collections.Generic.IEnumerable или более общий интерфейс IEnumerable<T>, может рассматриваться в виде последовательности данных и, таким образом, может быть обработана

используя стандартные операторы запросов LINQ. Эти стандартные операторы позволяют разработчикам создавать запросы, включая возможность создания проекций в виде новых типов. Эта функциональность используется совместно с рассмотренной в предыдущей главе возможностью автоматического определения типов переменных по их инициализации.

Для того чтобы увидеть LINQ в действии, создадим следующее простое консольное приложение, содержащее метод NumQuery(), заполняющий коллекцию целочисленными значениями:

```
class Program
{
    static void Main(string[] args)
    {
    }
    static void NumQuery()
    {
        var numbers = new int[] { 1, 4, 9, 16, 25, 36 };
    }
}
```

Обратим внимание на то, что в левой части присваивания не указан тип переменной numbers — вместо этого используется новое ключевое слово var. Напомним, что это — новая возможность в C# 3.0, позволяющая указать компилятору на необходимость автоматического определения типа данных, содержащихся в указанной переменной. В нашем примере мы создаем объект типа Int32[], поэтому компилятор присвоит именно этот тип переменной numbers.

Добавим к методу NumQuery() следующий код, выполняющий запрос к коллекции в поисках целочисленных значений:

```
var evenNumbers = from p in numbers
                  where (p % 2) == 0
                  select p;
```

На данном шаге в правой части приведенного выше присваивания находится запрос в синтаксисе, поддерживаемом на уровне технологии LINQ. Как и на предыдущем шаге, для упрощения кода здесь мы используем автоматическое определение типов. Тип, возвращаемый как результат выполнения запроса, может быть не совсем очевиден: в данном примере — это System.Collections.Generic.IEnumerable<Int32>. Отметим, что иногда не представляется возможным указать тип данных, возвращаемых в результате запроса — в этом случае на помощь приходит механизм авто-определения типов.

Добавим код, выводящий результаты выполнения запроса на экран:

```
Console.WriteLine("Result:");
foreach (var val in evenNumbers)
Console.WriteLine(val);
```

Обратим внимание на то, что в выражении foreach мы также используем нетипизованную переменную. Добавим вызов метода NumQuery() в метод Main(), выполним наше приложение и убедимся в том, что на экране выводятся числа 4, 16 и 36.

Теперь посмотрим, как использовать механизм запросов к более сложным, создаваемым пользователями типам данных. Создадим класс Customer, который реализуем следующим образом:

```
public class Customer
{
    public string CustomerID { get; set; }
    public string City { get; set; }

    public override string ToString()
    {
        return CustomerID + "\t" + City;
    }
}
```

Обратим внимание на то, что при объявлении класса Customer мы использовали еще одну новинку, появившуюся в C# 3.0 — автоматическое создание свойств для полей объекта. Также отметим, что у нашего класса нет конструктора — в предыдущих версиях от разработчиков требовалось создание экземпляра объекта и вызов конструктора по умолчанию, не имевшего параметров, а затем отдельно задать значения полей объекта.

В объявление класса Program добавим новый метод — CreateCustomers(), создающий список клиентов (внимательные читатели могут заметить, что данные взяты из базы данных Northwind, ставшей уже классическим примером, используемым для демонстрации различных технологий компании Microsoft):

```
static IEnumerable<Customer> CreateCustomers()
{
    return new List<Customer>
    {
        new Customer { CustomerID = "ALFKI", City = "Berlin"    },
        new Customer { CustomerID = "BONAP", City = "Marseille" },
        new Customer { CustomerID = "CONSH", City = "London"    },
        new Customer { CustomerID = "EASTC", City = "London"    },
        new Customer { CustomerID = "FRANS", City = "Torino"    },
        new Customer { CustomerID = "LONEP", City = "Portland"  },
```

```
        new Customer { CustomerID = "NORTS", City = "London"    },
        new Customer { CustomerID = "THEBI", City = "Portland"  }
    };
}
```

Обратите внимание на то, что новая коллекция заполняется непосредственно внутри фигурных скобок. Даже несмотря на то, что тип коллекции — List<T>, а не массив, мы все равно имеем возможность добавлять элементы без непосредственного вызова метода Add(). Помимо этого, элементы коллекции Customers создаются с использованием нового синтаксиса, называемого инициализатором объектов — мы обсуждали эту функциональность в предыдущей главе. Даже при отсутствии конструктора с двумя параметрами для класса Customer мы все равно имеем возможность создавать объекты этого типа как выражения, в которых внутри фигурных скобок задаются значения полей этого объекта.

Теперь напишем запрос к нашей коллекции, возвращающий клиентов, находящихся в Лондоне. Создадим метод ObjectQuery() и добавим вызов этого метода в метод Main():

```
static void ObjectQuery()
{
    var results = from c in CreateCustomers()
        where c.City == "London"
           select c;

    foreach (var c in results)
        Console.WriteLine(c);
}

static void Main(string[] args)
{
    ObjectQuery();
}
```

Если мы выполним наш пример, то получим список из трех клиентов. Как мы увидели в данном примере, используя LINQ-запросы, работать с комплексными, задаваемыми пользователями типами также просто, как и с примитивными, встроенными типами.

Механизмы доступа к XML-документам

Механизмы доступа к XML (т. н. XLINQ или XML LINQ) позволяют работать с кэшем XML, находящимся в памяти, а также предоставляют простые способы создания XML-документов и их фрагментов. В следующем при-

мере мы посмотрим, как считывать XML-документы в объект XDocument, как выполнять запросы к элементам этого объекта и как создавать документы и их элементы на лету.

Начнем с того, что добавим к нашему проекту ссылку на пространство имен System.Xml.Linq, которое позволит нам воспользоваться механизмами доступа к XML на уровне технологии LINQ:

```
using System.Xml.Linq;
```

В следующем примере мы выполним запрос к списку клиентов в поиске тех, кто находится в Лондоне. Но в отличие от предыдущего упражнения, список клиентов будет находиться не в памяти, а в XML-файле на диске. Как мы увидим ниже, несмотря на то, что источник данных о клиентах изменился, структура запроса остается той же, что и в предыдущем примере.

Изменим наш метод CreateCustomers() из предыдущего примера на:

```
static IEnumerable<Customer> CreateCustomers()
{
    return
        from c in XDocument.Load("Customers.xml")
            .Descendants("Customers").Descendants()

        select new Customer
        {
            City = (string)c.Attribute("City"),
            CustomerID = (string)c.Attribute("CustomerID")
        };
}
```

Выполним наш пример и убедимся в том, что мы по-прежнему видим список из трех клиентов. Обратим внимание на то, что структура запроса осталось неизмененной — мы заменили только метод CreateCustomers().

В следующем примере мы посмотрим, как, используя возможности технологии LINQ, мы можем выполнять запросы к данным, находящимся в XML-файле, не загружая его в коллекцию. Предположим, что у нас нет класса Customers, в который мы загружаем данные из XML-файла. Ниже мы увидим, как обратиться к файлу напрямую, а не к коллекции, в которую загружены данные.

Добавим метод XMLQuery(), загружающий XML-документ и выводящий его содержимое на экран:

```
static void Main(string[] args)
{
    XMLQuery();
}
```

```
public static void XMLQuery()
{
    XDocument doc = XDocument.Load("Customers.xml");

    Console.WriteLine("XML Document:\n{0}",doc);
}
```

Если мы запустим данное приложение, то увидим на экране содержимое файла. Вернемся в метод XMLQuery() и выполним тот же самый запрос, что и в предыдущих примерах, возвращающий список клиентов, находящихся в Лондоне:

```
public static void XMLQuery()
{
    XDocument doc = XDocument.Load("Customers.xml");

    var results = from c in doc.Descendants("Customer")
      where (string)c.Attribute("City") == "London"
                  select c;

    Console.WriteLine("Results:");
    foreach (var contact in results)
        Console.WriteLine("\n{0}", contact);
}
```

Обратим внимание на то, что объекты, возвращаемые в результате выполнения запроса и используемые в итераторе foreach, имеют тип XElement, а не Customers, как в предыдущих примерах.

В следующем примере мы посмотрим, как мы можем преобразовать результаты выполнения запроса в новый XML-документ. Предположим, что мы хотим создать новый XML-файл, в котором будут находиться клиенты, расположенные в Лондоне. Данный файл будет отличаться структурой от файла Customers.xml — в нем каждый элемент, описывающий клиента, будет хранить название клиента и название города в виде вложенных элементов, а не в виде атрибутов, как в оригинальном файле.

Преобразуем метод XMLQuery() в следующий:

```
public static void XMLQuery()
{
    XDocument doc = XDocument.Load("Customers.xml");

          var results = from c in doc.Descendants("Customer")
                        where (string)c.Attribute("City") == "London"
                  select c;
```

```
    XElement transformedResults =
       new XElement("Londoners",
          from customer in results
          select new XElement("Contact",
             new XAttribute("ID", (string)customer.Attribute("CustomerID")),
             new XElement("Name", (string)customer.Attribute("ContactName")),
             new XElement("City", (string)customer.Attribute("City"))));

    Console.WriteLine("Results:\n{0}", transformedResults);
}
```

Временная переменная results служит для хранения данных, возвращаемых в результате выполнения запроса, перед тем как структура данных будет изменена.

Если мы запустим данное приложение, то увидим на экране те же данные, что и в предыдущем примере, но в другом формате.

Теперь сохраним результаты наших манипуляций с данными в файле. Для этого добавим в метод XMLQuery() следующую строку:

```
transformedResults.Save("Output.xml");
```

В приведенных выше примерах мы увидели, как осуществлять запросы к данным, хранящимся в XML-файлах, как трансформировать результаты запросов, выполняя преобразование структуры их представления и хранения, и как сохранять результаты таких преобразований в XML-файлах.

Механизмы доступа к базам данных

В этом разделе мы обсудим, как использовать механизм LINQ to SQL, который является частью технологии LINQ. Механизм LINQ to SQL позволяет выполнять запросы и манипулировать объектами, ассоциированными с таблицами баз данных. Использование этого механизма позволяет избежать традиционных разногласий между таблицами, хранящимися в базах данных и объектами, представляющими бизнес-логику.

Для создания объектной модели для базы данных, классы должны быть приведены в соответствие с сущностями, хранящимися в базе данных. Можно выделить три способа реализации такого приведения — можно задавать атрибуты для существующих объектов, можно использовать специальное средство, позволяющее автоматически сгенерировать объекты и использовать утилиту командной строки SQLMetal. В наших примерах мы будем использовать первые два из описанных способов.

Начнем с того, что добавим к коду нашего примера ссылки на два пространства имен, которые позволят нам использовать механизм LINQ to SQL:

```
using System.Data.Linq;
using System.Data.Linq.Mapping;
```

Теперь добавим к классу Customer ряд атрибутов для того, чтобы привести его в соответствие с таблицей базы данных Customers, которая содержит поля CustomerID и City. Ниже показано, как привести наш класс в соответствие с таблицей Customers в базе данных Northwind:

```
[Table(Name = "Customers")]
public class Customer
{
 [Column]
 public string CustomerID { get; set; }
 [Column]
 public string City { get; set; }

 public override string ToString()
 {
  return CustomerID + "\t" + City;
 }
}
```

Теперь обратимся к методу ObjectQuery() и преобразуем метод поиска клиентов, находящихся в Лондоне — выше мы уже видели, как это сделать для данных, расположенных в памяти и в XML-файле. Обратим внимание на то, что нам потребуется внести минимальные изменения для того, чтобы наш запрос мог быть обращен к данным, находящимся в базе данных. После того как мы создали соединение с базой данных, мы можем получить необходимые нам записи из таблицы Customers, выбрать те записи, которые содержат клиентов, находящихся в Лондоне, и вернуть их в виде IEnumerable<Cumstomer>. Вот код модифицированного метода ObjectQuery():

```
static void ObjectQuery()
{
 DataContext db = new DataContext
   (@"Data Source=.\sqlexpress;Initial Catalog=Northwind");
   var results = from c in db.GetTable<Customer>()
        where c.City == "London"
        select c;
   foreach (var c in results)
    Console.WriteLine("{0}\t{1}", c.CustomerID, c.City);
}
```

Запустим наше приложение и убедимся в том, что оно работает как ожидалось. В качестве дополнительной опции мы можем добавить код, показывающий SQL-запрос, выполняемый для получения данных. Вставьте следующую строку:

```
db.Log = Console.Out;
```

сразу после объявления переменной DataContext.

Теперь давайте посмотрим, как выполнить те же действия, но используя автоматически созданный объект, отраженный на таблицу базы данных. Прежде всего, удалим из нашего примера описание класса Customer. Затем выберем команду Add | New Item для добавления к проекту нового элемента и в списке Templates выберем шаблон LINQ To SQL File. В поле Name укажем имя Northwind и щелкнем кнопку Ok. Выберем команду View | Server Explorer (или нажмем последовательность клавиш Crtl+W, L), затем щелкнем кнопку Connect to database. В диалоговой панели Add Connection в поле Server name укажем имя сервера баз данных (например, .\sqlexpress), выберем базу данных Northwind в списке Select or enter a database name и щелкнем кнопку Ok.

Следующий шаг — это создание объекта представления. Откроем дерево под названием Data Conenctions, затем папку Northwind, затем папку Tables и откроем файл Northwind.dbml. Из папки с таблицами перетащим на панель методов таблицы Customers, Products, Employees и Orders. Затем из папки с хранимыми процедурами перетащим на панель методов хранимую процедуру Top Most Expensive Products. Нажмем комбинацию клавиш Ctrl+Shift+B для сборки приложения. Посмотрим на автоматически сгенерированный класс и обратим внимание на использование атрибутов. Отметим, что для баз данных с большим числом таблиц и хранимых процедур утилита SQLMetal предоставляет большее число возможностей для генерации объекта представления.

Вернемся к нашему методу ObjectQuery(). Каждая таблица представляет собой соответствующее свойство переменной db. В нашем примере запрос к базе данных будет практически аналогичным запросу, написанному в предыдущих примерах. Добавим в метод ObjectQuery() следующий код для получения списка клиентов, расположенных в Лондоне:

```
static void ObjectQuery()
{
  var db = new NorthwindDataContext();
  db.Log = Console.Out;
  var results = from c in db.Customers
            where c.City == "London"
                select c;
    foreach (var c in results)
     Console.WriteLine("{0}\t{1}", c.CustomerID, c.City);
}
```

В приведенном выше примере мы создали объект NorthwindDataContext, который представляет собой строго типизированное соединение с базой

данных. Отметим, что мы в явном виде не используем строку соединения с базой данных.

Запустим приложение на выполнение и убедимся в том, что оно работает как и предполагалось. Используя дизайнер, можно создавать отображения и на другие таблицы. Класс Customer использует отображение «один ко многим» на таблицу Orders. В следующем запросе показано, как извлечь данные из нескольких таблиц:

```
static void ObjectQuery()
{
  var db = new NorthwindDataContext();
  db.Log = Console.Out;
  var results = from c in db.Customers
                from o in c.Orders
                 where c.City == "London"
                select new { c.ContactName, o.OrderID };
  foreach (var c in results)
  Console.WriteLine("{0}\t{1}", c.ContactName, c.OrderID);
}
```

В приведенном выше примере показана конструкция select, создающая новый объект анонимного типа (новая возможность в C# 3.0). Созданный тип содержит два элемента данных — две строки с названиями, соответствующими оригинальным данным (в нашем случае — это ContactName и OrderID). Использование анонимных типов чрезвычайно полезно при работе с запросами и существенно сокращает время на создание классов, содержащих результаты выполнения запросов.

В следующем примере мы рассмотрим, как, используя технологию LINQ, мы можем не только извлекать данные, но и выполнять операции создания, обновления и удаления, т. е. выполнять все четыре типа операций, которые обычно называют CRUD-операции (Create, Read, Update, Delete).

Добавим новый метод для модификации данных в базе данных — modifyData() и соответствующий вызов в методе Main():

```
static void Main(string[] args)
{
  modifyData();
}

static void modifyData()
{
  var db = new NorthwindDataContext();
  var newCustomer = new Customer
  {
```

```
   CompanyName = “AdventureWorks Cafe”,
   CustomerID = “ADVCA”
  };

  db.Customers.Add(newCustomer);

  Console.WriteLine(“Number Created:{0}”,
  db.Customers.Where( c => c.CustomerID == “ADVCA” ).Count());
}
```

Выполним наше приложение и отметим, что на экране показан 0, а не 1 и новая запись не отображена в результатах. Это связано с тем, что мы еще не вызвали метод submitChanges(). Теперь изменим данные в базе данных — модифицируем контактную информацию для первого клиента:

```
static void modifyData()
{
 var db = new NorthwindDataContext();
 var newCustomer = new Customer
 {
  CompanyName = “AdventureWorks Cafe”,
  CustomerID = “ADVCA”
 };

 db.Customers.Add(newCustomer);
 Console.WriteLine(“Number Created:{0}”,
 db.Customers.Where( c => c.CustomerID == “ADVCA” ).Count());

 var existingCustomer = db.Customers.First();

 existingCustomer.ContactName = “New Contact”;
 Console.WriteLine(“Number Updated:{0}”,
 db.Customers.Where( c => c.ContactName == “New Contact” ).Count());
}
```

Как и в предыдущем примере, на экране будет показан 0, так как мы все еще не вызвали метод submitChanges(). Для того чтобы локальные изменения были занесены в базу данных, добавим к нашему методу modifyData() следующий код:

```
 db.SubmitChanges();

 Console.WriteLine(“Number Created:{0}”,
 db.Customers.Where( c => c.CustomerID == “ADVCA” ).Count());

 Console.WriteLine(“Number Updated:{0}”,
 db.Customers.Where( c => c.ContactName == “New Contact” ).Count());
```

Выполним наше приложение и убедимся в том, что оно работает как и ожидалось. Отметим, что после того как новый клиент занесен в базу, он не может быть занесен еще раз из-за ограничений на первичные ключи, поэтому данный пример можно запустить только один раз.

Теперь посмотрим, как с помощью технологии LINQ мы можем вызывать хранимые процедуры — помните, что в дизайнере мы добавили одну хранимую процедуру? Создадим новый метод, который будет выводить результаты выполнения хранимой процедуры Top Most Expensive Products:

```
static void InvokeSproc()
{
 var db = new NorthwindDataContext();
 foreach (var r in db.Ten_Most_Expensive_Products())
 Console.WriteLine(r.TenMostExpensiveProducts + "\t" + r.UnitPrice);
}
```

Изменим код метода Main() на следующий:

```
static void Main(string[] args)
{
    InvokeSproc();
}
```

Выполним наше приложение и изучим результаты. Отметим, что когда по каким-то причинам база данных не может быть использована через динамические запросы на языке SQL, мы можем воспользоваться языком C# 3.0 и технологией LINQ для вызова хранимых процедур, предоставляющих доступ к данным.

Запросы, которые мы выполняли выше, в основном выполняли операции фильтрации данных, но возможности LINQ этим не ограничиваются. Так, например, для сортировки клиентов, находящихся в Лондоне, мы можем использовать конструкцию orderby:

```
static void ObjectQuery()
{
 var db = new NorthwindDataContext();
 db.Log = Console.Out;
 var results = from c in db.Customers
               where c.City == "London"
               orderby c.ContactName descending
               select new { c.ContactName, c.CustomerID };
 foreach (var c in results)
  Console.WriteLine("{0}\t{1}", c.CustomerID, c.ContactName);
}
```

Для того чтобы найти число клиентов, находящихся в каждом городе, мы можем использовать конструкцию group by:

```
static void ObjectQuery()
{
 var db = new NorthwindDataContext();
 db.Log = Console.Out;
 var results = from c in db.Customers
               group c by c.City into g
               orderby g.Count() ascending
               select new { City = g.Key, Count = g.Count() };
 foreach (var c in results)
 Console.WriteLine("{0}\t{1}", c.City, c.Count);
}
```

Конструкция group by создает переменную типа IGrouping<string, Customer>, где строка содержит название города.

Часто при написании запросов возникает необходимость в поиске по двум таблицам. Обычно для этих целей используется операция join, поддерживаемая в LINQ и C# 3.0. В методе ObjectQuery() заменим существующий запрос на приведенный ниже. Вспомним, что мы отображали все заказы для каждого клиента, находящегося в Лондоне. Теперь же мы отобразим число заказов, размещенных каждым клиентом.

```
static void ObjectQuery()
{
 var db = new NorthwindDataContext();
 db.Log = Console.Out;
 var results = from c in db.Customers
               join e in db.Employees on c.City equals e.City
               group e by e.City into g
               select new { City = g.Key, Count = g.Count() };
 foreach (var c in results)
 Console.WriteLine("{0}\t{1}", c.City, c.Count);
}
```

В этом разделе мы рассмотрели основные возможности технологии LINQ, которая может использоваться для работы с данными, хранимыми в структурах и объектах, в базах данных и XML-документах. Более подробно о технологии LINQ и ее возможностях см. раздел, посвященный этой технологии на сайте MSDN.

Глава 3

Основные изменения в .NET Framework 3.5

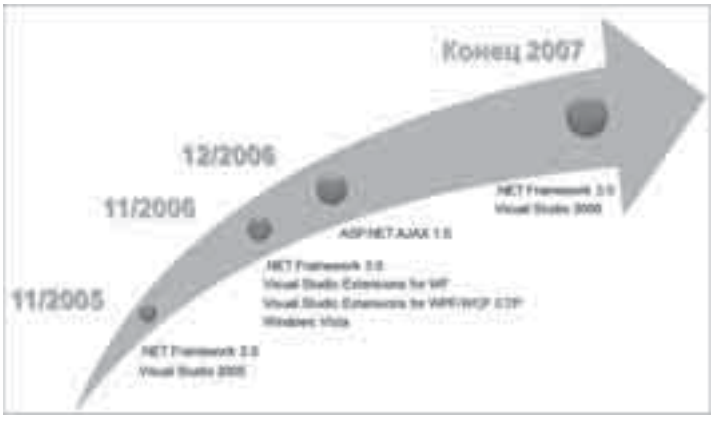
Новая версия библиотеки классов .NET Framework 3.5 является продолжением линейки .NET Framework и включает функциональность предыдущих версий. Библиотека классов .NET Framework 3.5 включает .NET Frame-work 3.0 SP1, которая, в свою очередь, включает .NET Framework 2.0 SP1. Такая комбинация различных версий библиотек классов является основой для поддержки написания кода для нескольких версий .NET Framework (multi-targeting), которая впервые появилась в Visual Studio 2008. С точки зрения установки библиотеки классов, достаточно установить .NET Frame-work 3.5, пакет установки сам установит версии 2.0 и 3.0.

В следующей таблице показаны основные версии .NET Framework с момента ее появления в 2002 году.

	2002	2003	2005	2006	2007
Библиотека	NexFx 1.0	NetFx 1.1	NetFx 2.0	NetFx 3.0	NetFx 3.5
Ядро	CLR 1.0	CLR 1.1	CLR 2.0	CLR 2.0	CLR 2.0

Кратко вспомним основные моменты, связанные с развитием .NET Framework — они (за исключением самых ранних версий) показаны на приведенной ниже иллюстрации.

- 2001 — .NET Framework 1.0 и Visual Studio .NET.
- 2003 — .NET Framework 1.1 и Visual Studio 2003.
- 2005 — .NET Framework 2.0 и Visual Studio 2005 (вместе с выходом SQL Server 2005).
- Ноябрь 2006 — .NET Framework 3.0 (WCF, WPF, WF, CardSpace), а также расширения для Visual Studio, поддерживающие новые технологии в .NET 3.0.
- Декабрь 2006 — официальный выпуск ASP.NET AJAX 1.0.
- Ноябрь 2007 — .NET Framework 3.5 и Visual Studio 2008.
- Весна 2008 — обновления для поддержки SQL Server 2008.



При разработке новой версии .NET Framework была предпринята попытка внесения минимальных изменений в существующие библиотеки. Практически все новинки были реализованы в новых библиотеках классов (в новых динамически загружаемых библиотеках), а новинки на уровне базовой библиотеки классов были реализованы в новых классах. Ниже перечислены новые сборки, появившиеся в .NET Framework 3.5.

- **System.DirectoryServices.AccountManagement.dll** — набор управляемых программных интерфейсов для работы с пользователями, группами и т. п.; базируется на интерфейсах SDS/ADSI и поддерживает как Active Directory, так и Active Directory Application Mode (ADAM, в Windows Server 2008 называется Active Directory Lightweight Directory Services).
- **System.Management.Instrumentation.dll** — расширения провайдера WMI.NET с улучшенной масштабируемостью при работе с большими коллекциями WMI-данных.
- **System.Web.Extensions.dll** — поддержка расширений для веб-приложений, включая ASP.NET AJAX, Client Application Services и ASP.NET 3.5 Controls.
- **System.Net.dll** — сетевые коммуникации, включая поддержку Peer-To-Peer.
- **System.WorkflowServices.dll** — поддержка интеграции Windows Communication Foundation и Windows Workflow Foundation.
- **System.ServiceModel.Web.dll** — поддержка протокола HTTP для веб-приложений.
- **System.Data.Linq.dll, System.Xml.Linq.dll** — поддержка технологии LINQ to SQL и LINQ to XML.
- **System.Data.DataSetExtensions.dll** — поддержка технологии LINQ to DataSet.

- **System.AddIn.dll, System.AddIn.Contract.dll** — поддержка механизмов расширения.
- **System.Core.dll** — расширения в базовой библиотеке классов: потоки, безопасность, TimeZone, диагностика, коллекции, WMI, LINQ и т. д.

Ниже мы рассмотрим новинки, появившиеся в базовой библиотеке классов (Base Class Library, BCL), а остальные изменения и дополнения — в соответствующих разделах, посвященных созданию Windows-приложений, веб-приложений, сервисов и т. д.

Новинки в базовой библиотеке классов

Начнем с расширений для работы с датой и временем – для поддержки временной зоны в .NET Framework 3.5 появились два новых типа данных:

- **TimeZoneInfo** — расширенная поддержка временной зоны, включая перечисление существующих зон, преобразование и сериализацию;
- **DateTimeOffset** — тип данных DateTime с поддержкой смещения относительно универсального времени (Coordinated Universal Time, UTC).

Рассмотрим несколько примеров, поясняющих использование DateTimeOffset. Создание новой структуры DateTimeOffset возможно одним из показанных ниже способов:

```
DateTimeOffset now = DateTimeOffset.Now;
DateTimeOffset now2 = DateTime.Now;
DateTimeOffset now3 = new DateTimeOffset(DateTime.Now);
DateTime tvShowPremiere = new DateTime(2007, 6, 13, 8, 0, 0);
DateTimeOffset tvShowPremiereUTC = new
DateTimeOffset(tvShowPremiere.ToUniversalTime());
DateTimeOffset tvShowPremiereLA = new DateTimeOffset(tvShowPremiere, new
TimeSpan(-7, 0, 0));
DateTimeOffset tvShowPremiereNYC =
 new DateTimeOffset(tvShowPremiere, new TimeSpan(-4, 0, 0));
DateTimeOffset tvShowPremiereLocal = tvShowPremiere;
DateTimeOffset utcNewYear07 =
 new DateTimeOffset(2007, 1, 1, 0, 0, 0,
 new TimeSpan(0, 0, 0));
DateTimeOffset seattleNewYear07 =
 new DateTimeOffset(2007, 1, 1, 0, 0, 0,
 new TimeSpan(-8, 0, 0));
```

Для преобразования DateTimeOffset в DateTime можно использовать один из следующих способов:

```
DateTime dtSeatleNewYear07Utc = seattleNewYear07.UtcDateTime;
DateTime dtSeattleNewYear07Local = seattleNewYear07.LocalDateTime;
```

а для обработки DateTimeOffset – один из показанных ниже способов:

```
DateTimeOffset dateA =
 DateTimeOffset.Parse("4/15/2006 6:00 AM -7:00");

DateTimeOffset dateB =
 DateTimeOffset.Parse("4/15/2006 6:00:00 AM -7:00");

DateTimeOffset dateC =
 DateTimeOffset.Parse("-7:00 4/15/2006 6:00 AM");

DateTimeOffset dateD =
 DateTimeOffset.Parse("4/15/2006 -7:00 6:00 AM");
```

С появлением новых типов данных, связанных с датой и временем, может возникнуть вопрос: когда следует использовать DateTime, а когда — DateTimeOffset? Ниже приведены основные рекомендации по выбору между двумя типами данных.

- Используйте **DateTimeOffset** в тех случаях, когда вы описываете точное время, например при вычислении значения «сейчас», времени изменения файлов, времени возникновения события и т. п. Если текущая временная зона не известна, используйте UTC.
- Используйте **DateTime** в тех сценариях, когда абсолютная величина времени не важна и не критична временная зона, например для хранения времени открытия и закрытия магазина.
- Используйте **DateTime** для сценариев взаимодействия с другими системами и обмена данными, в тех случаях, когда информация состоит из даты и времени, но не содержит данных о временной зоне — передача через OLE Automation, базы данных, существующие в .NET программные интерфейсы, которые используют тип данных DateTime и т. д.
- Используйте **DateTime** с компонентом времени в виде 00:00:00 для представления дат.
- Используйте **TimeSpan** для представления времени без использования даты.

Как видно из приведенных выше рекомендаций, тип данных **DateTimeOffset** предлагается использовать во всех типовых сценариях работы с датой и временем. В новых программных интерфейсах базовой библиотеки классов планируется использование именного этого типа данных. Это не означает, что тип данных DateTimeOffset пришел на смену типу данных **DateTimeOffset** — последний должен использоваться в описанных выше сценариях.

Тип данных **TimeZoneInfo** используется для хранения информации о временной зоне. Вот пример использования данного типа для извлечения информации о временных зонах, хранящейся на локальном компьютере:

```
using System;

public class TimeZoneInfoSample
{
 private static void Main()
 {
  String id = “Alaskan Standard Time”;
  TimeZoneInfo tzi;

  try
   {
    tzi = TimeZoneInfo.FindSystemTimeZoneById(id);
   }
  catch (TimeZoneNotFoundException e)
   {
    Console.WriteLine(id + “ not found on the local computer: “ + e);
    return;
   }
  catch (InvalidTimeZoneException e)
   {
    Console.WriteLine(id + “ is corrupt on the local computer: “ + e);
    return;
   }
  Console.WriteLine(“TimeZoneInfo.Id = “ + tzi.Id);
  Console.WriteLine(“TimeZoneInfo.DisplayName = “ + tzi.DisplayName);
  Console.WriteLine(“TimeZoneInfo.StandardName = “ + tzi.StandardName);
  Console.WriteLine(“TimeZoneInfo.DaylightName = “ + tzi.DaylightName);
  Console.WriteLine(“TimeZoneInfo.BaseUtcOffset = “ + tzi.BaseUtcOffset);
  Console.WriteLine(“TimeZoneInfo.SupportsDaylightSavingTime = “ +
   tzi.SupportsDaylightSavingTime);
  }
}
```

Следующая новинка — это тип **HashSet<T>**, реализованный в пространстве имен **System.Collections.Generic.HashSet**. Этот тип представляет собой неупорядоченную коллекцию уникальных элементов и, помимо стандартных операций типа Add, Remove, Contains, поддерживает такие операции, как объединение, пересечение и поиск симметричных отличий. Посмотрите следующие примеры, показывающие использование этого типа данных.

```
HashSet<int> theSet1 = new HashSet<int>();
theSet1.Add(1);
theSet1.Add(2);
theSet1.Add(2);
// в theSet1 теперь находятся 1,2

HashSet<int> theSet2 = new HashSet<int>();
theSet2.Add(1);
theSet2.Add(3);
theSet2.Add(4);
// в theSet2 теперь находятся 1,3,4

theSet1.UnionWith(theSet2);
// в theSet1 теперь находятся 1,2,3,4
```

Операция **Add()** по умолчанию возвращает значение типа **bool**, позволяющее узнать, был ли добавлен в коллекцию данный элемент или нет. Таким образом, в примерах, приведенных выше, мы можем проверить, был ли уже в коллекции добавляемый элемент:

```
bool added = theSet1.Add(2); // Значение added равно true
added = theSet1.Add(2);      // Значение added равно false
```

Обратим внимание на уникальность элементов — **HashSet** определяет равность элементов, используя метод **EqualityComparer()**, который вы можете предоставить, или метод **EqualityComparer()**, используемый по умолчанию для того или иного типа данных. В приведенном выше примере мы не указали метод **EqualityComparer()** — это означает, что будет использоваться метод по умолчанию для сравнения данных типа **Int32**. В следующем примере мы будем использовать метод **OddEvenComparer()**, который считает элементы равными, если они либо оба четные, либо оба нечетные.

```
class OddEvenComparer : IEqualityComparer<int> {
public OddEvenComparer() {}
public bool Equals(int x, int y) {
return (x & 1) == (y & 1);
}

public int GetHashCode(int x) {
 return (x & 1);
}
}

// Как использовать новый метод сравнения
```

```
HashSet<int> oddEvenSet = new HashSet<int>(new OddEvenComparer());
oddEvenSet.Add(1);
oddEvenSet.Add(3);
oddEvenSet.Add(4);
// oddEventSet содержит 1,4; 1 и 3 расцениваются как равные значения
```

Обратим внимание на метод **UnionWith** в первом примере. Этот метод, как и другие операторы для работы с наборами данных, модифицирует набор данных, для которого он вызывается, а не создает новый набор. Это важно, так как в технологии LINQ операции типа Union, Intersect и т. п. над интерфейсом **IEnumerable** создают новый набор данных. Таким образом, методы HashSet не дублируют аналогичные методы LINQ — они предоставляются для использования в тех случаях, когда не требуется создание нового набора данных и в названиях этих методов всегда присутствует ключевое слово **With**.

Расширения в System.Diagnostics

В пространстве имен System.Diagnostics появилось два новых класса — **EventSchemaTraceListener**, который служит для перенаправления отладочной или трассировочной информации в файл-протокола с сохранением данных в формате XML в соответствии с заданной схемой и **Eventing.EventProviderTraceListener**, который представляет собой парный класс для класса System.Diagnostics.TraceSource с возможностью записи данных в строку, разделенную заданным символом разделителя, а также ряд расширений класса **Eventing.Reader** и **PerfomanceData**. Последний поддерживается только в Windows Vista и Windows Server 2008 и представляет собой реализацию ряд программных интерфейсов библиотеки advapi32.dll на управляемом коде.

Расширения в System.Security.Cryptography

Пространство имен System.Security.Cryptography пополнилось поддержкой ряда алгоритмов, включая Advanced Encryption Standard (AES) на уровне базового класса, от которого должны наследовать классы, реализующие данный стандарт. Помимо этого, реализована поддержка алгоритмов Elliptic Curve Diffie Hellman и Elliptic Curve Digital Signature Algorithm. Также в пространстве имен System.Security.Cryptography появился ряд новых Hash-алгоритмов. Поддерживаемые алгоритмы показаны в следующей таблице.

Алгоритм	Класс	Операционная система
MD5	MD5Cng	Windows Vista
SHA-1	SHA1Cng	Windows Vista
SHA-256	SHA256CryptoServiceProvider	Windows 2003
	SHA256Cng	Windows Vista
SHA-384	SHA384CryptoServiceProvider	Windows 2003
	SHA384Cng	Windows Vista
SHA-512	SHA512CryptoServiceProvider	Windows 2003
	SHA512Cng	Windows Vista

В следующей таблице показаны поддерживаемые в пространстве имен System.Security.Cryptography алгоритмы симметричного шифрования:

Алгоритм	Класс	Операционная система
AES	AesCryptoServiceProvider	Windows XP SP2
	AesManaged	Все поддерживаемые платформы

и асимметричного шифрования:

Алгоритм	Класс	Операционная система
Elliptic Curve DSA	ECDSACng	Windows Vista
Elliptic Curve Diffie Hellman	ECDiffieHellmanCng	Windows Vista

Поддержка каналов

Новое пространство имен **System.IO.Pipes** содержит классы, предназначенные для реализации синхронных и асинхронных коммуникаций между процессами через именованные и анонимные каналы (pipe). В данном пространстве имен реализованы следующие классы:

Класс	Описание
AnonymousPipeClientStream	Представляет собой клиентскую часть анонимного канала, связанного с потоком, который поддерживает как синхронные, так и асинхронные операции чтения и записи
AnonymousPipeServerStream	Представляет собой поток поверх анонимного канала, который поддерживает как синхронные, так и асинхронные операции чтения и записи

Класс	Описание
NamedPipe-ClientStream	Представляет собой клиентскую часть именованного канала, связанного с потоком, который поддерживает как синхронные, так и асинхронные операции чтения и записи
NamedPipe-ServerStream	Представляет собой поток поверх именованного канала, который поддерживает как синхронные, так и асинхронные операции чтения и записи
PipeAccessRule	Абстрактный класс для access control entry (ACE), задающей правила доступа к каналу
PipeAuditRule	Абстрактный класс для access control entry (ACE), задающей правила аудита использования канала
PipeSecurity	Представляет собой средства управления доступом к каналу и аудитом безопасности
PipeStream	Представляет собой поток поверх канала, который поддерживает как анонимные, так и именованные каналы

Как мы увидели выше, в .NET Framework 3.5 входит поддержка использования каналов из приложений, написанных на управляемом коде. Каналы используются для организации коммуникаций между процессами, выполняющимися на одной машине, или процессами, выполняющимися на различных Windows-машинах, объединенных в локальную сеть. Обеспечивается поддержка как анонимных, так и именованных каналов. Новые типы данных, связанных с поддержкой каналов, реализованы в библиотеке **System.Core.dll** в пространстве имен **System.IO.Pipes**.

Анонимные каналы используются для полудуплексного обмена текстовой информацией и не поддерживают работу в сети — только в рамках одного экземпляра сервера. Такие каналы чаще всего используются для обмена сообщениями между потоками или между родительскими и дочерними процессами — в том случае, когда ссылка на канал может быть передана дочернему процессу при его создании.

В следующем примере показано, как передать текстовое сообщение от родительского процесса дочернему процессу. Родительский процесс считывает строку из консоли и передает ее дочернему процессу. Дочерний процесс выводит полученную от родительского процесса строку на экран.

```
// Родительский процесс
Process process = new Process();
process.StartInfo.FileName = “child.exe”;
using (AnonymousPipeServerStream pipeStream =
    new AnonymousPipeServerStream(PipeDirection.Out,
HandleInheritability.Inheritable))
{
    process.StartInfo.Arguments = pipeStream.GetClientHandleAsString();
    process.StartInfo.UseShellExecute = false;
```

```
    process.Start();
     pipeStream.DisposeLocalCopyOfClientHandle();
     using (StreamWriter sw = new StreamWriter(pipeStream)) {
        sw.AutoFlush = true;
        sw.WriteLine(Console.ReadLine());
    }
}
 process.WaitForExit();
 process.Close();
// Дочерний процесс
using (StreamReader sr = new StreamReader(
    new AnonymousPipeClientStream(PipeDirection.In, args[0]))) {
    string line;
    while ((line = sr.ReadLine()) != null) {
        Console.WriteLine("Echo: {0}", line);
    }
}
```

По сравнению с анонимными каналами, именованные каналы обладают более широким набором свойств. Они поддерживают дуплексный обмен информацией, работу в сети, могут использовать несколько экземпляров сервера. Более того, именованные каналы поддерживают передачу сообщений переменной длины и механизмы имперсонации, позволяющие подсоединяющимся процессам использовать собственные наборы прав на удаленных серверах.

В тех случаях, когда второй процесс не может унаследовать ссылку на канал, используются именованные каналы. В следующем примере показано, как пересылаются строки от клиентского процесса серверному процессу.

```
// Серверный процесс
using (NamedPipeServerStream pipeStream = new NamedPipeServerStream("testpipe"))
{
    pipeStream.WaitForConnection();
   using (StreamReader sr = new StreamReader(pipeStream))
 {
        string temp;
        while ((temp = sr.ReadLine()) != null)
       {
            Console.WriteLine("{0}: {1}", DateTime.Now, temp);
       }
 }
}
// Клиентский процесс
using (NamedPipeClientStream pipeStream = new NamedPipeClientStream("testpipe")) {
    pipeStream.Connect();
```

```
    using (StreamWriter sw = new StreamWriter(pipeStream)) {
        sw.AutoFlush = true;
        string temp;
        while ((temp = Console.ReadLine()) != null) {
            sw.WriteLine(temp);
   }
  }
}
```

Именованные каналы также поддерживают передачу сообщений. Это позволяет считывающему процессу получать сообщения переменной длины от посылающего процесса. В следующем примере показано, как такие сообщения посылаются и считываются.

```
// Процесс 1
UTF8Encoding encoding = new UTF8Encoding();
 string message1 = "Named Pipe Message Example.";
string message2 = "Another Named Pipe Message Example.";
Byte[] bytes;
 using (NamedPipeServerStream pipeStream = new
        NamedPipeServerStream("messagepipe", PipeDirection.InOut, 1,
        PipeTransmissionMode.Message, PipeOptions.None)) {
    pipeStream.WaitForConnection();
 // Посылка двух сообщений
    bytes = encoding.GetBytes(message1);
    pipeStream.Write(bytes, 0, bytes.Length);
    bytes = encoding.GetBytes(message2);
    pipeStream.Write(bytes, 0, bytes.Length);
}
// Процесс 2
Decoder decoder = Encoding.UTF8.GetDecoder();
Byte[] bytes = new Byte[10];
Char[] chars = new Char[10];
 using (NamedPipeClientStream pipeStream =
        new NamedPipeClientStream("messagepipe")) {
    pipeStream.Connect();
    pipeStream.ReadMode = PipeTransmissionMode.Message;
     int numBytes;
    do {
        string message = "";
        do {
            numBytes = pipeStream.Read(bytes, 0, bytes.Length);
            int numChars = decoder.GetChars(bytes, 0, numBytes, chars, 0);
            message += new String(chars, 0, numChars);
```

```
            } while (!pipeStream.IsMessageComplete);
            decoder.Reset();
            Console.WriteLine(message);
        } while (numBytes != 0);
}
```

Три приведенных выше примера показывают, насколько просто осуществлять коммуникации между процессами, используя новые классы, появившееся в .NET Framework 3.5.

Коммуникации «Peer-to-Peer»

Два новых пространства имен — **System.Net.PeerToPeer** и **System.Net.PeerToPeer.Collaboration** содержат классы и дополнительные структуры для реализации коммуникаций типа «точка-точка» (Peer-to-Peer). Пространство имен **System.Net.PeerToPeer** обеспечивает поддержку базовых коммуникационных механизмов, а пространство имен **System.Net.PeerToPeer.Collaboration** — расширенные механизмы.

Одна из проблем, которая возникает при организации соединений «точка-точка» — это собственно обнаружение «точек» и обмен идентификаторами, затрудненный отсутствием записей об адресах, таких как, например, DNS (Domain Name System). Инфраструктура Peer-to-Peer в Windows решает эту проблему за счет механизма обнаружения имен и схемы обнаружения точек, которая называется PNRP. Протокол Peer Name Resolution Protocol (PNRP) обеспечивает механизм обнаружения имен для сетей «точка-точка» — он разработан таким образом, чтобы имена обнаруживались и присваивались точкам типа **IPEndPoint**.

Базовая функциональность протокола Peer Name Resolution Protocol обеспечивается классами, реализованными в пространстве имен **System.Net.PeerToPeer**. Эти классы обеспечивают возможность создания «точек» и их регистрации в «облаке» (Cloud) существующих «точек». Помимо этого, классы поддерживают возможность обнаружения имени (**PeerName**) для подключения к соответствующей сетевой «точке», порту TCP, «облаку» или другим возможным идентификаторам адреса.

Расширенная функциональность работы в режиме «точка-точка» обеспечивается классами, реализованными в пространстве имен **System.Net.PeerToPeer.Collaboration**.

В Windows Vista протокол PNRP доступен в виде сервиса, который запускается приложениями при необходимости организации соединений «точка-точка». Помимо этого, существует сервис регистрации PNRP, который может быть вручную включен приложениями или пользователями в тех случаях, когда им требуется публикация имен для обеспечения удаленных соединений и коммуникаций.

Версия протокола PNRP 1.0 входит в состав Windows XP Service Pack 2 (SP2), Windows XP Professional x64 Edition, Windows XP Service Pack 1 (SP1) и Advanced Networking Pack for Windows XP.

Пространство имен **System.Net.PeerToPeer.Collaboration** содержит классы, расширяющие сетевую функциональность классов из пространства имен System.Net.PeerToPeer. В частности, поддерживается возможность создания коммуникационных сессий без использования сервера. Инфраструктура для организации соединений «точка-точка» может использоваться в качестве основы для создания различных типов приложений, включая сетевые игры, проведение телеконференций, а также другие виды активностей, предполагающие участие нескольких человек. Инфраструктура, не требующая использования серверов, включает набор программных интерфейсов, которые облегчают процесс обнаружения «точек», отсылки приглашений на подключение, обнаружения «точек» в рамках одной подсети и управления контактами.

Программные интерфейсы Native Peer Collaboration API поддерживаются в 32-битных и 64-битных изданиях Windows Vista, но не доступны в Windows Server 2008.

Механизмы расширений

Пространства имен **System.AddIn**, **System.AddIn.Contract**, **System.AddIn.Hosting** и **System.AddIn.Pipeline** содержат ряд классов, используя которые можно создавать приложения, поддерживающие расширения (addins). Расширение — это код, написанный сторонним разработчиком, который динамически загружается и активируется хост-приложением в определенном контексте (запуск хост-приложения, загрузка документа, возникновение события, вызов функции и т. п.). Такой код предоставляет хост-приложению дополнительные сервисы, может управлять приложением используя объектную модель, поддерживаемую хост-приложением.

Можно выделить следующие ключевые сценарии использования механизмов расширения:

- Провайдер сервисов.
  - Модуль расширения обеспечивает дополнительные сервисы для хост-приложения. Например, система проверки орфографии для текстового редактора.
  - Функциональность хост-приложения может зависеть от провайдера сервисов.
- Управление хост-приложением.
  - Хост-приложение обеспечивает сервисы для модуля расширения.
  - Пример — модули расширения для приложений Microsoft Office, которые могут быть реализованы как на уровне шаблонов документов, так и на уровне самих приложений.

- Расширения функциональности хост-приложений.
  - ☐ Хост-приложение обеспечивает ключевые точки расширяемости.
  - ☐ Пример — расширения Microsoft CRM, располагаемые в Microsoft Outlook.

Можно выделить три категории разработчиков, относящихся к созданию и использованию модулей расширения. К первой относятся разработчики расширений. Создаваемый ими код может выглядеть так:

```
[AddIn("Sample AddIn", Version="1.0.0.0")]
public class AddIn : Calculator
```

Вторая категория — это разработчики хост-приложений, которые включают в приложения возможность использования моделей расширения:

```
IList<AddInToken> tokens =
   AddInStore.FindAddIns(typeof(AddInType),
  addinPath);
foreach (AddInToken token in tokens)
{
  token.Activate<AddInType>
   (new AddInProcess(), AddInSecurityLevel.FullTrust);
}
```

К третьей категории отнесем разработчиков объектной модели хост-приложения, которая может использоваться модулями расширения для управления хост-приложением и расширения его функциональности.

Кратко перечислим основные возможности, предоставляемые механизмами расширения, реализованными в .NET Framework 3.5.

- Обнаружение.
  - ☐ Используются специальные типы и папки с предопределенными именами.
  - ☐ Получение метаданных без непосредственной загрузки сборки или активации модуля расширения.
  - ☐ Отсутствие необходимости в регистрации модулей расширения.
  - ☐ Отдельное хранилище для модулей расширения.
- Активация.
  - ☐ Выбор контекста изоляции (домены приложений и/или процессы).
  - ☐ Создание очереди из загруженных сборок.
  - ☐ Создание экземпляров компонентов (модулей расширения).
  - ☐ Типизация согласно типам, используемым хост-приложением.
- Изоляция.
  - ☐ Изоляция между хост-приложением и модулем расширения.

- ☐ Изоляция между модулями расширения.
- ☐ Границы изоляции — домены приложений, процесс и домены приложений внутри них.

■ Возможность выгрузки.

- ☐ Освобождение ресурсов.
- ☐ Возможность выгрузки модулей расширения на уровне хост-приложения.
- ☐ Возможность выгрузки модулей расширения на основе жизненного цикла приложения.

■ Безопасность.

- ☐ Модули расширения могут выполняться с уровнем привилегий меньшим, либо равным уровню привилегий хост-приложения.
- ☐ Три уровня привилегий — Full Trust, Internet, Intranet.
- ☐ Поддержка настраиваемых наборов привилегий.

■ Управление жизненным циклом.

- ☐ «Прозрачная» сборка мусора.
- ☐ Возможность выгрузки модулей расширения.
- ☐ Выгрузка модулей расширения на основе жизненного цикла приложения.

Пространство имен **System.AddIn** содержит атрибут **AddInAttribute**, который используется для идентификации модулей расширения и их обнаружения. Пространство имен **System.AddIn.Contract** содержит интерфейсы и структуры, которые обеспечивают основу для обмена данными между компонентами (например, основное приложение и модуль расширения), каждый из которых может обновляться самостоятельно. Компоненты могут использовать интерфейсы и структуры из пространства имен **System.AddIn.Contract** для обмена данными между процессами или приложениями, а также для коммуникаций с другими компонентами в рамках одного процесса или домена приложения. Интерфейсы, реализованные в пространстве имен **System.AddIn.Contract**, часто называют контрактами. Все контракты наследуют от интерфейса IContract, который играет ту же роль для компонентов, созданных средствами .NET Framework, что и интерфейс IUnknown для COM-компонентов. Для определения, реализует ли объект то или иной контракт, используется метод **QueryContract**.

Пространства имен **System.AddIn.Contract.Automation** и **System.AddIn.Contract.Collections** содержат дополнительные контракты, расширяющие функциональность пространства имен **System.AddIn.Contract**. Пространство имен System.AddIn.Contract.Automation включает контракты, которые компоненты могут использовать для доступа к информации о типах и активизации определенных членов типов. Пространство имен

System.AddIn.Contract.Collections содержит контракты, определяющие коллекции объектов типа **IContract** и **RemoteArgument**.

Пространство имен **System.AddIn.Hosting** обеспечивает классы для обнаружения, регистрации, активизации и управления модулями расширения, а пространство имен **System.AddIn.Pipeline** включает классы для создания коммуникационных каналов между хост-приложением и модулями расширения.

В следующей главе мы рассмотрим основные изменения в Visual Studio 2008, связанные с созданием Windows-приложений.

Глава 4

Создание Windows-приложений

Создание Windows-приложений, или, как их еще называют, «толстых клиентов», возможно в Visual Studio 2008 с использованием двух ключевых компонентов .NET Framework — Windows Forms (появился в .NET Framework 1.0) и Windows Presentation Foundation (WPF, появился в ноябре 2006 г. в составе .NET Framework 3.0).

Технология Windows Forms

Технология Windows Forms (пространство имен **System.Windows.Forms**) представляет собой набор .NET классов, реализующих функциональность, присущую стандартным и расширенным компонентам Windows-приложений. Разработчики используют формы и диалоговые окна в качестве контейнеров для размещения различных интерфейсных элементов — кнопок, меню, полей ввода, полей выбора и т. п. Каждый такой интерфейсный элемент представлен соответствующим классом и поддерживает набор методов, свойств и обработчиков событий, которые можно использовать как для управления представлением компонентов и обеспечения реакции на действия пользователей, производимые с этими компонентами. Классы, реализованные в пространстве имен **System.Windows.Forms**, можно разделить на несколько функциональных групп, которые мы кратко обсудим ниже.

Формы и компоненты (Control, UserControl, Form)

Большинство классов, определенных в пространстве имен System.Windows.Forms, наследуют от класса **Control**. Данный класс обеспечивает базовую функциональность для всех компонентов, которые могут отображаться на форме. Класс Form представляет собой окно Windows-приложения. К окнам относятся диалоговые панели, немодальные окна, MDI-клиенты и родительские окна. В качестве основы для создания собственных интерфейсных элементов следует использовать класс **UserControl**.

Меню, панели задач и панели статуса

В состав Windows Forms входит большой набор классов, позволяющих создавать собственные панели задач и меню, поддерживающие различные визуальные стили и представления. Для создания панелей задач, меню, контекстных меню и панелей статуса следует использовать, соответственно, классы **ToolStrip**, **MenuStrip**, **ContextMenuStrip** и **StatusStrip**.

Интерфейсные элементы

В пространстве имен System.Windows.Forms содержится большое число классов, реализующих основные интерфейсные элементы Windows-приложений. Ряд компонентов — **TextBox** или **ComboBox** предназначен для ввода данных. Другие компоненты — **Label** и **ListView** служат для отображения

данных. В пространстве имен System.Windows.Forms также содержатся компоненты, используемые для активизации каких-либо действий — например **Button**. Для отображения HTML-документов в Windows-приложениях используется компонент **WebBrowser** и соответствующие классы, например **HtmlDocument**. Компонент **MaskedTextBox** представляет собой расширенное средство ввода данных на основе шаблонов. При необходимости создания собственного дизайнера форм, можно использовать компонент **PropertyGrid**, который отображает таблицу свойств компонентов, которые можно изменять.

Управление расположением компонентов

Ряд классов из пространства имен System.Windows.Forms используются для управления расположением компонентов на формах или в составе других компонентов. К таким классам относятся **FlowLayoutPanel, TableLayoutPanel** и **SplitContainer**. Класс **FlowLayoutPanel** представляет собой панель, которая динамически располагает свое содержимое горизонтально или вертикально. Класс **TableLayoutPanel** служит для задания ячеек и рядов для расположения элементов в фиксированной таблице. Класс **SplitContainer** используется для разделения формы на две или более частей с возможностью изменения размеров каждой части.

Использование данных

В Windows Forms поддерживаются богатые возможности по связи компонентов с различными источниками данных — базами данных и XML-файлами. Компонент **DataGridView** представляет собой настраиваемую таблицу для отображения данных с возможностью управления ячейками, рядами, колонками и рамками. Компонент **BindingNavigator** служит для навигации по данным, отображаемым в форме и обычно используется совместно с компонентом **BindingSource** для перемещения по записям в источнике данных.

Диалоговые панели общего назначения

В Windows поддерживается ряд диалоговых панелей общего назначения, которые используются для унификации таких пользовательских операций, как открытие и сохранение файлов, манипуляции со шрифтами и цветами, а также вывод данных на устройства печати. Классы **OpenFileDialog** и **SaveFileDialog** реализуют, соответственно, отображение диалоговых панелей, позволяющих пользователям выбрать местоположение файла, который планируется открыть или сохранить. Класс **FontDialog** отображает диалоговую панель для выбора шрифтов, используемых в приложении. Класс **ColorDialog** используется для отображения стандартной панели выбора цветов и задания собственных цветовых комбинаций.

Классы **PageSetupDialog**, **PrintPreviewDialog** и **PrintDialog** используются для отображения диалоговых панелей, позволяющих пользователям управлять различными аспектами вывода информации на устройства печати. Сами же функции управления печатью реализованы в пространствах имен **System.Drawing.Printing** и **System.Windows.Forms.Printing**.

Помимо стандартных диалоговых панелей в пространстве имен **System.Windows.Forms** реализован класс **MessageBox**, который может использоваться в различных ситуациях, требующих отображения панели сообщений и интерактивного ввода данных от пользователей.

Дополнительные компоненты

Помимо рассмотренных выше компонентов, в пространстве имен **System.Windows.Forms** есть ряд классов, которые не наследуют от класса **Control**, но тем не менее представляют собой визуальные компоненты, которые можно использовать при создании Windows-приложений. К таким классам относятся, например, классы **ToolTip, ErrorProvider**, **Help** и **HelpProvider**. Класс **ToolTip** используется для отображения краткого описания назначения компонента. Класс **ErrorProvider** служит для отображения информации об ошибках, а классы **Help** и **HelpProvider** — для отображения справочной информации для пользователей приложения.

Дочерние пространства имен **System.Windows.Forms** отвечают за расширенную функциональность, связанную с разработкой и выполнением Windows-приложений. К таким пространствам имен относятся:

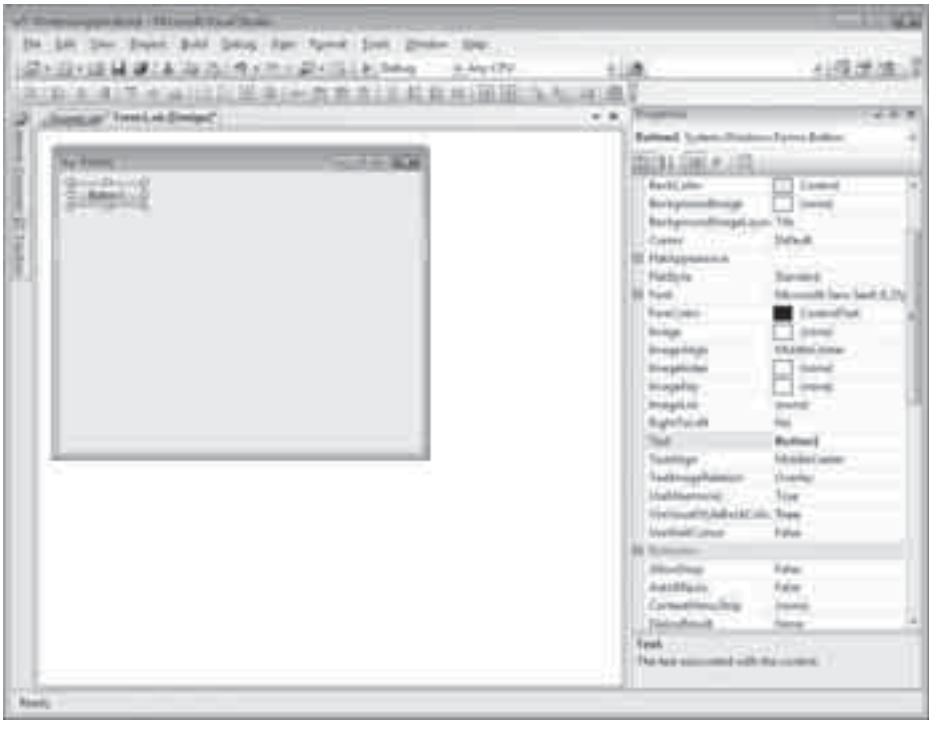
- **System.Windows.Forms.ComponentModel.Com2Interop** — содержит вспомогательные классы для отображения панелей свойств компонентов в режиме дизайна;
- **System.Windows.Forms.Design** — содержит классы, используемые для обеспечения конфигурируемости компонентов Windows Forms в режиме дизайна. Эти классы состоят из класса-дизайнера и набора сервисов, доступных в режиме дизайна. Также в данном пространстве имен есть класс UITypeEditor, используемый для установки значений ряда типов свойств, и классы, обеспечивающие импорт компонентов ActiveX;
- **System.Windows.Forms.Design.Behavior** — содержит классы для создания расширений пользовательского интерфейса и поддержки таких расширений в режиме дизайна. В данное пространство имен включены следующие классы:
  - ☐ Класс **Glyph** для обработки отрисовки и щелчков мышью;
  - ☐ Класс **Adorner** для управления коллекцией классов типа Glyph;
  - ☐ Класс **BehaviourService** для управления расширениями интерфейсных элементов в режиме дизайна;

- Класс **SnapLine** для отображения горизонтальных и вертикальных сегментов, позволяющих пользователям выравнивать компоненты в режиме дизайна.

- **System.Windows.Forms.Integration** — обеспечивает интеграцию между компонентами Windows Forms и Windows Presentation Foundation (WPF). Содержит классы, позволяющие использовать компоненты Windows Forms на страницах, созданных средствами Windows Presentation Foundation и компоненты Windows Presentation Foundation в приложениях на основе Windows Forms. Два ключевых класса обеспечивают интеграционные возможности двух технологий — **WindowsFormsHost** и **ElementHost**. Класс **WindowsFormsHost** используется при необходимости отображения компонента Windows Forms на странице, созданной средствами Windows Presentation Foundation. Класс **ElementHost** служит для отображения элементов WPF в приложениях на основе Windows Forms.
- **System.Windows.Forms.Layout** — содержит классы, используемые для реализации механизмов расположения элементов на формах или в составе других компонентов. К таким классам относятся классы **LayoutEngine** и **ArrangedElementCollection**. Класс **LayoutEngine** служит в качестве абстрактного класса, используемого компонентами **TableLayoutPanel** и **FlowLayoutPanel** для реализации различных способов расположения компонентов. Класс **ArrangedElementCollection** представляет собой коллекцию объектов, которые находятся под управлением **LayoutEngine**.
- **System.Windows.Forms.PropertyGridInternal** — обеспечивает поддержку реализации компонента **PropertyGrid**, который используется для отображения таблицы свойств компонентов в режиме дизайна;
- **System.Windows.Forms.VisualStyles** — содержит классы, обеспечивающие отображение компонентов и других интерфейсных элементов Windows-приложений с использованием визуальных стилей. В состав данного пространства имен входят следующие основные классы:
  - **VisualStyleElement** является базовым классом для задания визуальных стилей, поддерживаемых компонентом или интерфейсным элементом. Помимо **VisualStyleElement** в пространстве имен **System.Windows.Forms.VisualStyles** содержится большое число вложенных классов, унаследованных от **VisualStyleElement**, статические свойства которых возвращают визуальные свойства для каждого возможного состояния компонента, части компонента или интерфейсного элемента, поддерживающего визуальные стили. Например, свойство **Pressed** класса **VisualStyleElement.Button.PushButton** возвращает тип **VisualStyleElement**, идентифицирующий визуальный стиль для кнопки в нажатом состоянии;

- **VisualStyleRenderer** предоставляет методы для отрисовки и получения информации о каждом визуальном элементе (**VisualStyleElement**) для визуального стиля, поддерживаемого операционной системой — размер по умолчанию, тип фона и фонового изображения, набор цветов и т. п.;
- **VisualStyleInformation** предоставляет набор статических свойств для получения информации о текущем визуальном стиле, поддерживаемом операционной системой.

Разработка Windows-приложений выполняется на основе шаблона **Windows Forms Application**, доступного в ветви Windows для выбранного языка программирования — Visual Basic .NET или Visual C#. Входящий в состав Visual Studio 2008 дизайнер позволяет создавать Windows-приложения перетаскиванием компонентов из палитры доступных компонентов (ToolBox) на поверхность формы и написанием кода для обработки событий, возникающих при активизации пользователями тех или иных компонентов, расположенных на форме, а также событий, отражающих жизненный цикл самого Windows-приложения.



Дизайнер Windows-приложений в Visual Studio

Панель задания свойств компонентов позволяет управлять различными аспектами визуальных элементов, располагаемых на формах — при-

вязкой к форме, размером, цветом, шрифтами и т. п. Щелчок мышью по компоненту приводит к появлению редактора кода, в котором разработчики описывают действия, выполняемые при активизации того или иного интерфейсного элемента — например действия при нажатии кнопки или выборе того или иного элемента меню, а также описывают другие части приложения, например действия, которые должны выполняться при начальной загрузке приложения или завершении работы с приложением.

Возможность переключения между визуальным дизайном и редактором кода позволяет максимально наглядно выполнять как собственно разработку интерфейса приложения, так и создавать код, реализующий логику работы самого приложения.

Палитра компонентов для Windows-приложений

Отображение данных

Как было отмечено выше, технология Windows Forms позволяет не только создавать интерфейсы Windows-приложений, используя входящие в ее состав разнообразные классы, реализующие компоненты интерфейса, но и использовать эти интерфейсные элементы для отображения данных, располагаемых в базах данных, XML-файлах, извлекаемых из веб-сервисов или других источников.

Источники данных, доступные для приложения в режиме разработки, перечислены в специальном окне Visual Studio — **Data Sources**. В данном окне отображаются различные источники данных — базы данных, веб-сервисы, объекты в рамках данного проекта и т. п. Создание компонентов, обес-

печивающих связь с данными, осуществляется простым перетаскиванием источника данных на форму, включенную в состав проекта. Также создать компоненты связи с данными можно перетаскиванием объектов из окна **Data Sources** на существующие компоненты, расположенные на форме.

В состав Windows Forms входит компонент, специально разработанный для табличного отображения данных — **DataGridView**. Разработчикам предоставляются широкие возможности по управлению данным компонентом — можно изменять настройки ячеек, фиксировать размеры и местоположение определенных рядов и колонок, отображать компоненты внутри ячеек и т. п.

Для обеспечения связи с источниками данных используется компонент **BindingSource**, который впервые появился в Visual Studio 2005 в составе .NET Framework 2.0. Этот компонент представляет собой соединение с источником данных и предоставляет методы для связи компонентов с данными, для перемещения по записям, редактирования записей и сохранения изменений непосредственно в источнике данных. Компонент **BindingNavigator** обеспечивает простой пользовательский интерфейс на основе компонента **BindingSource**, который позволяет перемещаться между записями в источнике данных.

Развертывание приложений — технология ClickOnce

После того как приложение создано и отлажено, необходимо обеспечить возможность передачи этого приложения конечным пользователям. При использовании технологии ClickOnce вы можете развертывать приложения непосредственно из Visual Studio, указав адрес веб-сайта, на котором будет расположено ваше приложение. Технология ClickOnce управляет всеми элементами и зависимостями вашего приложения и гарантирует корректную установку приложения на клиентском компьютере.

Приложения, развертываемые с помощью ClickOnce, могут быть сконфигурированы таким образом, чтобы они запускались только при наличии сетевого соединения или при наличии и отсутствии такового. При указании на то, что приложение должно поддерживать работу в отсоединенном режиме, ClickOnce добавляет ссылку на приложение в меню Start на пользовательском компьютере — в этом случае приложение запускается без обращения к нему через URL.

При обновлении приложения вы публикуете новый манифест развертывания (deployment manifest) и новую копию приложения на веб-сервере. Технология ClickOnce обнаруживает наличие обновлений и обновляет пользовательские версии, заменяя компоненты приложения и сборки на более новые версии — все это происходит автоматически — для реализации этой функциональности не требуется написания дополнительного кода.

В Visual Studio 2008 и .NET Framework 3.5 был реализован ряд улучшений в технологии ClickOnce. К этим улучшениям относятся поддержка развертывания приложений с нескольких сайтов без необходимости в изменении манифестов и возможность управления внешним видом пакетов инсталляции (third-party branding). Утилита **Mage.exe**, используемая для генерации и редактирования манифестов, используемых для развертывания приложений средствами ClickOnce, была обновлена до версии 3.5 и теперь поддерживает все ключевые возможности данной технологии.

Развертывание приложений, используя механизмы **Windows Installer**, теперь поддерживает новую версию .NET Framework (.NET Framework Launch Condition поддерживает версии 3.0 и 3.5) и полностью совместимо с Windows Vista даже при установке с использованием User Account Control (UAC).

Сервисы для клиентских приложений

Сервисы для клиентских приложений — это новинка в .NET Framework 3.5, позволяющая Windows-приложениям (включая приложения, созданные средствами Windows Forms и Windows Presentation Foundation) обращаться к сервисам ASP.NET — идентификация, роли и профили. Эти сервисы позволяют разработчикам аутентифицировать пользователей и получить данные о пользовательских ролях и настройках приложения, хранящихся на сервере.

Для включения поддержки сервисов для клиентских приложений необходимо сконфигурировать провайдеров клиентских сервисов в конфигурационном файле приложения или в дизайнере проектов Visual Studio. Эти провайдеры подключаются к серверу через модель расширения и позволяют приложениям обращаться к веб-сервисам через стандартные программные интерфейсы для получения данных об аутентификации, ролях и настройках. Сервисы клиентских приложений также поддерживают работу в отсоединенном режиме — в этом случае используются данные, сохраненные в локальном кэше данных.

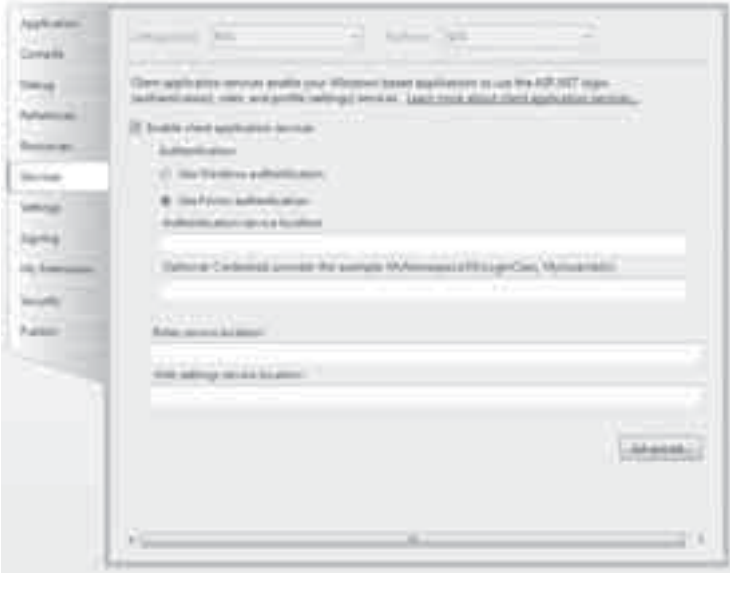
Для проверки пользователей через существующие сервисы аутентификации ASP.NET AJAX используется статический метод **Membership.Validate User**. Этот метод обращается к провайдеру клиентских сервисов и возвращает значение типа Boolean, указывающее на результат проверки пользователя.

Для получения ролевой информации от ролевых сервисов ASP.NET AJAX используется метод **IsInRole** интерфейса **IPrincipal**, получаемого из статического свойства **Thread.CurrentPrincipal**. Метод **IsInRole** требует указания имени роли в качестве параметра и возвращает значение типа Boolean, указывающее на принадлежность пользователя указанной роли.

Сервисы клиентских приложений также можно использовать для получения пользовательских настроек, хранящихся в сервисе профилей

ASP.NET AJAX. Клиентский сервис обращения к настройкам на сервере интегрирован с сервисами хранения прикладных настроек, которые появились в .NET Framework 2.0. Для получения настроек необходимо сгенерировать класс **Settings** для вашего проекта, используя вкладку **Settings** в дизайнере проектов Visual Studio. На странице **Settings** следует использовать кнопку **Load Web Settings** для получения настроек с сервера и добавления их в класс **Settings**, который будет содержать настройки для всех аутентифицированных пользователей или всех анонимных пользователей.

На C# настройки доступны через **Properties.Settings.Default**, а на Visual Basic — через **My.Settings**.



Конфигурация сервисов клиентских приложений

Следующие классы обеспечивают поддержку сервисов для клиентских приложений.

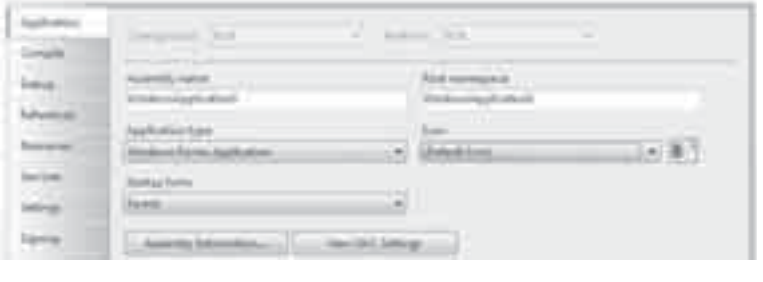
Класс	Описание
ClientFormsIdentity	Управляет идентификацией пользователей и authentication cookies, используемых при аутентификации на базе форм
ClientRolePrincipal	Управляет пользовательскими ролями
ConnectivityStatus	Поддерживает статическое свойство **IsOffline**, позволяющее переключить приложение в отсоединенный режим
ClientFormsAuthenticationCredentials	Представляет собой учетную запись пользователя

Класс	Описание
ClientForms-Authentication MembershipProvider	Управляет доступом к удаленным сервисам аутентификации, используемым при аутентификации на базе форм
ClientWindows-Authentication-MembershipProvider	Управляет Windows-аутентификацией
ClientRoleProvider	Управляет доступом к удаленным ролевым сервисам
ClientSettingsProvider	Управляет доступом к удаленным сервисам конфигурации
IClientForms-Authentication-CredentialsProvider	Этот интерфейс обеспечивает дополнительный способ получения пользовательских данных из клиентского приложения
SettingsSavedEventArgs	Предоставляет свойство **FailedSettingsList**, которое используется в обработчике события **ClientSettings-Provider.SettingsSaved**
UserValidatedEventArgs	Предоставляет свойство **UserName**, которое используется в обработчике события **UserValidated**

Поддержка Windows Vista

Существующие приложения, созданные с использованием технологии Windows Forms, должны работать под управлением Windows Vista без каких-либо проблем. Более того, в большинстве случаев интерфейс таких приложений будет обновлен таким образом, как будто эти приложения были специально написаны для Windows Vista. Так, например, обновлены диалоговые панели общего назначения (см. классы типа **CommonFileDialog**). Помимо этого, в .NET Framework 3.5 обеспечена поддержка иконки Shield (свойство **Shield** в классе **SystemIcons**), используемой в User Account Control (UAC).

В среде разработки Visual Studio 2008 также появилась возможность автоматической генерации манифестов для Windows-приложений. Доступ к манифесту по умолчанию (файл **app.manifest**) осуществляется щелчком кнопки **View UAC Settings** в панели настроек свойств проекта на вкладке **Application**.



Панель настройки свойств проекта

Технология Windows Presentation Foundation

Вторая технология для создания Windows-приложений — это Windows Presentation Foundation (WPF). WPF — это один из ключевых компонентов .NET Framework 3.0, который пополнился рядом расширений и дополнений в новой версии библиотеки классов — .NET Framework 3.5. Приложения на основе WPF создаются с помощью специального шаблона проекта — WPF Application, который доступен для типов проектов Windows на языках программирования Visual Basic .NET и Visual C#.

Создание WPF-приложений практически не отличается от создания приложений на основе Windows Forms — вы перетаскиваете компоненты на форму и создаете обработчики событий. Но есть ряд существенных отличий — в среде разработчика помимо дизайнера, окна настройки свойств и галереи компонентов есть еще окно, которое содержит код на языке XAML — eXtensible Application Markup Language. Этот язык используется в WPF для описания пользовательского интерфейса.

Например, на языке XAML описание основного окна приложения выглядит следующим образом:

```
<Window x:Class="Window1"
    xmlns="http://schemas.microsoft.com/winfx/2006/xaml/presentation"
    xmlns:x="http://schemas.microsoft.com/winfx/2006/xaml"
    Title="Window1" Height="300" Width="300">
    <Grid>
    </Grid>
</Window>
```

При разработке приложения с использованием Windows Forms, вы можете перетащить компонент из галереи на форму или, при необходимости, написать код, инициализирующий и создающий требуемый компонент. При перетаскивании компонента на форму весь необходимый код генерируется автоматически. При создании WPF-приложения вы также можете создавать компоненты на языке XAML или перетаскивать их из галереи на окно WPF-приложения.

Язык XAML представляет иерархическую организацию элементов, каждый из которых заключен в угловые скобки. Например, кнопка будет описана так:

```
<Button></Button>
```

С помощью атрибутов описывается внешний вид интерфейсных элементов. Атрибуты состоят из названия, символа присваивания (=) и значения атрибута в кавычках. Например, для задания высоты кнопки мы можем написать:

```
<Button Height="23"></Button>
```

При перетаскивании компонентов из галереи в дизайнер, Visual Studio автоматически генерирует соответствующий XAML-код. Например, следующий код будет сгенерирован при перетаскивании кнопки на форму:

```
<Button Height="23" HorizontalAlignment="Left" Margin="10,10,0,0"
    Name="Button1" VerticalAlignment="Top" Width="75">Button</Button>
```

Двойной щелчок мышью по компоненту приводит к генерации кода обработчика события, который может выглядеть так:

```
Sub ButtonOKClicked(ByVal Sender As Object, _
    ByVal e As RoutedEventArgs)

End Sub
```

Компоненты WPF-приложений

Как мы обсудили выше, в WPF-приложениях, подобно приложениям на основе Windows Forms, также используются компоненты. Ниже мы кратко рассмотрим основные группы компонентов, используемых в WPF-приложениях. Все они описаны в пространстве имен **System.Windows. Controls**.

Группа	Компоненты
Кнопки	Button RepeatButton
Диалоговые панели	OpenFileDialog PrintDialog SaveFileDialog
Отрисовка	InkCanvas InkPresenter
Документы	DocumentViewer FlowDocumentPageViewer FlowDocumentReader FlowDocumentScrollViewer StickyNoteControl
Ввод информации	TextBox RichTextBox PasswordBox

Группа	Компоненты
Расположение элементов	Border
	BulletDecorator
	Canvas
	DockPanel
	Expander
	Grid
	GridView
	GridSplitter
	GroupBox
	Panel
	ResizeGrip
	Separator
	ScrollBar
	ScrollViewer
	StackPanel
	Thumb
	Viewbox
	VirtualizingStackPanel
	Window
	WrapPanel
Медиа	Image
	MediaElement
	SoundPlayerAction
Меню	ContextMenu
	Menu
	ToolBar
Навигация	Frame
	Hyperlink
	Page
	NavigationWindow
	TabControl
Выбор	CheckBox
	ComboBox
	ListBox
	TreeView
	RadioButton
	Slider

Группа	Компоненты
Отображение информации	AccessText
	Label
	Popup
	ProgressBar
	StatusBar
	TextBlock
	ToolTip

Ключевые сервисы Windows Presentation Foundation

Ввод данных и генерация команд

Часто компонентам требуется обнаружение пользовательских действий и обеспечение реакции на такие действия. В системе ввода данных в WPF поддерживаются как прямые, так и перенаправленные события, которые могут использоваться для ввода текстовой информации, управления фокусом, позиционирования «мыши» и т. п.

Расположение элементов

При создании приложений разработчики располагают компоненты с помощью дизайнера. Ключевым требованием для подсистемы управления расположением элементов является динамичная реакция на изменения размеров окна или настройки дисплея. В состав WPF входит система уп-

равления расположением элементов, которая позволяет разработчикам не заботиться об изменениях в размерах окон или величине шрифта — все необходимые корректировки интерфейса происходят автоматически.

Система управления расположением интерфейсных элементов базируется на ряде компонентов, входящих в состав WPF:

- **Canvas** — дочерние компоненты могут сами управлять своим расположением.
- **DockPanel** — дочерние компоненты выравниваются по краям панели.
- **Grid** — дочерние компоненты располагаются в рядах и колонках.
- **StackPanel** — дочерние компоненты накладываются друг на друга вертикально или горизонтально.
- **VirtualizingStackPanel** — дочерние компоненты выравниваются по одной линии — горизонтальной или вертикальной.
- **WrapPanel** — дочерние компоненты располагаются слева направо и «заворачиваются» на следующую строку, когда заканчивается место на текущей строке.

Связь с данными

В основе поддержки возможности WPF-отображать данные из различных источников лежит класс **Binding**, в задачу которого входит связь определенного компонента с источником данных. Так же как и в Windows Forms, в WPF поддерживаются механизмы отображения данных, навигации по данным и внесения изменений в оригинальные данные в случае их изменений в компоненте.

Все, что мы рассматривали выше, в той или иной степени можно реализовать и при помощи технологии Windows Forms. Реальные преимущества использования технологии Windows Presentation Foundation заключаются в мощном графическом ядре и программном интерфейсе для отрисовки различных графических элементов, которые мы рассмотрим ниже.

Графика

В Windows Presentation Foundation реализовано расширяемое, масштабируемое и гибкое графическое ядро, с помощью которого можно работать с графикой, не зависящей от текущего разрешения и аппаратной платформы, использовать 2- и 3-мерные графические примитивы и анимацию, а также, при необходимости, задействовать все доступные аппаратные ресурсы системы для получения максимальной производительности.

2-мерные примитивы

В WPF поддерживаются основные 2-мерные векторные примитивы — прямоугольники, эллипсы и т. п. Такие примитивы могут быть интерактивными — они реагируют на ввод с клавиатуры или манипуляции «мышью».

2-мерная геометрия

2-мерные примитивы, включенные в состав WPF, подходят для решения большинства задач, но иногда могут потребоваться какие-то специальные формы — в этом случае используется 2-мерная геометрия, поддерживаемая в WPF для отрисовки замкнутых или открытых форм, кривых и т. п. 2-мерная геометрия может использоваться для обрезания изображений, тестирования ввода или для отрисовки 2-мерных данных.

2-мерные эффекты

В состав поддержки 2-мерной графики, реализованной в Windows Presentation Foundation, входит набор визуальных эффектов, таких как градиентная заливка, использование битовых масок, графических изображений, эффекты с использованием видео, вращение, масштабирование и ряд других.

3-мерная отрисовка

В состав Windows Presentation Foundation также входит поддержка 3-мерной графики и отрисовки, интеграция которой с 2-мерной графикой позволяет создавать различные представления данных.

Анимация

Использование поддержки анимации, реализованной в Windows Presentation Foundation, позволяет управлять размерами элементов, поворачивать их на любой угол, изменять их визуальные характеристики и т. п. Большинство классов, входящих в состав WPF, поддерживают анимацию — это утверждение распространяется даже на классы, создаваемые разработчиками.

Медиа и графика

В Windows Presentation Foundation полноценно реализована поддержка аудиовизуальных данных (используя для этого элемент **MediaElement**) — таким образом, в приложениях можно использовать графические изображения, видео- и аудиоинформацию.

Текст и шрифты

Для обеспечения отрисовки текстовой информации в состав Windows Presentation Foundation включены следующие возможности:

- Поддержка шрифтов OpenType;
- Расширения технологии ClearType;
- Возможность использования аппаратных ускорителей;
- Возможность интеграции текстовой информации, мультимедийных данных, графики и анимации;

- Поддержка локальных версий шрифтов и возможность подстановки шрифтов в случае их отсутствия.

Документы

В Windows Presentation Foundation реализована полноценная поддержка работы с тремя типами документов:

- Документы с нефиксированным отображением (flow documents). Такие документы оптимизированы для просмотра и отображения и могут динамически реагировать на изменения размеров окон или настроек дисплея.
- Документы с фиксированным отображением (fixed documents). Такие документы предназначены для точного отображениях их содержимого (принцип «what you see is what you get», WYSIWYG) и вывода на устройство печати.
- Документы, созданные в соответствии со спецификацией XML Paper Specification (XPS). Такие документы являются расширенной версией документов с фиксированным отображением. XPS-документы описываются XML-схемой и представляют собой постраничное отображение электронного документа. Спецификация XPS является открытой, кроссплатформенной и позволяет существенно упростить создание, распространение, вывод на устройства печати и архивирование документов.

  К основным характеристикам XPS-документов относятся:

  - Упаковка XPS-документов в виде файлов **ZipPackage**, которые соответствуют спецификации Open Packaging Conventions (OPC). Для этого используется программный интерфейс **System.IO.Packaging**.
  - Возможность отображения документов в браузере и обычном приложении.
  - Возможность создания и управления документами из WPF-приложений.
  - Поддержка на уровне спулера Windows Vista.
  - Поддержка на уровне принтеров ведущих производителей.
  - Интеграция с компонентом **DocumentViewer**, используемым для отображения документов.

Помимо этого, в WPF обеспечивается поддержка создания, отображения и управления документами, возможность включения аннотаций и вывод документов на устройства печати.

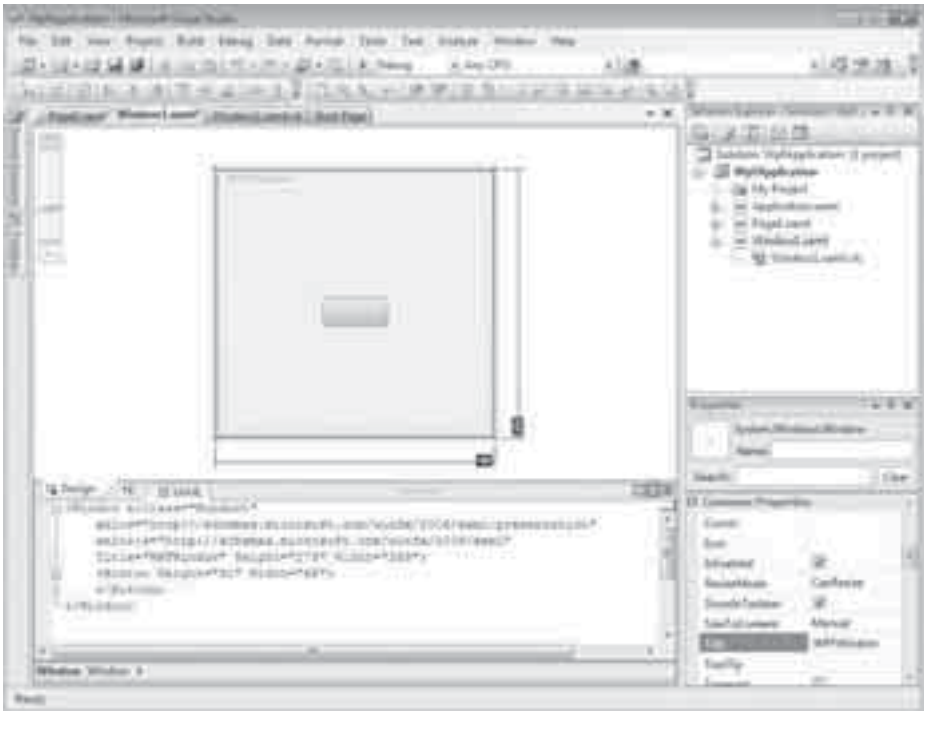
Типы WPF-приложений

Существует два типа WPF-приложений — т. н. полноценные приложения, которые выполняются как обычные Windows-приложения, и приложения, выполняемые в браузере (т. н. XAML Browser Applications, XBAP-приложения).

При создании полноценных приложений вы оперируете такими элементами, как Window для создания окон и диалоговых панелей, которые могут быть доступны при активации соответствующих команд меню или панелей задач. Помимо этого возможно использование панелей сообщений и стандартных диалоговых панелей — **MessageBox**, **OpenFileDialog**, **SaveFileDialog** и **PrintDialog**.

При создании приложений, выполняемых в браузере, вы создаете страницы (**Page**) и страничные функции (**PageFunction(T)**). Навигация по страницам осуществляется с помощью гипертекстовых ссылок (класс **Hyperlink**). WPF-приложения могут размещаться в Internet Explorer версий 6 и 7. Существует две дополнительные возможности хостинга WPF-приложений в браузере — использование фреймов для отображения страниц в рамках окна или замена всего окна на компонент **NavigationWindow**.

Основные новинки в Visual Studio 2008



Дизайнер WPF-приложений

Самой важной новинкой в Visual Studio 2008 стало включение в состав продукта полноценного дизайнера WPF-приложений. Данный дизайнер поддерживает создание и редактирование XAML-файлов с полной поддерж-

кой Intellisense, интеграцию с редакторами кода на VB.NET и C# для написания обработчиков событий, возможность работы с несколькими окнами — например в одном окне отображается XAML-код, в другом — его визуальное представление, а также режим дизайна с использованием Visual Design Surface с возможностью перетаскивания компонентов из галереи редактирования их свойств в специальном окне и использованием компонентов-контейнеров типа **Grid**, **Canvas** и **DockPanel** или **StackPanel** для расположения компонентов на форме.

Основные новинки в Windows Presentation Foundation 3.5

Ниже мы рассмотрим основные изменения и дополнения в версии Windows Presentation Foundation 3.5.

Совместимость с версией 3.0

- Все приложения, созданные для WPF 3.0, будут корректно работать под управлением WPF 3.5.
- Если приложения, созданные для WPF 3.5, используют функции, поддерживаемые в версии 3.0, они будут корректно работать под управлением WPF 3.0.
- В WPF 3.5 используется новое пространство имен — http://schemas.microsoft.com/netfx/2007/xaml/presentation. Его рекомендуется использовать при создании приложений для WPF 3.5. Помимо этого, также можно использовать и пространство имен, определенное для WPF 3.0.

Прикладная модель

В версии 3.5 появился ряд изменений в прикладной модели WPF. К таким изменениям, в частности, относятся:

- Поддержка моделей расширения для визуальных и невизуальных компонентов, выполняющихся как самостоятельные приложения или как XBAP-приложения. Модули расширения создаются на базе классов, реализованных в пространстве имен **System.AddIn** и компонента **FrameworkElement**.
- Поддержка выполнения XBAP-приложений под управлением браузера Firefox 2.0 — для этого необходимо использовать plug-in for WPF 3.5.
- Возможность обмена cookie между XBAP- и веб-приложениями, выполняющимися в рамках одного сайта.
- Улучшенная поддержка IntelliSense для XAML.
- Расширенная поддержка локализации приложений.

Графика

У разработчиков появилась возможность кэшировать графические изображения, загружаемые по протоколу HTTP в локальном кэше (Microsoft Internet Explorer Temporary File Cache) — таким образом последующие загрузки файлов будут происходить с локального диска, а не из Интернета. В зависимости от размера графического изображения, вы можете получить значительные улучшения в производительности приложений. Возможность кэширования поддерживается следующими функциями:

- BitmapImage.UriCachePolicy.
- BitmapDecoder.Create(Uri, BitmapCreateOptions, BitmapCacheOption, RequestCachePolicy).
- BitmapFrame.Create(Uri, RequestCachePolicy).
- BitmapFrame.Create(Uri, BitmapCreateOptions, BitmapCacheOption, RequestCachePolicy).

Событие **BitmapSource.DecodeFailed** возникает в тех случаях, когда невозможна загрузка графического изображения из-за нарушений данных в заголовке изображения.

3-мерная графика

В объектной модели поддержки 3-мерной графики появились следующие расширения:

- Обеспечивается поддержка концепций **UIElement** — ввод данных, фокус, событийная модель и т. п. Новые классы — **UIElement3D** и унаследованные от него **ContainerUIElement3D** и **ModelUIElement3D**, появившиеся в WPF 3.5, полностью поддерживают эти возможности.
- Новый класс — **Viewport2DVisual3D** — обеспечивает поддержку размещения интерактивных 2-мерных элементов в 3-мерных объектах.
- Реализованы новые сервисы трансформации — GeneralTransform3D и GeneralTransform2DTo3D. Класс GeneralTransform3DTo2D позволяет выполнять трансформации между объектами Visual3D и 2- и 3-мерными объектами в обе стороны.

Связь с данными

В WPF 3.5 появился ряд расширений для механизмов связи компонентов с данными. К ним относятся:

- Новый механизм отладки, позволяющий более просто отлаживать связи компонентов с данными. Новое свойство — **PresentationTraceSources.TraceLevel**, которое может использоваться с объектами, участвующими в обеспечении связи с данными, позволяет получать информацию о статусе текущей связи. Статический класс **PresentationTraceSources** реализован в пространстве имен **System.Diagnostics**.

- Проверка на уровне модели данных с поддержкой интерфейса **IDataErrorInfo**, который позволяет бизнес-объектам проверять валидность вводимых данных. Данный интерфейс задает индексатор, который получает на вход имя свойства, а возвращает строку, используя новое правило проверки вводимой информации — **DataErrorValidationRule**.
- В классах **Binding** и **MultiBinding** появилось два новых свойства — **ValidatesOnExceptions** и **ValidatesOnDataErrors**, которые могут использоваться вместо правил **ExceptionValidationRule** и **DataErrorValidationRule**.
- Модель связи с данными с поддержкой LINQ и XLINQ на уровне **BindingListCollectionView**. Этот механизм позволяет осуществлять связь с данными через коллекции **BindingList(T)**.

Компоненты

В Windows Presentation Foundation 3.5 произошел ряд изменений в некоторых компонентах — мы кратко рассмотрим эти изменения ниже.

- В компоненте **TextBox** появилась поддержка Input Method Editor (IME). В классе **FrameworkTextComposition** появился ряд новых свойств — **CompositionOffset**, **CompositionLength**, **ResultOffset** и **ResultLength**. Класс **FrameworkTextComposition** используется как свойство **TextCompositionEventArgs.TextComposition**, когда пользователь вводит текст в компоненте **TextBox**, используя IME и возникают события **TextInput**, **TextInputUpdate** или **TextInputStart**.
- В компоненте **RichTextBox** теперь сохраняются расширенные объекты типа **TextElement** — это происходит при сохранении объекта **TextElement** и при работе с областью обмена данными. Такая возможность поддерживается на уровне следующих программных интерфейсов:
  - Класс TextRange содержит новый метод **Save (Stream, String, Boolean)**, последний параметр которого указывает, требуется ли поддержка объектов **TextElement** или нет.
  - Класс **TextElementEditingBehaviorAttribute** позволяет указать атрибуты для расширенного объекта **TextElement** — если значения атрибутов **TextElementEditingBehaviorAttribute.IsMergeable** и **TextElementEditingBehaviorAttribute.IsTypographicOnly** равны **false**, **RichTextBox** сохраняет содержимое объекта **TextElement** при редактировании содержимого **RichTextBox**. Новое свойство **IsDocumentEnabled**, появившееся у компонента **RichTextBox**, позволяет указать, должны ли интерфейсные элементы принимать данные от пользователя.
- Класс **TextBoxBase** пополнился новым свойством — **UndoLimit**, которое позволяет указать максимальное число действий, запоминаемых компонентом.

- Компонент **SoundPlayerAction** теперь может загружать аудио-файлы, расположенные по URI-адресам.

Документы

У классов **FlowDocumentPageViewer**, **FlowDocumentScrollViewer** и **FlowDocumentReader** появилось свойство **Selection**, которое возвращает **TextSelection**, представляющее выбранный в документе фрагмент.

Аннотации

Поддержка механизмов аннотаций расширилась возможностью обнаружения соответствий аннотаций аннотированным объектам. Новый интерфейс — **IAnchorInfo** и новый метод — **GetAnchorInfo** добавлены в класс **AnnotationHelper**, который используется для реализации описанной выше функциональности.

Взаимодействие Windows Forms и Windows Presentation Foundation

Для обеспечения взаимодействия Windows Presentation Foundation и Windows Forms существует два подхода. Первый заключается в хостинге WPF-приложений в окне WIn32 — в этом случае вы сможете использовать богатые графические возможности WPF в обычных Windows-приложениях. Ключевой компонент для реализации данного подхода — **HwndSource**, который обеспечивает взаимодействие двух технологий на уровне ссылок на окно (window handle).

Второй подход решает обратную задачу — хостинг Win32-окна в WPF-приложении. В этом случае у вас появляется возможность хостинга стандартных компонентов в WPF-приложениях и обмена данными между двумя типами приложений. Ключевой компонент для реализации данного подхода — класс **HwndHost**, который представляет собой окно в виде WPF-элемента, который может быть добавлен в стандартную цепочку WPF-элементов. Класс **HwndHost** предоставляет программную модель, которая может использоваться для решения различных задач интеграции двух технологий, например для обеспечения обработки сообщений, поступающих окну.

На этом мы завершим наше знакомство с ключевыми новинками в .NET Framework 3.5 и Visual Studio 2008, связанными созданием Windows-приложений. В следующем разделе мы обсудим создание веб-приложений.

Глава 5

Создание веб-приложений

В данном разделе мы рассмотрим ключевые механизмы создания веб-приложений, реализованные в .NET Framework, их поддержку в Microsoft Visual Studio, а также основные изменения и дополнения, появившиеся в .NET Framework 3.5 и Microsoft Visual Studio 2008.

Для создания веб-приложений на основе .NET Framework средствами Microsoft Visual Studio используется технология, называющаяся Active Server Pages .NET (ASP.NET). ASP.NET представляет собой унифицированную модель создания Web-приложений, которая поддерживает разработку на всех языках программирования, совместимых с Common Language Runtime (CLR), включая такие языки, как Microsoft Visual Basic, C#, JScript .NET и J#. В состав ASP.NET входит модель описания страниц и компонентов, располагаемых на этих страницах, специальный компилятор, инфраструктура обеспечения безопасности, средства управления состоянием, механизмы конфигурации приложений, средства мониторинга производительности приложений, средства отладки, поддержка создания веб-сервисов на базе XML, поддержка хостинга и управления жизненным циклом приложения, а также профессиональные средства дизайна и разработки веб-приложений. Рассмотрим каждый из перечисленных компонентов подробнее.

Модель описания страниц и компонентов

Модель описания страниц и компонентов представляет собой набор программных средств, выполняемых на веб-сервере и динамически генерирующих и отрисовывающих веб-страницы (называемых ASP.NET веб-страницами). Такие страницы могут использоваться в любом браузере или мобильном устройстве — ASP.NET возвращает запросившему браузеру код страницы на языке HTML. ASP.NET веб-страницы полностью объектно-ориентированы. Это означает, что в рамках страницы вы можете обращаться к HTML-элементам через их свойства, методы и событийную модель. ASP.NET предоставляет в распоряжение разработчиков унифицированную модель реакции на клиентские события в коде, который выполняется на сервере. Программная модель автоматически поддерживает состояния для всей страницы и ее отдельных элементов во время цикла обработки страницы.

Функциональность пользовательского интерфейса реализована в ASP.NET на уровне набора интерфейсных компонентов, которые представляют собой как основные элементы, поддерживаемые на уровне языка HTML, так и дополнительные компоненты, расширяющие интерфейсные возможности, предоставляемые языком HTML.

В состав ASP.NET входят следующие типы компонентов.

- **Серверные HTML-компоненты.** Эти компоненты представляют собой программную реализацию стандартных HTML-элементов и позво-

ляют управлять различными аспектами их поведения из кода, выполняемого на сервере. Серверные HTMl-компоненты перечислены в следующей таблице.

HtmlAnchor	HtmlButton	HtmlForm	HtmlGenericControl
HtmlHead	HtmlImage	HtmlInputButton	HtmlInputCheckBox
HtmlInputFile	HtmlInputHidden	HtmlInputImage	HtmlInputPassword
HtmlInputRadio Button	HtmlInputReset	HtmlInputSubmit	HtmlInputText
HtmlLink	HtmlMeta	HtmlSelect	HtmlTable
HtmlTableCell	HtmlTableRow	HtmlTextArea	HtmlTitle

По умолчанию, HTML-элементы, включенные в состав файла ASP.NET представляют собой обычный текст и не могут использоваться из серверного кода (они доступны только из клиентского JavaScript — кода через DOM-модель). Для того чтобы такие элементы стали программно доступны, они должны представлять собой серверные HTML-компоненты — для этого в HTML-текст добавляется атрибут **runat="server"**.

HTML-элементы в галерее элементов

Помимо этого, для программного обращения к таким элементам можно использовать атрибут **id**, задающий идентификатор соответствующего элемента. Атрибуты также используются для задания свойств элементов и связи их с обработчиками событий. Например, HTML-элемент **<button>** соответствует серверному компоненту **HtmlButton**, который имеет следующие атрибуты:

```
<button
    CausesValidation="False|True"
    Disabled="Disabled"
    EnableViewState="False|True"
    Id="string"
    ValidationGroup="String"
    Visible="False|True"
    OnDataBinding="OnDataBinding event handler"
    OnDisposed="OnDisposed event handler"
    OnInit="OnInit event handler"
    OnLoad="OnLoad event handler"
    OnPreRender="OnPreRender event handler"
    OnServerClick="OnServerClick event handler"
    OnUnload="OnUnload event handler"
    runat="server"
    >
        <!- Текст, отображаемый на кнопке ->
</button>
```

Связь с обработчиком события, означающего нажатие клавиши, задается следующим образом:

```
<%@ Page Language="C#" AutoEventWireup="True" %>
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Transitional//EN"
    "http://www.w3.org/TR/xhtml1/DTD/xhtml1-transitional.dtd">
<html xmlns="http://www.w3.org/1999/xhtml" >
<head>
<title>HtmlButton Control</title>

   <script runat="server">
      void Button1_OnClick(object Source, EventArgs e)
      {
         Span1.InnerHtml="You clicked Button1";
      }
   </script>
</head>
<body>
<h3>HtmlButton </h3>
 <form id="Form1" runat="server">
  <p />
   <button id="Button1"
    onserverclick="Button1_OnClick"
    style="font: 8pt verdana;
    border-color:black;
    height:30;
    width:100"
```

```
   runat="server">
   Click me!
  </button>
 </form>
</body>
</html>
```

- **Серверные веб-компоненты.** Такие компоненты расширяют набор стандартных компонентов, поддерживаемых на уровне языка HTML — календарь, меню, средства отрисовки древовидных структур и т. п. и также предоставляют в распоряжение разработчиков программную модель, позволяющую управлять различными аспектами их поведения из кода, выполняемого на сервере.

  В состав ASP.NET входят следующие веб-компоненты:

AccessDataSource	AdRotator	BulletedList	Button
Calendar	ChangePassword	CheckBox	CheckBoxList
Content	ContentPlaceHolder	CreateUserWizard	DataGrid
DataList	DataPager	DetailsView	DropDownList
FileUpload	FormView	GridView	HiddenField
HyperLink	Image	ImageButton	ImageMap
Label	LinkButton	LinqDataSource	ListBox
ListView	Literal	Localize	Login
LoginName	LoginStatus	LoginView	Menu
MultiView	ObjectDataSource	Panel	PasswordRecovery
PlaceHolder	RadioButton	RadioButtonList	Repeater
ScriptManager	ScriptManagerProxy	SiteMapDataSource	SiteMapPath
SqlDataSource	Substitution	Table	TableCell
TableRow	TextBox	Timer	TreeView
UpdatePanel	UpdateProgress	View	Wizard
Xml	XmlDataSource		

Веб-компоненты описываются атрибутом **asp:**. Например, вот как описывается кнопка в виде серверного веб-компонента:

```
<asp:Button id="SubmitButton"
     Text="Submit"
     CommandName="Submit"
     OnCommand="CommandBtn_Click"
     runat="server"/>
```

и обработчик события для нее:

```
void CommandBtn_Click(Object sender, CommandEventArgs e)
{
    Message.Text = "You clicked the " + e.CommandName +
                   " - " + e.CommandArgument + " button.";
}
```

Серверные веб-компоненты в галерее компонентов

- **Компоненты проверки ввода.** Эти компоненты позволяют описывать логику проверки данных, вводимых пользователями в компоненте TextBox. Компоненты проверки ввода позволяют задавать поля, значения которых не могут быть пустыми, указывать шаблоны вводимой информации, диапазоны допустимых значений и т. п. В ASP.NET входят следующие компоненты проверки ввода:

Тип проверки	Компонент	Описание
Поле, которое не может быть пустым	**RequiredFieldValidator**	Позволяет убедиться в том, что пользователь обязательно введет значение в данное поле
Сравнение со значением	**CompareValidator**	Выполняет сравнение введенных данных с константой, значением другого поля или определенным типом данных
Проверка диапазона	**RangeValidator**	Проверяет, находится ли введенное пользователем значение в указанном диапазоне
Соответствие шаблону	**RegularExpression-Validator**	Проверяет, соответствует ли вводимая пользователем информация заданному шаблону в виде регулярного выражения. Используется для ввода адресов электронной почты, номеров телефонов, почтовых индексов и т. п.
Расширенная проверка	**CustomValidator**	Позволяет задавать собственную логику поверки вводимых данных

Компоненты проверки ввода используются совместно с компонентом **ValidationSummary**, который позволяет отображать сообщения об ошибках, выдаваемые данными компонентами.

Компоненты проверки ввода в галерее компонентов

- **Пользовательские компоненты** — это компоненты, которые разработчики создают для расширения набора стандартных интерфейсных элементов, входящих в состав ASP.NET.

Модель описания страниц и компонентов также позволяет управлять внешним видом интерфейсных элементов — для этого используются темы и «скины». Возможно использование тем и «скинов» как на уровне страниц, так и на уровне отдельных компонентов.

Помимо тем в ASP.NET поддерживаются т. н. «мастер-страницы», позволяющие задавать единое расположение элементов для всех страниц (или группы страниц), входящих в состав веб-приложения.

Специальный компилятор

Так как код ASP.NET-страниц является компилируемым, это позволяет обеспечить, помимо всего прочего, поддержку строгой типизации, оптимизацию производительности и раннее связывание. После того как код откомпилирован, ядро исполнения .NET — Common Language Runtime — преобразует его в машинный код для обеспечения максимальной производительности. В состав ASP.NET входит компилятор для обработки компонентов веб-приложения, включая страницы и визуальные компоненты — они преобразуются в сборки, которые выполняются под управлением среды ASP.NET.

Инфраструктура обеспечения безопасности

В дополнение к механизмам безопасности, реализованным на уровне .NET, ASP.NET включает расширенную инфраструктуру безопасности, обеспечивающую сервисы аутентификации и авторизации пользователей, а также решения ряда связанных с обеспечением безопасности задач. Поддерживается возможность использования аутентификации Windows, поддерживаемой на уровне Internet Information Services, либо собственных механизмов аутентификации на основе форм ASP.NET и ASP.NET Membership с использованием собственной базы данных пользователей. Помимо этого, имеется возможность управления авторизацией веб-приложений, используя группы Windows или роли на уровне ASP.NET с помощью собственной базы данных. Поддерживается возможность удаления, добавления или замены описанных выше механизмов в зависимости от требований к конкретному веб-приложению.

Отметим, что ASP.NET всегда выполняется под определенной учетной записью Windows — таким образом вы можете дополнительно защитить приложение, используя такие возможности Windows, как NTFS Access Control Lists (ACLs), разрешения на уровне баз данных и т. п.

Средства управления состоянием

В ASP.NET входят встроенные средства управления состоянием, позволяющие сохранять информацию между запросами к странице. Наиболее часто такие сервисы требуются для хранения данных о пользователях или о содержимом корзины покупателя в сценариях электронной коммерции. Поддерживается возможность сохранять и управлять информацией на уровне приложения, на уровне сессии, на уровне страницы, на уровне пользователя и т. п. Разработчики могут расширять эти механизмы для предоставления дополнительной функциональности по хранению данных. Такая

информация может быть полностью независимой от компонентов, располагаемых на странице. В ASP.NET поддерживаются распределенные средства сохранения информации — например, для нескольких экземпляров приложения, выполняющихся на одном сервере или на нескольких серверах.

Механизмы конфигурации приложений

Приложения, создаваемые средствами ASP.NET, используют систему конфигурации, которая позволяет задавать конфигурационные настройки для веб-сервера, веб-сайта или определенного приложения. Эти настройки можно выполнять в процессе развертывания ASP.NET-приложений — это минимально влияет на работу как других веб-приложений, так и самого сервера. Настройки ASP.NET хранятся в XML-файлах, и благодаря этому вносить изменения в конфигурацию можно с помощью различных редакторов. При необходимости, конфигурационная схема может быть расширена — ASP.NET не устанавливает никаких ограничений на расширения содержимого конфигурационных файлов.

Средства мониторинга производительности приложений

В ASP.NET входят средства, позволяющие выполнять мониторинг производительности и других ключевых характеристик приложений. Средства слежения за «здоровьем» приложений (health monitoring) позволяют получить отчеты о ключевых событиях, связанных с жизненным циклом приложения и различных состояниях, в которых оно может находиться, включая ошибки. Эти события объединяют ключевые характеристики мониторинга и диагностики и обеспечивают максимальную гибкость при определении, какие данные и когда должны заноситься с протокол. В ASP.NET поддерживается две группы счетчиков производительности, которые доступны из приложений — группа системных счетчиков, собирающих данные о работе всей подсистемы, и группа прикладных счетчиков, отвечающих за сбор данных о конкретном приложении.

Средства отладки

В ASP.NET полностью используется инфраструктура времени исполнения, предоставляемая .NET CRL для обеспечения функций отладки. Поддерживается возможность отладки как объектов, написанных на неуправляемом коде, так и объектов, созданных средствами Microsoft .NET, используя любые языки программирования, поддерживаемые на уровне CLR, а также скриптовые языки. Помимо этого, модель описания страниц и компонентов поддерживает режим трассировки, позволяющий разработчикам использовать механизмы управления приложениями (instrumentation) непосредственно в веб-страницах, создаваемых средствами ASP.NET.

Поддержка создания веб-сервисов на базе XML

В ASP.NET полностью поддерживаются веб-сервисы на базе XML. Такой веб-сервис представляет собой компонент, содержащий бизнес-функциональность и позволяющий приложениям обмениваться информацией через межсетевые экраны, используя такие стандарты, как протокол HTTP и XML Messaging. Веб-сервисы на связаны с какой-то конкретной компонентной технологией или соглашением о вызове объектов. В результате, приложения, написанные на любом языке программирования, использующие любую компонентную модель и любую программную платформу, могут обращаться к веб-сервисам.

Поддержка хостинга и управления жизненным циклом приложения

В состав ASP.NET входит среда хостинга с возможностью расширения, которая позволяет управлять жизненным циклом приложений — от первого доступа к определенному ресурсу (например, веб-странице) приложения до завершения работы приложения (остановке веб-сервера). ASP.NET использует веб-сервер (Internet Information services, IIS) в качестве сервиса для хостинга среды выполнения и самих приложений, но при этом самостоятельно обеспечивает расширенные возможности хостинга. Например архитектура ASP.NET позволяет обрабатывать события на уровне приложений и создавать собственные обработчики HTTP-запросов и HTTP-модули.

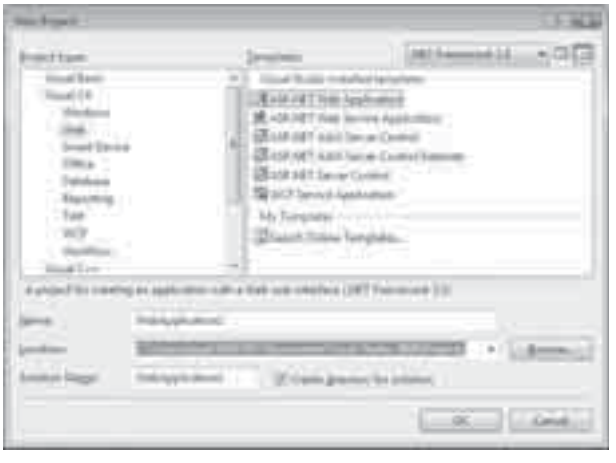
Профессиональные средства дизайна и разработки веб-приложений

Все описанные выше ключевые возможности технологии ASP.NET полностью поддерживаются в средствах визуального дизайна и разработки веб-приложений, входящих в состав Visual Studio.

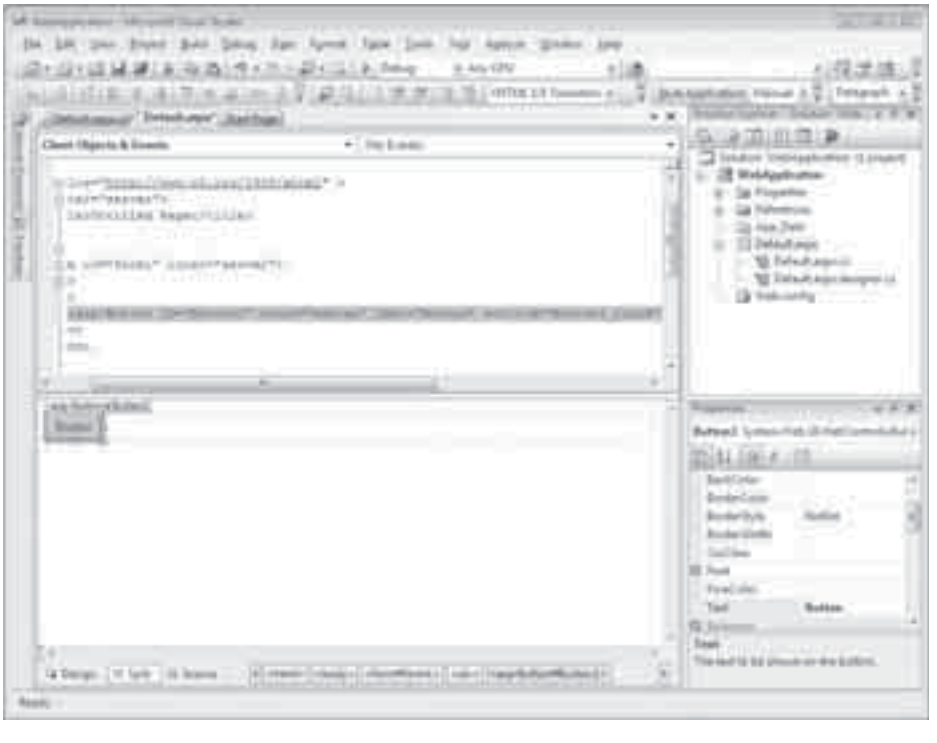
Создание ASP.NET-приложений в Microsoft Visual Studio строится на использовании шаблонов, выбираемых из ветви веб для поддерживаемых языков программирования — Visual Basic .NET и Visual C#. Доступные шаблоны показаны на следующей иллюстрации.

После выбора шаблона (в большинстве случае для веб-приложений используется шаблон ASP.NET Web Application), мы попадаем в дизайнер, который позволяет в среду визуального создания веб-приложений. Данная среда позволяет работать с веб-страницами на уровне исходного кода (вкладка Source), на уровне дизайна (вкладка Design) или в совмещенном режиме (вкладка Split) — последний является новинкой в Visual Studio 2008. Галерея компонентов содержит различные компоненты — HTML и веб и может использоваться для их перетаскивания на разрабатываемую страницу. Панель управления свойствами позволяет настраивать различные атрибуты самой страницы и размещенных на ней компонентов, а двой-

ной щелчок «мышью» по компоненту приводит к появлению редактора для написания кода для обработки соответствующего события.



Шаблоны для создания веб-приложений в Visual Studio



Среда разработки веб-приложений

Поддерживается автоматическая проверка кода на соответствие различным стандартам, включая Internet Explorer 6.0, Internet Explorer 3.02/Net-

scape Navigator 3.0, Netscape Navigator 4.0, HTML 4.01, XHTML 1.0 и XHTML 1.1. По умолчанию используется режим совместимости с XHTML 1.0 Transitional, поддерживаемым в Netscape 7, Opera 7 и Internet Explorer 6.

После того как мы кратко познакомились с ключевыми характеристиками технологии ASP.NET, используемой для создания веб-приложений на основе .NET Framework средствами Microsoft Visual Studio, давайте обсудим основные изменения и дополнения, появившиеся в .NET Framework 3.5 и Microsoft Visual Studio 2008. К ним можно отнести:

- Интегрированную поддержку ASP.NET AJAX 1.0;
  - ☐ на уровне .NET Framework 3.5.
  - ☐ на уровне шаблона проекта ASP.NET AJAX.
- Новые шаблоны проектов Web Application в Visual Studio;
- JavaScript IntelliSense и возможность отладки кода;
- Расширенную поддержку HTML/CSS в дизайнерах:
  - ☐ Split View (исходный текст и дизайнер).
  - ☐ Просмотр вложенных Master Pages в дизайнере.
  - ☐ Свойства CSS.
  - ☐ Окно «Применить стили».
  - ☐ Панель применения стилей.
  - ☐ Улучшения в расположении и визуализации CSS/HTML в дизайнере.
- Новые компоненты для данных: **LinqDataSource**, **ListView**, **DataPager**.

Рассмотрим эти изменения и дополнения более подробно. Начнем с поддержки технологии AJAX.

Интегрированная поддержка ASP.NET AJAX 1.0

Технология AJAX — это технология создания веб-приложений, обладающих повышенной интерактивностью, скоростью и производительностью. AJAX базируется на комбинации таких технологий, как XHTML (или HTML) и CSS, клиентский скриптинг (JavaScript, Jscript) и объект **XMLHttpRequest**. Это позволяет клиенту (браузеру) обмениваться с сервером небольшими объемами данных, необходимыми для обновления определенных частей страницы, а не всей страницы, как при использовании традиционных технологий создания веб-приложений. Использование AJAX позволяет создавать более «богатые» интерфейсы для веб-приложений, что делает пользователей более продуктивными и обеспечивает реализацию сценариев, ранее недоступных для тонких клиентов.

Технологический набор, обеспечивающий функционирование технологии AJAX, был разработан компанией Microsoft и в настоящее время поддерживается всеми ведущими производителями. Первым шагом на пути

реализации данной технологии стало появление HTML-элемента **IFRAME** в Microsoft Internet Explorer 3.0, выпущенном в 1996 году. Использование данного элемента позволяло обновлять часть веб-страницы без необходимости в полной перезагрузке всей страницы. В 1998 Microsoft реализовала концепцию удаленного скриптинга (remote scripting), которая служила альтернативой элементу **IFRAME**, а в рамках браузера Internet Explorer 5.0 появился объект **XMLHttpRequest**, который очень эффективно использовался для обеспечения функционирования тонкого клиента почтовой программы Outlook — Outlook Web Access.

В Microsoft Visual Studio 2008 входят шаблоны для создания серверных компонентов на базе технологии AJAX (ASP.NET AJAX Server Control) и расширений для таких компонентов (ASP.NET AJAX Server Control Extender).

Поддержка ASP.NET AJAX реализована на основе следующих ключевых компонентов:

- Microsoft AJAX Library;
  - ☐ Клиентская библиотека на JavaScript;
  - ☐ Работает на любом браузере и поддерживается любым веб-сервером (включая PHP, ColdFusion и т. п.);
- ASP.NET 2.0 AJAX Extensions;
  - ☐ Серверные расширения для интеграции с ASP.NET 2.0;
- ASP.NET AJAX Control Toolkit;
  - ☐ Бесплатный набор компонентов с полным исходным текстом, доступный для загрузки по адресу http://ajax.asp.net.

Ключевыми компонентами, обеспечивающими поддержку технологии AJAX в ASP.NET-приложениях, являются **ScriptManager** (и **ScriptManagerProxy**) и **UpdatePanel**. Компонент **UpdatePanel** используется для задания регионов страницы, которые должны быть обновляемыми без обновления содержимого всей страницы.

```
<asp:UpdatePanel id="updatepanel1" runat="server">
    <ContentTemplate>
        <!-Содержимое будет автоматически обновляться! ->
        <asp:Calendar id="calndr1" runat="server"/>
    <ContentTemplate>
</asp:UpdatePanel>
```

Помимо этого также можно расширять функциональность существующих компонентов, добавляя к ним поддержку технологии AJAX.

Новые шаблоны проектов Web Application

Проект Web Application, появившийся в Microsoft Visual Studio 2008, является альтернативой проекту Web Site в предыдущих версиях Visual Studio.

Проектная модель веб-приложения полностью поддерживает все возможности Visual Studio 2008 и ASP.NET 2.0.

По умолчанию, проектная модель Web Site использует структуру каталогов, которые отражают содержимое проекта. В этой модели нет файлов проекта — все файлы в каталоге являются частью проекта. В проектной модели Web Application только файлы, непосредственно включенные в проект, являются частью этого проекта и только они отображаются в Solution Explorer и компилируются в процессе сборки проекта. Использование проектного файла в модели Web Application позволяет более просто реализовать ряд сценариев. Например, у разработчиков появляется возможность разделить ASP.NET-приложение на несколько проектов в рамках Visual Studio и ссылаться на разные проектные файлы и при необходимости легко исключать файлы из проекта. Проектную модель Web Application следует использовать в следующих случаях:

- Миграция больших по объему приложений, созданных средствами Visual Studio .NET 2003, в Visual Studio 2008;
- Управление именами сборок, создаваемых при компиляции проекта;
- Использование отдельных классов для ссылки на классы страниц и пользовательские компоненты;
- Создание веб-приложений, используя несколько веб-проектов;
- Добавление шагов, выполняемых перед и после основной компиляции проекта.

Поддержка JavaScript

Поддержка JavaScript важна для обеспечения создания приложений с использованием таких технологий, как AJAX и Microsoft Silverlight. В Microsoft Visual Studio 2008 реализована полная поддержка написания кода на JavaScript, включая:

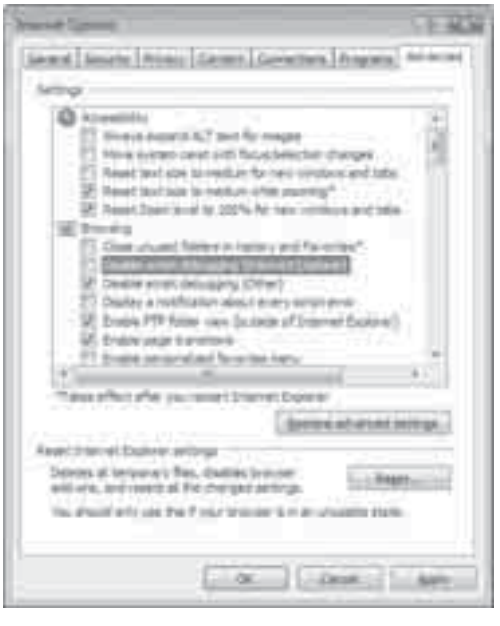
- поддержку на уровне технологии Intellisense в редакторе кода;
- полноценную поддержку в средствах отладки приложений.

JavaScript Intellisense

Поддержка JavaScript на уровне технологии Intellisense позволяет автоматически определять тип переменных (Type inference), обеспечивать корректную работу с внешними библиотеками на JavaScript (файлы с рас-

ширением .js), на которые ссылается текущий код, а также использовать комментарии, добавляемые в описаниях функций для определения типов.

Поддержка отладки JavaScript-кода позволяет устанавливать точки останова непосредственно в файлах с расширениями .html, .js, .aspx и .master — для этого не требуется сначала генерировать клиентский код. Помимо этого поддерживается присоединение к Internet Explorer для отладки кода на JavaScript в рамках любой страницы (HTML, PHP, JSP и т. д.), но для этого требуется корректная конфигурация Internet Explorer — на вкладке Advanced панели Internet Options (команда Tools | Internet Options) следует отключить опцию Disable Script Debugging, которая по умолчанию включена.

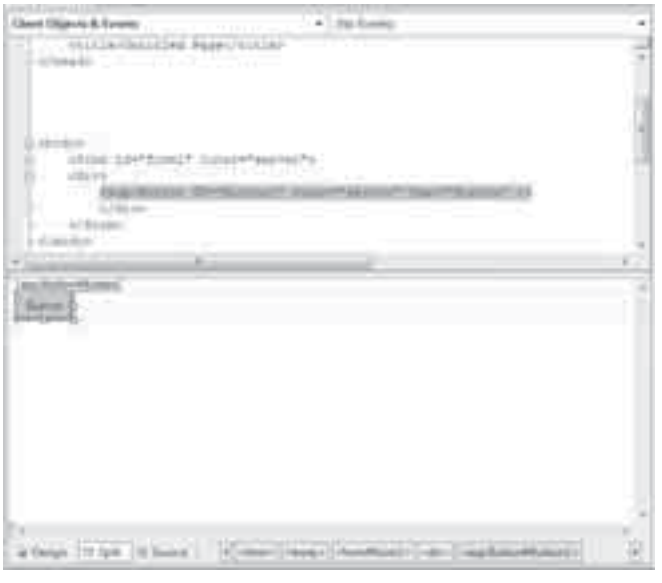


Панель настроек Internet Explorer

В целом, поддержка отладки JavaScript в Visual Studio 2008 обеспечивает возможность использования следующих функций — установка точек остановки по условию (Conditional Breakpoints), просмотр значений локальных переменных (Locals Window), а также использование окон Imme-diate Window и Watch Windows.

Расширенная поддержка HTML/CSS в дизайнерах

Дизайнер HTML-кода в Visual Studio 2008 пополнился возможностью отображения как кода, так и его визуального представления с двунаправленной синхронизацией вносимых изменений — этот режим называется Split-View. Переключение между тремя способами отображения — исходный текст, дизайн и Split-View — стало существенно быстрее.



Режим Split View в HTML-редакторе

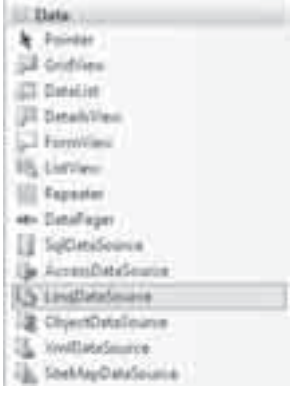
Редактор стилей в Visual Studio 2008

Средства дизайна CSS теперь единые для Visual Studio и семейства продуктов Expression. В новой версии появилось окно «Manage Styles», которое позволяет задавать CSS-правила и таблицы стилей для редактируемой страницы. Поддерживается возможность перетаскивания стилей — вместо встроенных стилей их можно сделать внешними файлами. Можно создавать новые стили или редактировать существующие — двойной щел-

чок мышью по стилю переводит его в режим редактирования. Окно свойств CSS позволяет задавать значения любых свойств стиля и поддерживает режим просмотра, позволяющий определить примененные стили и уровень их вложенности и наследования.

Новые компоненты для данных

В .NET Framework 3.5 появился ряд новых ASP.NET-компонентов для работы с данными — мы кратко рассмотрим эти компоненты ниже.



Компоненты для работы с данными в галерее компонентов

Компонент LinqDataSource

Компонент **LinqDataSource** предназначен для доступа к технологии Language Integrated Query (LINQ) через стандартную архитектуру источников данных ASP.NET. Этот компонент используется на веб-страницах, где требуется получение или модификация данных с использованием программной модели LINQ. Компонент поддерживает автоматическое создание команд для работы с данными и позволяет существенно сократить код, требуемый для реализации операций с данными по сравнению с использованием таких компонентов, как **SqlDataSource** или **Object-DataSource**. Помимо этого, **LinqDataSource** позволяет разработчикам использовать одну программную модель для доступа к различным типам данных.

Разработчики могут использовать декларативное описание для создания компонента **LinqDataSource**, связанного либо с базой данных, либо со структурой в памяти (например коллекцией). В этом описании указываются критерии для отображения, фильтрации, сортировки и группировки данных, а также разбиения результатов на страницы. Когда источником данных является таблица в базе данных, можно сконфигурировать компонент таким образом, что он будет обновлять, вставлять и удалять данные — разработчикам не требуется писать SQL-команды для выполнения

этих задач. Компонент **LinqDataSource** поддерживает событийную модель, которая позволяет создавать обработчики событий, возникающих при работе с источниками данных, а также набор свойств, позволяющих управлять внешним видом и поведением компонента.

Компонент ListView

Компонент **ListView** пришел на смену компонентам **DataList** и **Repeater**. Этот компонент позволяет, например, генерировать выпадающие списки, таблицы и не отсортированные списки и предназначен для совместной работы с компонентами типа **LinqDataSource** и **DataPager**. Компонент может легко настраиваться, например используя CSS-стили, и связываться с практически любыми элементами (например, **<select>**).

Компонент DataPager

Компонент **DataPager** используется для страничного доступа к данным, отображаемым в компонентах, поддерживающих интерфейс **IPageableItemContainer** — например компонент ListView. Этот компонент поддерживает встроенные средства навигации по страницам, которые указываются объектом **NumericPagerField**, который позволяет пользователям выбирать страницы по их номеру. Помимо этого можно использовать объект **NextPreviousPagerField**, позволяющим пользователям перемещаться по страницам или переходить на первую или последнюю страницу. Также можно создавать собственные механизмы навигации — для этого служит объект **TemplatePagerField**.

Утилита ASP.NET Merge

Утилита ASP.NET Merge (**Aspnet_merge.exe**) позволяет комбинировать и управлять сборками, создаваемыми пред-компилятором ASP.NET (**Aspnet_compiler.exe**). Используя данную утилиту, можно создавать единые сборки для всего сайта — объединять все сборки в одну, сборки для каждого каталога веб-сайта или только для файлов, отвечающих за визуальные элементы сайта — страницы и компоненты.

В следующем разделе мы рассмотрим, как используя Microsoft Visual Studio создавать приложения на основе платформы Microsoft Office.

Глава 6

Создание приложений на платформе Microsoft Office

Новая версия Visual Studio Tools for Office — версия 3.0 (также называется VSTO 2008) позволяет создавать решения на основе ключевых продуктов, входящих в состав Microsoft Office 2003 и Microsoft Office 2007. Поддерживаются языки программирования C# и Visual Basic .NET. В VSTO 3.0 входит ряд новых и расширенных функций, которые мы кратко рассмотрим ниже. Visual Studio Tools for Office входит в состав следующих изданий Visual Studio: Visual Studio 2008 Professional, Team Edition for Architects, Team Edition for Database Professionals, Team Edition for Developers и Team Edition for Testers.

Поддержка создания решений на основе Microsoft Office впервые появилась в 2003 году — первая версия Visual Studio Tools for Office поддерживала только Excel 2003 и Word 2003 и позволяла реализовывать только расширения на уровне документов (document-level code behind) используя управляемый код.

В VSTO 2005 появилась поддержка InfoPath, а также возможность создания модулей расширения для Outlook. Помимо этого в VSTO 2005 была реализована поддержка создания панелей задач (ActionsPane) для ряда продуктов, входящих в семейство Microsoft Office 2003.

В VSTO 2005 Second Edition была добавлена поддержка Office 2007, включая возможность создания расширений для PowerPoint и Visio. Этот продукт был доступен для бесплатной загрузки для легальных пользователей Microsoft Visual Studio.

Как мы отметили выше, VSTO 3.0 входит в состав Visual Studio 2008, начиная с версии Professional, и обеспечивает поддержку разработки модулей расширения на уровне документов и приложений как для Office 2003, так и для Office 2007. Обеспечена возможность создания Workflow для Microsoft SharePoint и развертывание по технологии ClickOnce.

Основные требования

Для того чтобы вы смогли создавать решения с использованием Visual Studio Tools for Office, на компьютере, используемом для разработки, должны быть установлены следующие компоненты.

- Поддерживаемые версии Microsoft Office. В зависимости от типа проекта могут потребоваться различные версии продукта, но все они должны быть установлены локально, на компьютере разработчика.

Все типы проектов для Microsoft Office 2007	Проекты на уровне документов для Microsoft Office 2003	Проекты на уровне моделей расширений для Microsoft Office 2003
Все издания Microsoft Office 2007 (Standard, Professional, и т. п.), а также отдельные продукты: Excel 2007, InfoPath 2007, Outlook 2007, PowerPoint 2007, Project 2007, Visio 2007, Word 2007 и SharePoint Server 2007	Следующие издания: Microsoft Office Professional Edition 2003, Microsoft Office Professional Edition 2003 Trial, Microsoft Office Professional Enterprise Edition 2003, Microsoft Office System Evaluation 2003 Enterprise Edition или отдельные продукты: Excel 2003 и Word 2003	Следующие издания: Microsoft Office Professional Edition 2003, Microsoft Office Professional Enterprise Edition 2003, Microsoft Office Standard Edition 2003 или отдельные продукты: Excel 2003, Outlook 2003, PowerPoint 2003, Project 2003, Visio 2003 и Word 2003
	Рекомендуется установка Microsoft Office 2003, Service Pack 1 (SP1) или более поздней версии пакета обновлений	

- Для всех установленных продуктов семейства Microsoft Office потребуется установка PIA (Primary Interop Assemblies), которые обеспечивают доступ к программной модели продуктов семейства Office из кода, написанного на управляемых языках программирования и выполняемого под управлением CLR (Common Language Runtime).
- Для того чтобы решения на базе Microsoft Office, создаваемые средствами Visual Studio Tools for Office, можно было отлаживать и выполнять, PIA должны быть установлены в глобальном кэше сборок (global assembly cache, GAC).
- Если перед установкой Microsoft Office на компьютере была установлена библиотека .NET Framework версии 2.0 или более поздней, вы можете установить PIA в глобальный кэш сборок, выбрав в Office опцию «Complete Installation». В противном случае вам потребуется отдельно установить PIA.
- Отметим, что входящие в состав Visual Studio Tools for Office PIA устанавливаются локально и используются только для сборки разрабатываемых проектов.
- Как часть установки Microsoft Office необходимо установить Visual Basic for Applications (VBA).

Для установки Visual Studio Tools for Office на компьютере, где планируется разработка для Microsoft Office System 2007, необходимо выполнить следующие шаги.

- Установить .NET Framework 3.5 или более поздней версии — это необходимо сделать перед установкой Primary Interop Assemblies, которые помещаются в глобальный кэш сборок.
- Установить требуемую версию Microsoft Office, включая Primary Interop Assemblies. Отметим, что Visual Studio Tools for Office не поддерживают разработку для Microsoft Office 2003 и 2007 (или различных версий отдельных продуктов, например Word 2003 и Word 2007) на одном компьютере — для создания приложений для Microsoft Office 2003 и 2007 потребуется отдельный компьютер для каждой версии продукта.
- Установить версию Visual Studio, в состав которой входит Visual Studio Tools for Office. При необходимости получения локализованных сообщений, также потребуется установить Language Package, который доступен на сайте Microsoft.

В случае использования Visual Studio Tools for Office для создания приложений для Microsoft Office 2003, вам потребуется установить .NET Framework 2.0, а также пакет обновлений для Office — Office Service Pack 2.

Если вы планируете создавать и расширять SharePoint Workflow, необходимо установить Visual Studio Tools for Office на компьютере, операционная система которого поддерживает Microsoft Office SharePoint Server 2007 — Windows Server 2003 или Windows Server 2008. В результате будут установлены шаблоны проектов, которые позволят вам создавать расширения для SharePoint Workflow.

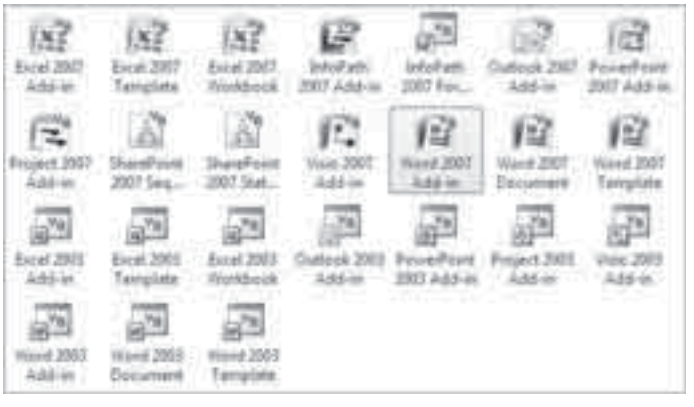
После того как все необходимые компоненты, продукты и средства разработки установлены, можно приступать к разработке решений на базе Microsoft Office. К основным возможностям, предоставляемым Visual Studio Tools for Office, относятся создание дополнительных модулей, работа с документами Word и Excel, расширения интерфейсного элемента «лента», создание панелей задач, использование Word Content Controls, разработка расширения на основе Outlook Forms и создание расширений для SharePoint Workflow.

Создание дополнительных модулей

Дополнительные модули на уровне приложений (application-level add-ins) позволяют расширять функциональность ключевых приложений, входящих в состав Microsoft Office. Дополнительные модули доступны в приложении в независимости от того, какой документ в нем открыт — это отличает механизмы расширения на основе дополнительных модулей от механизмов расширений на основе шаблонов документов, которые мы рассмотрим ниже.

В Visual Studio Tools for Office входят средства для создания дополнительных модулей на основе новых шаблонов проектов для большинства прило-

жений, входящих в семейства Microsoft Office 2003 и Microsoft Office 2007 — Excel 2003 и 2007, InfoPath 2007, Outlook 2003 и 2007, PowerPoint 2003 и 2007, Project 2003 и 2007, Visio 2003 и 2007, Word 2003 и 2007. Использование новой программной модели существенно упрощает создание дополнительных модулей, так как не требует использования COM-технологий.



Шаблоны, входящие в состав Visual Studio Tools for Office

Среди новинок для модулей расширений отметим возможность обеспечения доступа к объектам внутри ваших моделей расширений из других компонентов Microsoft Office — например других модулей расширений или кода на Visual Basic for Applications, включенного в состав документа. Таким образом, у разработчиков появляется возможность создания на уровне модулей расширения разделяемых сервисов, которые могут использоваться другими компонентами и сервисами.

Работа с документами Word и Excel

Создание расширений на уровне документов представляет собой способ добавления расширенной функциональности к тому или иному документу или электронной таблице. Так как расширения реализованы на уровне документов, они будут доступны только в тех документах или электронных таблицах, с которыми они ассоциированы.

В состав Visual Studio Tools for Office входит набор возможностей, упрощающих создание расширений на уровне документов для Word 2007 и Excel 2007. К таким возможностям относятся:

- Создание расширений для документов и электронных таблиц в формате Office Open XML для Word 2007 и Excel 2007 или бинарных форматах, поддерживаемых в Microsoft Office 2003 или более ранних версий.
- Дизайн документов и шаблонов в Visual Studio и написание кода в среде разработчика.

- Возможность добавления компонентов Windows Forms в документы и шаблоны.
- Добавление компонентов-расширений (host-controls) для встроенных объектов Word и Excel, которые расширяют объектную модель и обеспечивают возможность связи с данными.

Настройка и расширение «ленты»

Интерфейсный элемент «лента» (Ribbon) впервые появился в Microsoft Office 2007 и представляет собой объединение функциональных возможностей меню, списков, галерей и ряда других элементов. Лента поддерживает контекстное переключение, группировку задач и ряд других возможностей. В Microsoft Office 2007 поддерживается программная настройка и создание расширений интерфейсного элемента «лента» для следующих продуктов:

- Microsoft Office Excel 2007;
- Microsoft Office Outlook 2007;
- Microsoft Office PowerPoint 2007;
- Microsoft Office Word 2007.

Существует два способа настройки элемента «лента» — с помощью дизайнера (Ribbon Designer), включенного в состав Visual Studio Tools for Office 2008 и через описание «ленты» на языке Ribbon XML.

Настройка элемента «лента» с помощью дизайнера поддерживает следующую функциональность:

- Возможность добавления настраиваемой «ленты» к проекту Office, используя специальный шаблон проекта — **Ribbon (Visual Designer)**.
- Визуальное создание новых вкладок с помощью дизайнера.
  - ☐ Перетаскивание компонентов на «ленту».
  - ☐ Настройка расположения компонентов и их внешнего вида.
  - ☐ Добавление обработчиков событий двойным щелчком по компоненту «ленты».
- Установка свойств «ленты» и ее компонентов в окне свойств (**Properties**).
- Добавление кода обработчиков событий на языках Visual C# или Visual Basic с полной поддержкой проверки типов и технологии IntelliSense.

Дизайнер также может использоваться для добавления компонентов к меню, которые открываются при щелчке на кнопку Office (Microsoft Office Button).

Настройка «ленты», используя язык Ribbon XML, позволяет более детально контролировать «ленту» и ее элементы и выполнять ряд задач, не под-

держиваемых в дизайнере Ribbon Designer. В Visual Studio Tools for Office 2008 работа с «лентой» на уровне Ribbon XML поддерживается на уровне специального шаблона — **Ribbon (XML)**, помимо этого имеется возможность экспорта любой «ленты», созданной средствами дизайнера, в шаблон для редактирования «ленты» на уровне языка XML.

В Visual Studio Tools for Office 2008 реализована типизованная объектная модель, которая может использоваться для управления компонентами «ленты» в режиме выполнения приложения. Например, можно динамически добавлять элементы меню или управлять отображением отдельных компонентов в зависимости от контекста, в котором в данный момент находится приложение. Объектная модель состоит из трех ключевых элементов — класса **Ribbon**, событийной модели и классов, отвечающих за отдельные компоненты «ленты».

Класс **Ribbon** наследует от класса **OfficeRibbon** и представляет собой класс, код которого разделен в файлах кода, создаваемого разработчиком и файла, генерируемого дизайнером «ленты». Событийная модель поддерживает три события — **Load** — возникает при начальной загрузке расширений «ленты», **LoadImage** — используется для кэширования графических изображений, используемых в «ленте», и **Close**, которое возникает при завершении работы экземпляра «ленты».

В пространстве имен **Microsoft.Office.Tools.Ribbon** определены следующие классы, отвечающие за отдельные компоненты «ленты»:

Компонент	Имя класса
Box	RibbonBox
Button	RibbonButton
ButtonGroup	RibbonButtonGroup
CheckBox	RibbonCheckBox
ComboBox	RibbonComboBox
DropDown	RibbonDropDown
Edit	RibbonEdit
Gallery	RibbonGallery
Group	RibbonGroup
Label	RibbonLabel
Menu	RibbonMenu
Separator	RibbonSeparator
SplitButton	RibbonSplitButton
Tab	RibbonTab
ToggleButton	RibbonToggleButton

Помимо свойств, каждый компонент «ленты» поддерживает ряд событий — они перечислены в следующей таблице.

Событие	Описание
Click	Возникает при щелчке мышью по компоненту
TextChanged	Возникает при изменении текста в поле редактирования или поле выбора
ItemsLoading	Возникает при обращении Office к коллекции Items данного компонента
ButtonClick	Возникает при щелчке мышью по кнопке в компонентах RibbonDropDown или RibbonGallery
SelectionChanged	Возникает при изменениях в компонентах RibbonDropDown или RibbonGallery
DialogLauncherClick	Возникает при щелчке мышью по иконке в нижней правой области группы

Все рассмотренные выше классы, события и другие элементы поддержки «ленты» в Visual Studio Tools for Office 2008 реализованы в пространстве имен **Microsoft.Office.Tools.Ribbon**. В этом пространстве имен содержатся следующие классы:

Класс	Описание
OfficeRibbon	Служит базовым классом для всех настроек «ленты»
RibbonBox	Используется для сборки и выравнивания компонентов на «ленте»
RibbonButton	Представляет собой кнопку, располагаемую на «ленте»
RibbonButtonCollection	Представляет собой коллекцию объектов типа RibbonButton
RibbonButtonGroup	Представляет собой группу кнопок, располагаемых на «ленте»
RibbonCheckBox	Представляет собой кнопку «включить/выключить», располагаемую на ленте
RibbonComboBox	Представляет собой комбинированный список, располагаемый на ленте
RibbonComponent	Служит базовым классом для всех компонентов «ленты»
RibbonComponent-Collection(T)	Коллекция компонентов «ленты»
RibbonControl	Базовый класс для компонентов, используемых при настройке «ленты»
RibbonControlEventArgs	Предоставляет данные для различных событий, возникающих при использовании «ленты» и ее отдельных компонентов
RibbonControlId	Включает строку идентификации, используемую Microsoft Office для обращения к компоненту

Класс	Описание
RibbonDialogLauncher	Представляет собой иконку, которая может использоваться в группе для открытия диалоговой панели
RibbonDropDown	Представляет собой список элементов, из которых пользователь может выбрать один или другой, а также список кнопок, которые пользователь может щелкнуть
RibbonDropDownItem	Представляет собой элемент выпадающего списка в коллекции RibbonDropDownItemCollection.
RibbonDropDownItem-Collection	Представляет собой коллекцию объектов типа RibbonDropDownItem
RibbonEditBox	Представляет собой поле редактирования, располагаемое на «ленте»
RibbonEventAttribute	Используется событийной моделью «ленты»
RibbonGallery	Представляет собой компонент, который отображает меню из элементов типа RibbonDropDownItem и компонентов типа RibbonButton
RibbonGroup	Группа компонентов на вкладке «ленты»
RibbonLabel	Текстовая подпись для компонентов типа RibbonGroup или RibbonBox
RibbonLoadImage-EventArgs	Предоставляет данные для события LoadImage
RibbonManager	Управляет «лентой» в проекте Visual Studio Tools for Office
RibbonMenu	Представляет собой меню или вкладку или меню Microsoft Office Menu
RibbonOfficeMenu	Обеспечивает доступ к компонентам, которые могут быть добавлены в меню Microsoft Office Menu
RibbonPosition	Представляет собой местоположение компонента на «ленте» относительно встроенных компонентов, местоположение вкладки относительно встроенных вкладок или местоположение группы относительно встроенных групп
RibbonReadOnly-Collection	Обеспечивает доступ к экземпляру класса OfficeRibbon
RibbonSeparator	Представляет собой компонент-разделитель для группы или меню
RibbonSplitButton	Представляет собой кнопку или переключатель и выпадающее меню
RibbonTab	Объединяет одну или более групп компонентов
RibbonToggleButton	Представляет собой переключатель, располагаемый на «ленте»
RibbonUIEventArgs	Предоставляет данные для событий, которые возникают в процессе использования «ленты» и ее компонентов

Создание панелей задач

Создание панелей задач — это еще один способ настройки приложений Microsoft Office, используя Visual Studio Tools for Office 2008. Панели задач представляют собой интерфейсный элемент, который обычно располагается справа от основного окна приложения Microsoft Office. Панели задач позволяют расширять функциональность офисных приложений и часто используются как механизм обеспечения интеграции между офисными приложениями и приложениями, автоматизирующими какие-либо бизнес-задачи.

Панели задач обычно создаются либо на уровне документов — в этом случае они называются Actions Panes, или на уровне приложений — в этом случае их называют Custom Task Panes. В следующей таблице приведены основные рекомендации по использованию Actions Panes и Custom Task Panes.

	Actions Pane	**Custom Task Pane**
Поддержка	VSTO 2005 / Office 2003	VSTO 2005 SE / Office 2007
Когда использовать	В тех случаях, когда интерфейс содержит активности, специфичные для конкретного документа или шаблона	В тех случаях, когда интерфейс содержит активности, добавляемые модулем расширения на уровне приложения, в независимости от того, с каким документом работает пользователь
Доступность	Документ/шаблон Word Документ/шаблон Excel (поддерживаются версии 2003 и 2007)	Модуль расширения для Word 2007 Модуль расширения для Excel 2007 Модуль расширения для PowerPoint 2007 Модуль расширения для Outlook 2007 Модуль расширения для InfoPath 2007 Модуль расширения для Access 2007

Панель задач типа Actions Pane создается на основе класса **ActionsPane**, который реализован в пространстве имен **Microsoft.Office.Tools**. Данный класс поддерживает обширную объектную модель с большим числом методов, свойств и событий, которые могут использоваться разработчиками для управления различными аспектами создания и отображения панелей задач, а также обеспечения всего жизненного цикла панели — от ее инициализации до завершения работы всего приложения.

Панель задач, используемая на уровне приложения (Custom Task Pane) реализуется на основе класса **CustomTaskPane**, который также реализо-

ван в пространстве имен **Microsoft.Office.Tools**. Ниже показан пример создания панели задач, используя метод **Add(UserControl, String)**. Данный пример использует свойства объекта **CustomTaskPane** для установки ряда значений по умолчанию и задает обработчики событий **DockPositionChanged** и **VisibleChanged**, реагирующие, соответственно, на изменения местоположения панели и переключение панели с отображаемой на неотображаемую.

```
private MyUserControl myUserControl1;
private Microsoft.Office.Tools.CustomTaskPane myCustomTaskPane;

private void ThisAddIn_Startup(object sender, System.EventArgs e)
{
    myUserControl1 = new MyUserControl();
    myCustomTaskPane = this.CustomTaskPanes.Add(myUserControl1,
        "New Task Pane");

    myCustomTaskPane.DockPosition =
        Office.MsoCTPDockPosition.msoCTPDockPositionFloating;
    myCustomTaskPane.Height = 500;
    myCustomTaskPane.Width = 500;
    myCustomTaskPane.DockPositionRestrict =
        Office.MsoCTPDockPositionRestrict.msoCTPDockPositionRestrictNoHorizontal;
    myCustomTaskPane.Visible = true;

    myCustomTaskPane.VisibleChanged +=
        new EventHandler(myCustomTaskPane_VisibleChanged);
    myCustomTaskPane.DockPositionChanged +=
        new EventHandler(myCustomTaskPane_DockPositionChanged);
}

private void myCustomTaskPane_DockPositionChanged(object sender, EventArgs e)
{
    Microsoft.Office.Tools.CustomTaskPane taskPane =
        sender as Microsoft.Office.Tools.CustomTaskPane;

    if (taskPane != null)
    {
        if (taskPane.DockPosition ==
            Office.MsoCTPDockPosition.msoCTPDockPositionFloating)
        {
            taskPane.Height = 500;
            taskPane.Width = 500;
        }
    }
}
```

```
private void myCustomTaskPane_VisibleChanged(object sender, EventArgs e)
{
    Microsoft.Office.Tools.CustomTaskPane taskPane =
        sender as Microsoft.Office.Tools.CustomTaskPane;

    if (taskPane != null)
    {
        if (taskPane.Visible)
        {
            taskPane.DockPosition =
                Office.MsoCTPDockPosition.msoCTPDockPositionRight;
        }
    }
}
```

Использование Word Content Controls

Компоненты, называемые *Content Controls*, используются в Microsoft Office Word 2007 для структурирования документов. Такой компонент задает область документа, которая может иметь только определенное содержание — текст, дату, графические изображения и т. п. Компоненты *Content Controls* используются для ограничения возможностей пользователей по вводу неверной информации в определенные области документа.

В состав Visual Studio Tools for Office входит следующий набор средств, упрощающий разработку компонентов Content Controls:

- Написание кода, использующего классы на управляемом коде для каждого компонента, входящего в состав Word 2007.
- Обработка событий для каждого компонента.
- Связь компонентов с элементами XML-документов, включаемых в состав документов, базами данных и объектами на управляемом коде.
- Добавление компонентов к документам с помощью специального дизайнера.
  - ☐ Перетаскивание компонентов в документы.
  - ☐ Установка свойств документов.
  - ☐ Создание обработчиков событий по двойному щелчку мышью.
- Программное добавление компонентов к документам в режиме выполнения.

Поддержка компонентов Content Controls реализована в пространстве имен **Microsoft.Office.Tools.Word** в виде классов, каждый из которых представляет собой компонент отдельного типа. Поддерживаются следующие компоненты:

Компонент	Описание
Building Block Gallery	Позволяет пользователям выбрать отдельные блоки для создания документов. К блокам для создания документов могут относиться титульный лист, отформатированная определенным образом таблица, верхние и нижние колонтитулы и т. п. Данный компонент реализован в классе **BuildingBlockGalleryContentControl**
Combo Box	Представляет собой комбинированный список. Реализован в классе **ComboBoxContentControl**
Date Picker	Календарь с возможностью выбора года, месяца, даты, с поддержкой различных форматов представления даты и отображение региональных календарей. Реализован в классе **DatePickerContentControl**
Drop-Down List	Представляет собой выпадающий список. Реализован в классе **DropDownListContentControl**
Group	Представляет собой защищенную область документа, которую пользователь не может редактировать или удалять. Может содержать любые элементы, включая текст, таблицы, графические изображения и другие компоненты. Реализован в классе **GroupContentControl**
Picture	Отображает графику. Реализован в классе **PictureContentControl**
Rich Text	Текстовое поле, которое может содержать таблицы, графические изображения и другие элементы. Реализован в классе **RichTextContentControl**
Plain Text	Текстовое поле, которое может содержать только текст. Реализован в классе **PlainTextContentControl**

Класс **ContentControl** представляет собой основу для одного из перечисленных выше компоненентов — конкретный тип компонента задается свойством **Type**. Поддерживается создание компонентов как в режиме разработки приложения, так и в режиме выполнения.

Выше мы отметили возможность связи компонентов с различными источниками данных. Рассмотрим эту возможность более подробно. Наличие связи компонентов с данными позволяет отображать данные, которые будут обновляться при обновлении данных в источнике, а также вносить изменения в данные, которые будут передаваться источнику. Поддерживаются следующие опции связи с данными:

- Привязка компонентов к полям баз данных или объектам на управляемом коде — поддерживается та же функциональность, что и в Windows Forms.
- Привязка компонентов к элементам XML-документов (которые называются Custom XML Parts), которые встроены в основной документ.

При использовании модели связи с данными Windows Forms, поддерживается связь компонента с одним элементом данных — например колонкой в таблице базы данных. Помимо этого существует возможность связи с данными, используя окно Data Sources в Visual Studio. В следующей таблице показаны компоненты и поддерживаемые ими типы источников данных.

Тип данных	Компонент по умолчанию	Компоненты, которые могут быть связаны с этим типом данных
Boolean, Byte, Char, Double, Enum, Guid, Int16, Int32, Int64, SByte, Single, String, TimeSpan, UInt16, UInt32 и UInt64	PlainTextContentControl	BuildingBlockGalleryContentControl, ComboBoxContentControl, DatePickerContentControl, RichTextContentControl
DateTime	DatePickerContentControl	BuildingBlockGalleryContentControl, ComboBoxContentControl, PlainTextContentControl, RichTextContentControl
Image и массив Byte	PictureContentControl	Нет

Для связи компонентов с элементами XML-документов используется свойство **XMLMapping** соответствующего компонента. В следующем примере показано, как связать компонент типа PlaintextContentControl с элементом Price в ветви Product из XML-документа, входящего в состав основного документа.

```
plainTextContentControl1.XMLMapping.SetMapping
  ("/Product/Price", String.Empty, null);
```

При связывании компонентов с элементами XML-документов автоматически поддерживается т. н. двунаправленное связывание. Таким образом, если пользователь редактирует текст, соответствующий XML-элемент автоматически обновляется. Точно также, если значения XML-элементов обновляются, обновляется и содержимое связанного с ними компонента. Связь с XML-элементами поддерживается для следующих компонентов:

- ComboBoxContentControl;
- DatePickerContentControl;
- DropDownListContentControl;
- PictureContentControl;
- PlainTextContentControl.

Все компоненты поддерживают событийную модель, которая может использоваться для выполнения каких-либо действий, связанных с использованием данных. Событийная модель состоит из следующих событий:

Событие	Описание
ContentUpdating	Используется для выполнения кода перед тем, как Word автоматически обновит текст в компоненте, связанном с XML-элементом
StoreUpdatings	Используется для выполнения кода перед тем, как Word автоматически обновит содержимое XML-элемента, связанного с данным компонентом
Validating	Используется для проверки содержимого компонента на соответствие каким-либо правилам
Validated	Используется для выполнения кода после успешной проверки содержимого компонента

Расширения на основе Outlook Forms

Visual Studio Tools for Office позволяет создавать регионы форм, которые расширяют функциональность стандартных или дополнительных форм Microsoft Office Outlook. В Visual Studio Tools for Office поддерживаются следующие возможности по созданию регионов форм:

- Добавление регионов к проекту используя специальный шаблон — **Form Region**.
- Задание расположения региона, типа региона и других характеристик используя специальный «мастер» — **Form Region Wizard**.
- Визуальная разработка регионов с помощью специального дизайнера:
  - ☐ Перетаскивание компонентов в регионы.
  - ☐ Управление расположением компонентов и их характеристиками.
  - ☐ Создание обработчиков событий по двойному щелчку мышью.
- Добавление кода на Visual C# или Visual Basic с полной поддержкой проверки типов и технологии IntelliSense.
- Отладка проектов с использованием средств, которые автоматически запускают Outlook и генерируют файлы и конфигурационные настройки, позволяющие Outlook обнаружить и загрузить регионы форм.

Использование нового «мастера» — **New Outlook Form Region** позволяет импортировать любой регион формы, созданный средствами Microsoft Office Outlook. При использовании Microsoft Office Outlook для дизайна регионов форм имеется возможность использовать поля Outlook и компоненты, входящие в состав Outlook, которые не доступны через Visual Studio **Toolbox**. Помимо этого предоставляется возможность повторного использования регионов, разработанных для других проектов на основе Outlook. После того как регион импортирован, появляется возможность написания кода для обработчиков событий, используя штатные средства, предоставляемые Microsoft Visual Studio.

В следующей таблице показаны все возможные подходы к настройке и расширению интерфейса офисных приложений, поддерживаемые в Visual Studio Tools for Office.

	Action Pane	Smart Tags	Custom Task Pane	Регионы форм в Outlook	Настройка «ленты»	Меню и панели инструментов
Тип проекта	На уровне документа	На уровне документа	На уровне приложения	На уровне приложения	На уровне документа и на уровне приложения	На уровне документа и на уровне приложения
Приложения	Excel 2003, Excel 2007, Word 2003 и Word 2007	Excel 2003, Excel 2007, Word 2003 и Word 2007	Excel 2007, InfoPath 2007, Outlook 2007, PowerPoint 2007 и Word 2007	Outlook 2007	Excel 2007, Outlook 2007, PowerPoint 2007 и Word 2007	Excel 2003, Outlook 2003, PowerPoint 2003, Project 2003, Visio 2003 и Word 2003
Поддержка компонентов	Да, Windows Forms	Нет	Да, Windows Forms	Да, компоненты Outlook	Да, компоненты «ленты»	Нет для меню

Создание SharePoint Workflow

Visual Studio Tools for Office позволяет создавать workflow для SharePoint — использование workflow позволяет ассоциировать документы, хранящиеся в Microsoft Office SharePoint Server 2007 с процессами. Следующие возможности Visual Studio Tools for Office облегчают создание SharePoint workflow:

- Создание проектов для SharePoint workflow, используя шаблоны **SharePoint 2007 Sequential Workflow** и **SharePoint 2007 State Machine Workflow**.
- Отладка логики workflow нажатием клавиши F5. Visual Studio Tools for Office автоматически ассоциирует workflow с библиотекой документов по умолчанию на локальном портале SharePoint и запускает экземпляр workflow.

С помощью «мастера» **New Office SharePoint Workflow** пользователи могут указать конфигурацию для отладки. «Мастер» поддерживает следующие опции:

- Указание сайта SharePoint, библиотеки и списков, которые могут использоваться при отладке SharePoint workflow.
- Указать, какие действия запускают workflow.

Для создания проекта по разработке SharePoint workflow необходимо выполнить следующие шаги.

1. В диалоговой панели **New Project** в панели **Project Types** выбрать язык программирования — Visual Basic или Visual C# и выбрать одну из ветвей (все они содержат одинаковые шаблоны для SharePoint workflow:
   - ☐ Workflow;
   - ☐ Office;
   - ☐ Ветвь 2007 в ветви Office.
2. В зависимости от типа создаваемого workflow выбрать либо шаблон **SharePoint 2007 Sequential Workflow**, либо **SharePoint 2007 State Machine Workflow** и задать имя проекта.
3. В панели «мастера» **New Office SharePoint Workflow** задать опции отладки — они будут использоваться Visual Studio Tools for Office при развертывании workflow при его отладке из среды Visual Studio.
4. На странице **Specify the workflow name and site for debugging** необходимо указать SharePoint Server имя, которое будет использоваться для идентификации шаблона на сайте.
5. Далее необходимо указать адрес портала, который будет использоваться для выполнения разрабатываемого workflow. Портал должен располагаться на локальном компьютере. По умолчанию используется адрес http://localhost/Docs.
6. Щелкните кнопку **Next**.
7. Если текущий портал не содержит списка history list, «мастер» создаст такой список автоматически, так как он используется для протоколирования всех событий, связанных с выполнением workflow — например, создание workflow, создание или завершение задания и т. п.
8. На странице **Select the lists you will use when debugging** укажите библиотеку или список, которые будут использоваться для отладки workflow. Например, если разрабатываемый код предназначен для управления документом, следует указать библиотеку документов.
9. Далее необходимо включить опцию **Automatically Associate Workflow**, которая позволит автоматически добавлять шаблон workflow к указанной библиотеке или списку при отладке. В противном случае шаблон нужно будет добавлять вручную, используя страницу администратора **Add a Workflow** на сайте SharePoint.
10. Задайте список задач, который вы собираетесь использовать. Список задач отображает задачи в рамках workflow, доступные каждому участнику процесса.
11. Задайте список, который будет отображать все события, возникающие при выполнении workflow.
12. Нажмите кнопку **Next**.

13. На странице **You can specify the conditions for how your workflow is started** можно указать опции, которые будут использоваться при запуске workflow.
14. После того как работа с «мастером» завершена, вы попадаете в дизайнер Designer for Windows Workflow Foundation, в котором происходит написание кода для workflow.

Непосредственная разработка workflow начинается с создания расписания — workflow schedule, которое представляет собой набор активностей, отражающих выполняемые действия. К расписанию добавляются необходимые активности — в зависимости от того, для решения каких задач предназначается workflow. При необходимости, добавляются обработчики событий. К наиболее часто обрабатываемым событиям относятся события **onWorkflowActivated** и **onWorkflowItemChanged**.

Для тестирования workflow необходимо установить «точку остановки», например, на обработчике события **onWorkflowActivated**. После этого нужно выполнить действия над документом, приводящие к активизации шагов workflow.

Новые механизмы защиты и развертывания приложений на основе Microsoft Office

Разработчикам предоставляется возможность использования технологии ClickOnce для обеспечения безопасности и развертывания решений, созданных средствами Visual Studio Tools for Office для платформы Microsoft Office 2007, включая расширения на базе документов и расширения на базе офисных приложений. В Visual Studio Tools for Office поддерживаются следующие средства развертывания приложений:

- Мастер **Publish Wizard** для публикации и развертывания настроек и расширений.
- Автоматическое определение наличия обновлений для созданных расширений — поддерживается проверка наличия обновлений в заданные интервалы времени, загрузка и установка обновлений, а также откат к более ранним версиям расширений.
- Возможность загрузки и выполнения решений на основе Office даже в тех случаях, когда пользователь не подключен к сети.

Модель безопасности, реализованная в технологии ClickOnce, разработана таким образом, чтобы быть совместимой с будущими версиями среды выполнения Visual Studio Tools for Office, Microsoft .NET Framework и Microsoft Office. В Visual Studio Tools for Office поддерживаются следующие ключевые возможности по обеспечению безопасности.

- Для Microsoft Office 2007 предоставляется возможность обеспечения безопасности решений, используя независимую от версий модель безопасности ClickOnce.
- Использование Trust Center в Microsoft Office для выбора наиболее оптимальных для вашего решения настроек безопасности, включая подписание манифеста развертывания сертификатами безопасности, отображение информации о безопасности на уровне ClickOnce или программно добавив элементы в список пользователей.
- Задание политик безопасности для каждого индивидуального пользователя.

Компоненты VSTO 2008

Ключевые компоненты VSTO 2008 располагаются в каталоге \Program Files\Microsoft Visual Studio 9.0\Visual Studio Tools for Office. В корне расположены адаптеры для загрузки в Visual Studio в режиме дизайна Excel 2003/2007 и Word 2003/2007. Также здесь расположен «переходник» для использования компонентов, написанных на управляемых языках, в режиме дизайна и библиотека типов для дизайнера. Подкаталог PIA содержит сборки для реализации взаимодействия управляемого и неуправляемого кода (Primary Interop Assembly) как для версии Office 2003 (каталог Office11), так и для Office 2007 (каталог Office12). В каталоге vspkgs находятся расширения Visual Studio для реализации различных дизайнеров, а в каталоге VSTOTemplates мы найдем шаблоны всех типов проектов, поддерживаемых в Visual Studio Tools for Office 2008.

В следующем разделе мы рассмотрим новинки в Visual Studio 2008, связанные с созданием сервисов, использованием Windows Communication Foundation и Windows Workflow Foundation.

Глава 7

Создание сервисов

В данном разделе мы рассмотрим возможности создания и использования новых сервисов, появившихся в Microsoft .NET Framework 3.0, расширенных в .NET Framework 3.5 и поддерживаемых на уровне Microsoft Visual Studio 2008. Речь пойдет о двух ключевых компонентах Microsoft .NET Framework 3.0 — унифицированной программной модели и среде для выполнения сервисов — Windows Communication Foundation (WCF), и гибкой, декларативной среде выполнения workflow и активностей — Windows Workflow Foundation (WF).

Windows Communication Foundation (WCF)

Windows Communication Foundation — это платформа Microsoft для создания распределенных систем, которая включает лучшие характеристики таких технологий, как MSMQ, DCOM, ASMX Web services, .NET Remoting и ряд других. Windows Communication Foundation представляет собой новое поколение коммуникационной инфраструктуры, построенной на основе архитектуры веб-сервисов. Расширенная поддержка веб-сервисов в Windows Communication Foundation обеспечивает безопасную, надежную, транзакционную систему передачи сообщений с возможностью реализации гетерогенных систем.

Ориентированная на сервисы программная модель, реализованная в Windows Communication Foundation, построена на основе Microsoft .NET Framework и существенно облегчает и упрощает создание систем на базе развитых коммуникационных механизмов.

Windows Communication Foundation унифицирует широкий спектр возможностей коммуникационных систем, предоставляя компонентную и расширяемую архитектуру, обширный набор транспортных протоколов, систему безопасности, шифрование, сетевые топологии и модель хостинга. Ниже перечислены основные характеристики WCF.

- Унифицированная программная модель.
- Широкий набор коммуникационных возможностей.
- Взаимодействие.
- Поддержка корпоративных сервисов.
- Нейтральность к транспортному уровню, протоколам и форматам.
- Масштабируемость.

Windows Communication Foundation поддерживается на следующих платформах:

- Microsoft Windows Vista.
- Microsoft Windows XP.

- Microsoft Windows Server 2003.
- Microsoft Windows Server 2008.

При использовании Visual Studio 2005 разработчикам требовалось загрузить и установить специальный набор расширений — «Visual Studio 2005 extensions for Framework .NET 3.0 (WCF & WPF)» для того, чтобы получить шаблоны проектов для разработки с использованием WCF. В состав Visual Studio 2008 входят полноценные средства работы с WCF, включая все необходимые шаблоны, поддержку на уровне дизайнеров и т. п.

Шаблоны для WCF в составе Visual Studio 2008

К новинкам, появившимся в .NET Framework 3.5 и Visual Studio 2008 для поддержки работы с WCF, отнесем следующие:

- Поддержка авто-хостинга, клиента для тестирования сервисов, новый редактор конфигураций, средства отладки:
  - ☐ В предыдущих версиях Visual Studio создание сервиса также требовало написания кода для хостинга в тех случаях, когда сервис использовался не под управлением Internet Information Services (IIS). Функция **autohost**, реализованная в Visual Studio 2008, позволяет разработчикам не заботиться о хостинге сервисов.
  - ☐ В предыдущих версиях Visual Studio, после того как вы создали сервис, вам требовалось создать клиентское приложение для тестирования данного сервиса. В Visual Studio 2008 клиент для тестирования генерируется автоматически — данная функциональность схожа с автоматической генерацией клиента для тестирования веб-сервисов на основе ASMX.

Создание простейшего WCF-сервиса средствами Visual Studio состоит из ряда простых шагов. Мы начинаем с того, что выбираем шаблон — WCF Service Library, который расположен в ветви WCF для поддерживаемых языков программирования — Visual Basic .NET и Visual C#. Ниже показан код, создаваемый на основе шаблона WCF Service Library.

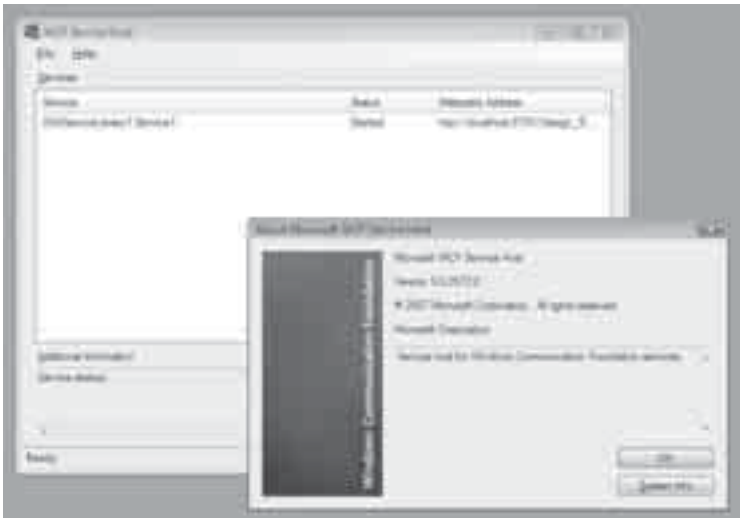
```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Runtime.Serialization;
using System.ServiceModel;
using System.Text;

namespace WcfServiceLibrary1
{
```

```
// NOTE: If you change the interface name "IService1" here,
// you must also update the reference to "IService1" in App.config.
    [ServiceContract]
    public interface IService1
    {
        [OperationContract]
        string GetData(int value);

        [OperationContract]
        CompositeType GetDataUsingDataContract(CompositeType composite);

        // TODO: Add your service operations here
    }

    // Use a data contract as illustrated in the sample below to
    // add composite types to service operations
    [DataContract]
    public class CompositeType
    {
        bool boolValue = true;
        string stringValue = "Hello ";

        [DataMember]
        public bool BoolValue
        {
            get { return boolValue; }
            set { boolValue = value; }
        }

        [DataMember]
        public string StringValue
        {
            get { return stringValue; }
            set { stringValue = value; }
        }
    }
}
```

Создаваемый на основе шаблона код сервиса готов к тестированию — нажатие клавиши F5 приводит к:

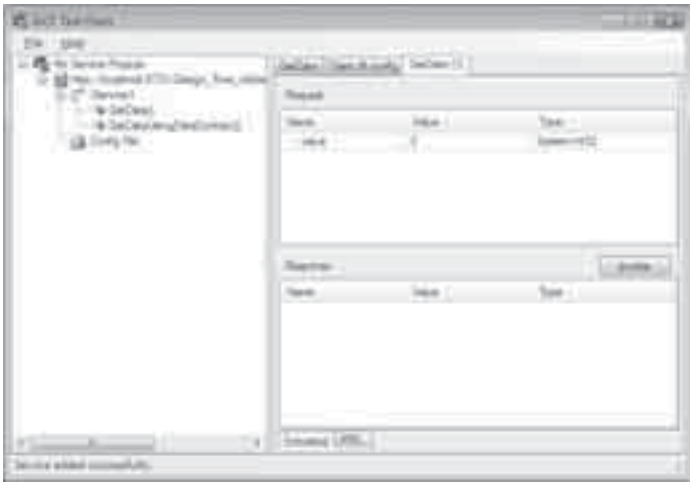
- автоматической генерации хоста для данного сервиса (**WcfSvcHost**);

Автоматически сгенерированный хостинг для сервиса (WcfSvcHost)



Автоматический хостинг для WCF-сервиса

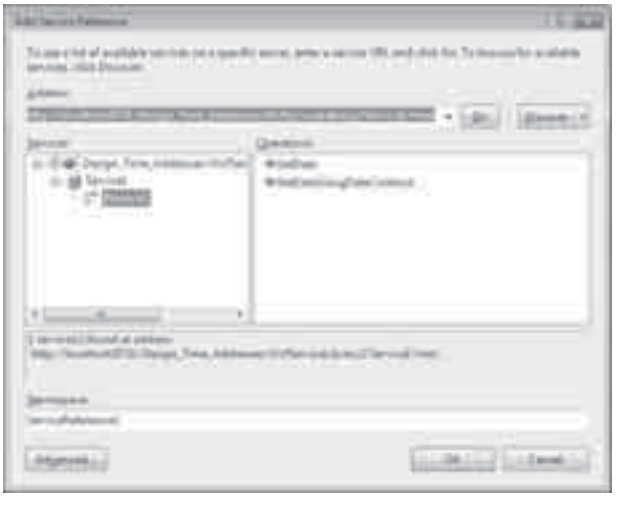
- а также тестового клиента.



Тестовый клиент для WCF-сервиса

Для того чтобы создать клиентское приложение, которое будет потреблять наш сервис, необходимо выполнить следующие простые шаги:

1. Выполнить команду **Project | Add Service Reference** и в диалоговой панели **Add Service Reference** щелкнуть кнопку **Discover**.



Добавление ссылки на сервис

2. Выбрать найденный сервис и щелкнуть кнопку **Ok** — ссылка на сервис будет добавлена к проекту.
3. Выполнить команду **File | Add | New Project** и выбрать шаблон **Windows Forms Application**.
4. На форму добавить три компонента — кнопку и два текстовых поля. В обработчике события, связанного с щелчком кнопки, написать следующий код:

```
Private Sub Button1_Click(ByVal sender As System.Object,
                          ByVal e As System.EventArgs)
                          Handles Button1.Click
‘ Использование ссылки на сервис
 Dim client As New ServiceReference1.Service1Client
 Dim returnString As String

 returnString = client.GetData(TextBox1.Text)
 TextBox2.Text = returnString
End Sub
```

5. В **Solution Explorer** выбрать **WindowsApplication1** и по щелчку правой кнопки мыши выбрать опцию **Set As StartUp Project** — в противном случае будет использоваться автоматически сгенерированный тестовый клиент.

6. Запустить приложение на выполнение и убедиться в том, что сервис работает так, как и ожидалось.

Тестирование сервиса из Windows-клиента

К дополнительным новинкам, появившимся в .NET Framework 3.5 и Visual Studio 2008 для поддержки работы с WCF, отнесем следующие:

- Поддержку программирования на уровне протокола HTTP, поддержку протокола REST и синдикации.
  - Поддержка программирования на уровне протокола HTTP означает, что разработчики могут полностью использовать возможности протокола REST, в частности, можно более просто интегрировать WCF с веб-приложениями, например для поставки данных через WCF-сервисы в приложения, созданные на основе ASP.NET.
- Поддержку передачи сообщений на уровне протокола JSON.
  - Протокол JSON (JavaScript Object Notation) особенно полезен при создании приложений на основе ASP.NET AJAX — JSON используется для взаимодействия с компонентами AJAX. Поддержка на уровне WCF означает, что у разработчиков появляется возможность создания сервиса, который будет поставлять данные компонентам ASP.NET AJAX, расположенным на клиенте.
- Поддержку новых протоколов стандарта OASIS.
  - Windows Communication Foundation 3.5 является одним из первых программных компонентов, полностью поддерживающих новые протоколы стандарта OASIS — это означает, что разработчики могут использовать новые типы «привязок» при создании WCF-сервисов.
- Интеграцию с Windows Workflow Foundation (WF).
  - В предыдущих версиях, использование и WCF и WF требовало написания собственного инфраструктурного кода. В Visual Studio 2008 такой код, обеспечивающий связывание двух технологии, генерируется автоматически.

Windows Workflow Foundation (WF)

Windows Workflow Foundation (WF) — программный компонент для создания и выполнения workflow, который поставляется в составе .NET Frame-work 3.0 и более поздних версий. Workflow — это набор активностей (activity), координирующих действия людей или программного обеспечения.

Workflow может быть представлено как логическая последовательность активностей (flowchart) или как диаграмма состояний (state diagram).

Логическая последовательность	Диаграмма состояний
Предопределенная последовательность действий	Внешние воздействия
Последовательные по своей природе действия	Непредсказуемая последовательность действий
Сценарий — управление процессом	Множество альтернативных «путей»
	Возможность перехода на любой шаг

Workflow создается в Visual Studio с помощью специального дизайнера (входит в комплект поставки WF). Возможно создание workflow целиком с помощью кода, с описанием на языке XOML и включением фрагментов кода (code behind) или только на языке XOML. Созданное описанными выше средствами workflow выполняется внутри приложения (hosting application). Еще один способ создания workflow — генерация дерева активностей на лету с последующей сериализацией.

Активности могут быть базовыми — определяют один шаг в рамках workflow, или составными — содержать другие активности. С точки зрения .NET Framework, активности — это полноценные классы, обладающие свойствами, методами и событиями. Разработчики могут создавать собственные библиотеки активностей или расширять существующие.

В состав WF входит набор предопределенных активностей — активности, управляемые правилами, управление последовательностями, транзакции, коммуникации (локальные и веб-сервисы). Базовые активности показаны в следующей таблице.

Управление последовательностью	Sequence, Parallel, While, IfElse, Listen, EventDriven, ConditionedActivityGroup, Replicator, Delay
Транзакции и исключения	ExceptionHandler, Throw, Compensate, Suspend, Terminate
Данные и формы	UpdateData, SelectData, WaitForData, WaitForQuery
Коммуникации	InvokeWebService, WebServiceReceive, WebService-Response, InvokeMethod, EventSink
Выполнение кода	Code
Дополнительные активности	StateInitialization, State, SetState

Workflow поддерживают состояния, возможность сохранения в потоки (файловая система, реляционное хранилище), транзакции, потоки, таймеры и ряд других сервисов, включая интеграцию с ASP.NET.

Имеется возможность использования дизайнера workflow как компонента для Windows Forms.

Ядро выполнения workflow позволяет «выставлять» workflow в виде веб-сервисов.

Также имеется возможность «общения» с локальными объектами внутри хост-приложения (local communication services) на уровне обмена данными, вызова методов и обработки событий.

Помимо перечисленных выше возможностей, в WF поддерживаются адаптивные workflow — предоставляется механизм для динамического изменения экземпляров workflow в режиме выполнения (dynamic update). Эта функциональность доступна как из самого выполняемого workflow, так и из хост-приложения.

Windows Workflow Foundation поддерживается на следующих платформах:

- Microsoft Windows Vista.
- Microsoft Windows XP.
- Microsoft Windows Server 2003.
- Microsoft Windows Server 2008.

В состав Visual Studio входит набор шаблонов для использования WF в приложениях, включая возможность разработки workflow для Microsoft Office SharePoint Server 2007 (см. главу 6).

Шаблоны для Windows Workflow Foundation

Набор активностей, входящих в состав Windows Workflow Foundation 3.0, показан на следующем рисунке.

Активности, входящие в состав WF

Создание проектов с использованием WF состоит из последовательности простых действий, перечисленных ниже.

1. Необходимо выбрать тип проекта, который мы будем использовать для реализации workflow. Команда **File | New | Project** приводит к отображению диалоговой панели **New Project**, в которой есть ветвь **Workflow** для поддерживаемых языков программирования — Visual Basic .NET и Visual C#. В зависимости от задачи, мы можем выбрать один из следующих шаблонов:
   - ☐ **Sequential Workflow Console Application** — логическая последовательность активностей в консольном приложении.
   - ☐ **Sequential Workflow Library** — библиотека логических последовательностей.
   - ☐ **Workflow Activity Library** — создание библиотеки активностей.
   - ☐ **State Machine Workflow Console Application** — диаграмма состояний в консольном приложении.
   - ☐ **State Machine Workflow Library** — библиотека диаграмм состояний.
   - ☐ **Empty Workflow Project** — пустой проект для создания приложений с использованием WF.
2. Наиболее простой способ для освоения WF — это создание консольных приложений либо с использованием логической последовательности активностей, либо с использованием диаграммы состояний.
3. И в том, и в другом случае мы попадаем в дизайнер workflow, который позволяет нам составлять workflow из активностей, расположенных в галерее.

Дизайнер workflow для логической последовательности

При создании приложения на базе шаблона **Empty Workflow Project** у разработчиков появляется возможность создавать более сложные workflow, объединяющие логические последовательности и диаграммы состояний — необходимо использовать команду **Project | Add** для выбора добавляемых к проекту элементов — логическая последовательность, диаграмма состояний или активность.

Добавление элементов проекта

Выбор любой из опций приводит к появлению диалоговой панели Add New Item, в которой можно выбрать соответствующий шаблон:

Доступные шаблоны

Обратите внимание на то, что и для активностей, и для логической последовательности и диаграммы состояний поддерживается два режима работы — на уровне кода и на уровне XML с выделением кода в отдельный файл.

К новинкам, появившимся в Visual Studio 2008, относятся возможность интеграции между Windows Workflow Services и Windows Communication Foundation — в этих сценариях WCF-сервисы могут выполнять роль workflow, а Workflow может хоститься как WCF-сервис, а также новые средства создания контрактов. Эти расширения позволяют, в частности, использовать сервисы хранения состояния WF в WCF и модель безопасности, реализованную в WCF в ряде сценариев создания workflow.

В следующей главе мы рассмотрим ключевые новинки, связанные с созданием приложений для мобильных устройств.

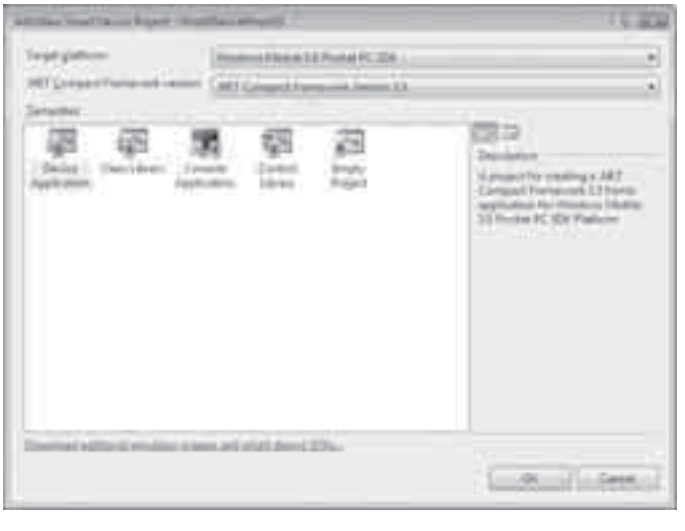
Глава 8

Создание мобильных приложений

В Visual Studio 2008 входит богатый набор интегрированных средств для создания приложений, выполняющихся на платформах Windows CE и Windows Mobile. Разработчикам предоставляется возможность создания приложений на управляемом коде, выполняемом под управлением .NET Compact Framework, используя те же средства редактирования, дизайна и отладки, что и при создании настольных приложений. Для того чтобы начать создание приложений для мобильных устройств, необходимо выбрать шаблон **Smart Device Project** для поддерживаемых языков программирования — Visual Basic .NET и Visual C# — шаблон находится в ветви **Smart Device** панели **New Project**. Помимо шаблонов и средств разработки и дизайна, в состав Visual Studio также входят эмуляторы для различных устройств, позволяющие выполнять и отлаживать код на компьютере, а также средства для упаковки приложений и их ресурсов в CAB-файлы для развертывания на устройствах конечных пользователей.

После выбора шаблона **Smart Device Project** разработчики попадают в диалоговую панель **Add New Smart Device Project**, в которой можно выбрать:

- платформу для разработки — возможен выбор следующих платформ:
  - ☐ Pocket PC 2003;
  - ☐ Windows CE;
  - ☐ Windows Mobile 5.0 Pocket PC SDK;
  - ☐ Windows Mobile 5.0 Smartphone SDK.



Панель настроек проекта Smart Device Project

- версию .NET Compact Framework — 2.0 или 3.5;
- и один из следующих шаблонов приложения:

- **Device Application** — позволяет создавать приложения на основе .NET Compact Framework Forms для платформы Windows Mobile 5.0 Pocket PC SDK или другой выбранной платформы;
- **Class Library** — позволяет создавать библиотеку классов .dll) для выбранной платформы;
- **Console Application** — позволяет создавать приложения без графического интерфейса для выбранной платформы;
- **Control Library** — позволяет создавать компоненты для выбранной платформы;
- **Empty Project** — представляет собой пустой проект для создания приложений для выбранной платформы.

Помимо этого, панель **Add New Smart Device Project** позволяет загружать дополнительные эмуляторы и SDK для различных устройств, не входящие в базовый комплект поставки Visual Studio.

Среда разработки для мобильного устройства

Выбор наиболее распространенного шаблона — **Device Application** — приводит к появлению среды разработки (показана на следующем рисунке), состоящей из эмулятора устройства и знакомых разработчикам галереи компонентов, окна установки свойств и окна управления проектом. Работа, как и в случае с Windows Forms, заключается в выборе визуальных компонентов для создания интерфейса, перетаскивании их на форму, установке их свойств и написании кода обработчиков событий — редактор вызывается двойным щелчком мышью по соответствующему компоненту и написании логики самого приложения.

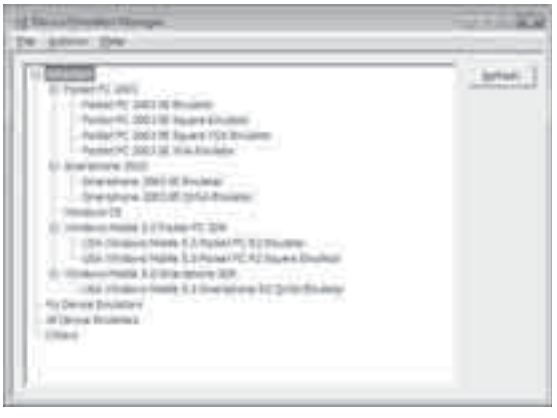
Галерея содержит набор визуальных компонентов, которые можно использовать для создания интерфейсов мобильных приложений. Компоненты расположены в следующих группах:

- **Common Controls** — содержит наиболее часто используемые компоненты — **CheckBox**, **Label**, **ListView**, **TextBox**, **WebBrowser** и т. п.
- **Device Containers** — содержит компонент **Panel**, используемый для размещения других компонентов на экране.
- **Device Menus & Toolbars** — содержит средства для создания меню и панелей задач — **MainMenu**.
- **Device Data** — содержит средства для работы с данными — **BindingSource**, **DataSet** и **DataGrid**.
- **Device Components** — содержит дополнительные компоненты — **ImageList**, **MessageQueue**, **SerialPort** и **Timer**.

Разработка приложений на управляемом коде происходит с использованием .NET Compact Framework версии 3.5, которая представляет собой подмножество .NET Framework, выполняемое на мобильном устройстве под управлением оптимизированной версии Common Language Runtime (CLR) с поддержкой таких технологий, как Windows Communication Foundation (WCF) и Windows Forms. Помимо этого в состав .NET Compact Framework входят классы, специально разработанные для этой платформы и позволяющие управлять различными характеристиками мобильных устройств.

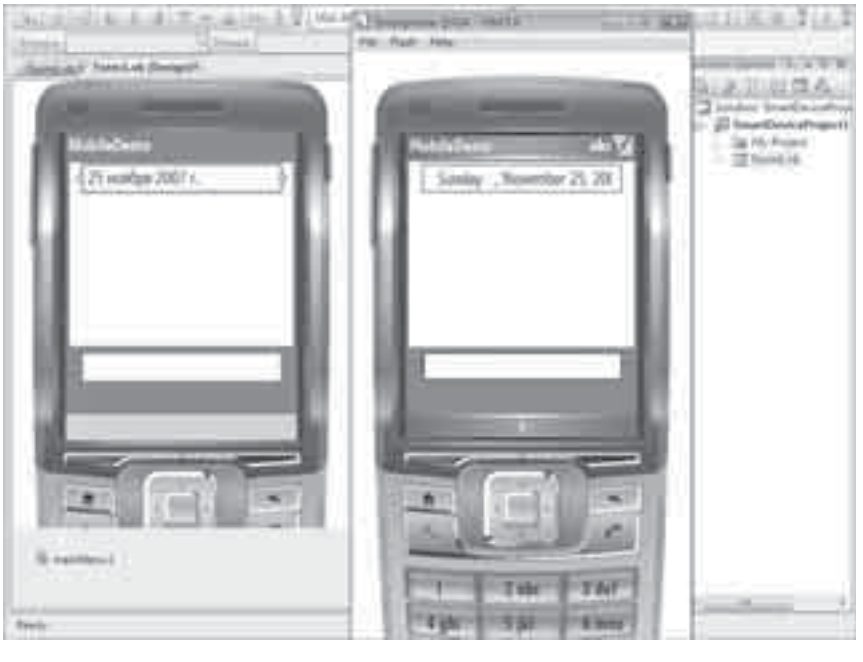
После того как приложение разработано, его можно проверить под одним из доступных в составе Visual Studio (или загруженных дополнительно) эмуляторов. Для этого необходимо выполнить следующие действия:

1. Выбрать опцию **Tools | Device Emulator Manager** для получения списка доступных эмуляторов устройств.



Device Emulator Manager

2. Используя команду **Actions | Connect**, подключиться к выбранному эмулятору для изучения его возможностей.
3. Запустить приложение на выбранном эмуляторе для того, чтобы убедиться, что оно корректно работает — Visual Studio выполнит все необходимые действия по упаковке приложения и требуемых дополнительных компонентов (Compact Framework и т. п.) и загрузке их в устройство.
4. Оттестировать приложение под управлением эмулятора.



Выполнение приложения под управлением эмулятора мобильного устройства

К новинкам для создания приложений для мобильных устройств, появившимся в Visual Studio 2008, относятся:

- Средства тестирования для проектов типа Smart Device — к этим средствам относятся возможность выполнения модульного тестирования (unit testing) и выполнение тестов как на реальных физических устройствах, так и под управлением эмулятора.
- Средства программного управления устройствами через Smart Device Connectivity API. Программные интерфейсы, включенные в состав Smart Device Connectivity API, позволяют выполнять следующие задачи:
  - ☐ Выполнять поиск доступных платформ и устройств;
  - ☐ Осуществлять соединение между локальным компьютером и мобильным устройством;
  - ☐ Развертывать приложения и дополнительные компоненты на мобильном устройстве;
  - ☐ Запускать, останавливать и получать информацию о процессах, запущенных на мобильном устройстве;
  - ☐ Обмениваться данными между локальным компьютером и мобильным устройством;
  - ☐ Программные интерфейсы Smart Device Connectivity API реализованы в пространстве имен **Microsoft.SmartDevice.Connectivity**.
- Средства управления безопасностью для Windows Mobile. Данные средства позволяют:
  - ☐ Имитировать различные настройки безопасности на мобильных устройствах;
  - ☐ Применять стандартные настройки безопасности;
  - ☐ Применять расширенные настройки безопасности;
  - ☐ Экспортировать настройки безопасности на локальный компьютер;
  - ☐ Добавлять и удалять сертификаты;
  - ☐ Для управления настройками безопасности используется утилита **RapiConfig.exe**.
- Встроенная поддержка Windows Mobile 5.
  - ☐ В состав Visual Studio 2008 входят эмуляторы, шаблоны проектов и библиотеки для создания приложений под Windows Mobile 5. К ним относятся: Windows Mobile 5 Pocket PC SDK и Windows Mobile 5 Smartphone SDK.
- Встроенная поддержка .NET Compact Framework 3.5.
- Поддержка разработки под Microsoft SQL Server Compact 3.5. Издание Microsoft SQL Server Compact 3.5 заменяет SQL Server Mobile Edition и предоставляет в распоряжение разработчиков новые средства дизайна с поддержкой создания приложений на Visual C# и Visual Basic.

- Новые версии эмуляторов устройств — **Device Emulator 3.0**.
- Новая диалоговая панель для управления проектами — **Project Dialog**, позволяющая выбирать устройства, версии .NET Compact Framework и соответствующие шаблоны.

Помимо новых средств, входящих в состав Visual Studio 2008, ряд новинок появился на уровне .NET Compact Framework 3.5. К ним относятся:

- Поддержка Windows Communication Foundation — клиентские приложения, выполняемые под управлением .NET Compact Framework, могут соединяться с веб-сервисами, созданными средствами WCF. Помимо этого, поддерживается новый транспортный уровень — **Microsoft Exchange Server Mail Transport**, доступный как для .NET Compact Framework, так и для приложений, работающих на обычных компьютерах.
- Поддержка LINQ — обеспечивает унифицированный доступ к данным, хранимым в реляционных базах данных, XML-файлах и различных структурах, располагаемых в памяти.
- Расширения в Windows Forms. К расширениям относятся возможность использования графики в компонентах **TabPage, Panel, Splitter и PictureBox**, поддержка шрифтов ClearType в компонентах на базе класса **Control** и новые свойства компонента ComboBox — **SelectionStart** и **SelectionLength**.
- Компонент **SoundPlayer** для воспроизведения различных звуковых сигналов.
- Функции сжатия данных – поддерживаются следующие классы в пространстве имен **System.IO.Compression** — **CompressionMode**, **DeflateStream** и **GZipStream**.
- Поддержка делегатов через метод **CreateDelegate**.
- Средства профилирования — .NET Compact Framework CLR Profiler для просмотра состояния кучи, процессов и работы сборщика мусора. .NET Compact Framework CLR Profiler и документация включены в **Power Toys for .NET Compact Framework** — набор бесплатных утилит, загружаемых с сайта Microsoft.
- Средства конфигурации — позволяют выполнять ряд административных функций, включая указание версии .NET Compact Framework, под которой должно выполняться приложение. Средства конфигурации включены в **Power Toys for .NET Compact Framework** — набор бесплатных утилит, загружаемых с сайта Microsoft.

В следующей главе мы познакомимся с ключевыми ресурсами по основным темам, связанным с использованием .NET Framework 3.5 и Microsoft Visual Studio 2008.

Глава 9

Знакомство с Microsoft Visual Studio Team System 2008

С появлением в линейке средств разработки продукта под названием Visual Studio 2005 Team System компания Microsoft вышла за рамки базового процесса разработки — редактирование-компилирование-откладка, и предложила набор инструментов, ранее доступных только от сторонних производителей. Средства, включенные в состав Microsoft Visual Studio 2005 Team System, предназначены для более ранних шагов цикла разработки, таких как дизайн и архитектура, а также шагов, выполняемых после непосредственной разработки — тестирование и развертывание. Включение подобных средств в состав Visual Studio 2005 позволяет использовать его на всех этапах создания программного продукта.

Новая версия — Visual Studio Team System 2008 продолжает развитие семейства продуктов для управления всем жизненным циклом создания приложений и содержит новые и улучшенные средства, процессы и руководства, которые помогут улучшить совместную командную работу и сделать ее более эффективной. Инструменты, входящие в состав Visual Studio Team System 2008, позволяют наладить более эффективные коммуникации между членами проектной группы и заказчиками, наладить эффективную совместную работу, обеспечить ожидаемое заказчиками качество кода, используя расширенные средства контроля качества, получить представление об активностях в рамках проекта и приоритетах, которые позволят принимать решения, основываясь на данных, предоставляемых в реальном времени.

Использование Visual Studio Team System в Microsoft

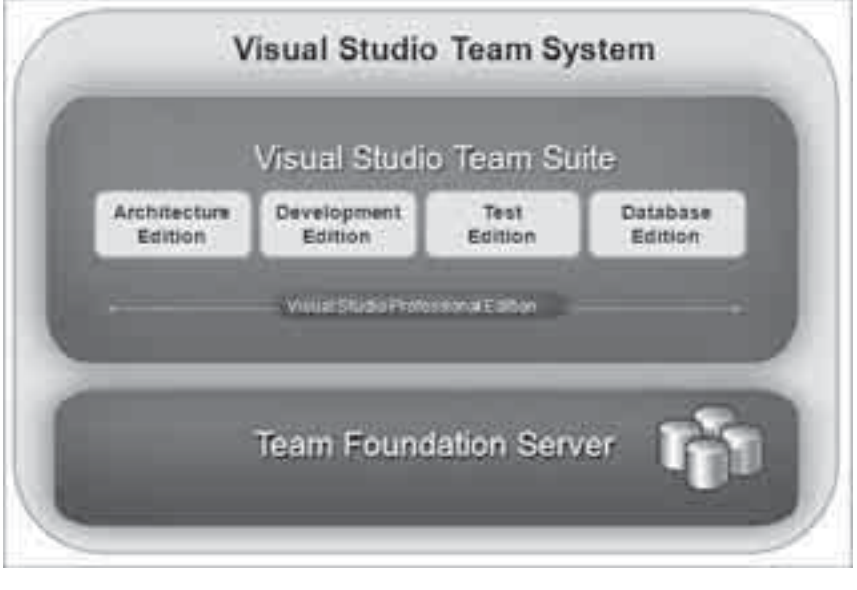
Проектные команды, отвечающие за разработку ключевых продуктов Microsoft (Windows, Office, SQL Server, Development Tools), активно используют Visual Studio Team System. Помимо этого, подразделение Microsoft IT также использует этот продукт для создания проектов для внутреннего использования. Вот лишь несколько цифр. В Microsoft установлены более 20 копий Team Foundation Server (большинство — это Beta 2 Team Foundation Server 2008, остальные — ранние версии Visual Studio Team System «Rosario»), на которых поддерживается более 730 проектов и более 5600 активных пользователей.

Роли в программном проекте

Задача любой формализованной методологии — структурировать процесс разработки программного обеспечения. В частности, любая методология определяет роли в программном проекте. Наиболее часто используются следующие роли:

- **Архитектор** отвечает за высокоуровневую структуру приложения, требования, ключевые технологии для реализации, взаимодействие приложения с другими приложениями, инфраструктурой и т. п.
- **Разработчик** занимается написанием и отладкой кода. Предыдущие версии Visual Studio, а также продукты, входящие в состав Visual Studio 2008 — Visual Studio Standard Edition и Visual Studio 2008 Professional Edition — предназначаются, в первую очередь, для этой роли.
- **Тестировщик** отвечает за обнаружение ошибок, их исправление; должен убедиться в том, что приложение соответствует изначальным требованиям (функциональным и нефункциональным).
- **Разработчик баз данных** занимается разработкой приложений, использующих базы данных (работа со схемой базы данных, создание хранимых процедур и т. п.).
- **Менеджер проекта** отвечает за ведение проекта, проектный офис, метрики, сроки и т. п.

Важно отметить, что каждая роль не обязательно соответствует наличию физического члена команды. В небольших проектах возможно совмещение ролей. Тем не менее, даже если один человек выполняет несколько ролей, для выполнения каждой задачи он использует разные средства.



Visual Studio Team System 2008

Visual Studio Team System 2008 — это продукт, который призван облегчить задачи, выполняемые каждой из перечисленных выше ролей. В состав продукта входят четыре т. н. ролевых издания, построенных поверх Visual Studio 2008 Professional Edition — **Architecture Edition, Development**

Edition, Database Edition (это издание впервые появилось в Visual Studio Team System 2008) и **Test Edition** — вместе эти издания называются Team Suite, а также серверный продукт, обеспечивающий совместную работу над проектами (совместный доступ к ресурсам, контроль версий, порталы на базе SharePoint, отчеты, слежение за элементами проекта и т. п.) — **Team Foundation Server.**

Командная разработка

В основе Visual Studio 2005 Team System находится серверное приложение под названием Team Foundation Server, состоящее из базы данных на SQL Server (используемой для хранения проектных данных) и компонентов среднего уровня на базе Windows Server 2003, ASP.NET и Windows SharePoint Services. Основные компоненты Team Foundation Server показаны на следующей диаграмме.

Задача Team Foundation Server — обеспечить инфраструктуру для совместной работы и обмена информацией между группами архитекторов, разработчиков, тестировщиков и менеджеров проекта. Клиентским компонентом является Team Foundation Client, который входит в состав всех продуктов семейства Team Edition, а также может использоваться продуктами Microsoft (Excel, Project) или продуктами других производителей.

Компоненты Team Foundation Server

Как видно из приведенной диаграммы, компоненты Team Foundation Server отвечают за управление требованиями, слежение за элементами проекта, обеспечивают автоматизацию сборки проекта, генерацию отчетов, а также интеграцию с другими подсистемами продуктов семейства Team System.

В состав Team Foundation входят средства создания проектов (Project Creation Wizard) с возможностью выбора шаблонов (по умолчанию используется шаблон MSF For Agile Development), задания типов артефактов, итераций, отчетов, контроля версий, адреса проектного портала и т. п., средства контроля версий, пришедшие на смену Visual SourceSafe и рас-

считанные на объемные проекты с большим числом разработчиков. В процессе работы над проектом собирается большой объем информации (для этих целей служит SQL Server 2005).

Эти данные могут быть отображены с помощью более чем 50 типов отчетов (на базе SQL Reporting Services), входящих в состав продукта (например, Work Item History — Record-Count, TransitionCount, RemainingWork, CompletedWork, BaselineWork, Code Churn — TotalLines, LinesAdded, LinesModified, LinesDeleted, Test Results — TotalTests, TestsFailed, AverageDuration и т. д.) и опубликованы на портале проекта (Project Portal на основе Windows SharePoint Services или Microsoft Office SharePoint Server 2007) через соответствующие веб-компоненты.

Проектная деятельность может вестись согласно определенным методологиям. В состав продукта входит поддержка двух методологий Microsoft — MSF For Agile Development и MSF For CMMI Process Improvement. Также поддерживается «классическая» методология Microsoft Solutions Framework. Помимо этого можно использовать методологии сторонних поставщиков или создавать собственные, отражающие подходы к созданию продуктов, сформированные внутри конкретной компании.

Visual Studio 2008 Team Foundation пополнился рядом новых возможностей, а также получил ряд улучшений в существующих компонентах.

Team Foundation Build — средство для управляемой сборки проектов. Теперь описания процесса сборки (Build Definitions) заменяют типы сборки (Build Types) из Microsoft Visual Studio 2005 Team System. В отличие от типов сборки, описания процесса сборки могут быть изменены в Team Explorer. Помимо этого из пользовательского интерфейса можно остановить выполняющуюся сборку и полностью удалить завершенный процесс. Также при создании нового описания процесса сборки или редактировании существующего можно установить триггер для процесса сборки. Можно использовать сборку по требованию, многократную сборку и интеграцию, при которой каждое освобождение (check-in) кода приводит к запуску процесса сборки. Также можно задать время ожидания между выполнениями сборок при многократных сборках. В Visual Studio 2008 Team Foundation Server появился ряд новых свойств для настройки процесса сборки, включая возможность настройки сборок для кода на C++. В состав Team Foundation Build входит ряд новых шаблонов, которые можно использовать для создания собственных процессов сборки.

Team Foundation Source Control — это входящее в состав Team Foundation средство контроля версий. В Visual Studio 2008 появились следующие возможности:

- Полное удаление файлов (команда Destroy), находящихся под управлением системы контроля версий.
- Автоматическое получение самой последней версии файла при выполнении операции check-out.
- Возможность аннотирования исходных файлов — построчный просмотр внесенных изменений, а также даты внесения этих изменений.
- Возможность сравнения двух папок на сервере, двух локальных папок или папки на сервере с папкой на локальном компьютере с использованием контроля версий. Пользователь получает информацию о различиях — отсутствующих файлах, файлах, в которые были внесены добавления, удаления, а также о конфликтах между файлами в разных папках.

Team Foundation Work Item Tracking — это механизм слежения за элементами проекта. В Visual Studio 2008 появились существенные улучшения при отслеживании элементов проекта под большими нагрузками. По сравнению с Team Foundation Server 2005, пропускная способность увеличилась вдвое, существенно снизилось время, требуемое для завершения отдельных операций и снизилось потребление процессорных ресурсов сервера, на котором располагается слой базы данных Team Foundation Server. Масштабируемость Team Foundation Server 2008 также была увеличена — так что время отклика для большинства операций слежения существенно сократилось, даже при высоких нагрузках на сервер. Это особенно заметно при работе проектных команд, состоящих из более чем 500 человек. Таким образом крупные компании могут использовать больше проектных элементов и следить за ними на существующих серверных конфигурациях, работающих под управлением Team Foundation Server 2008.

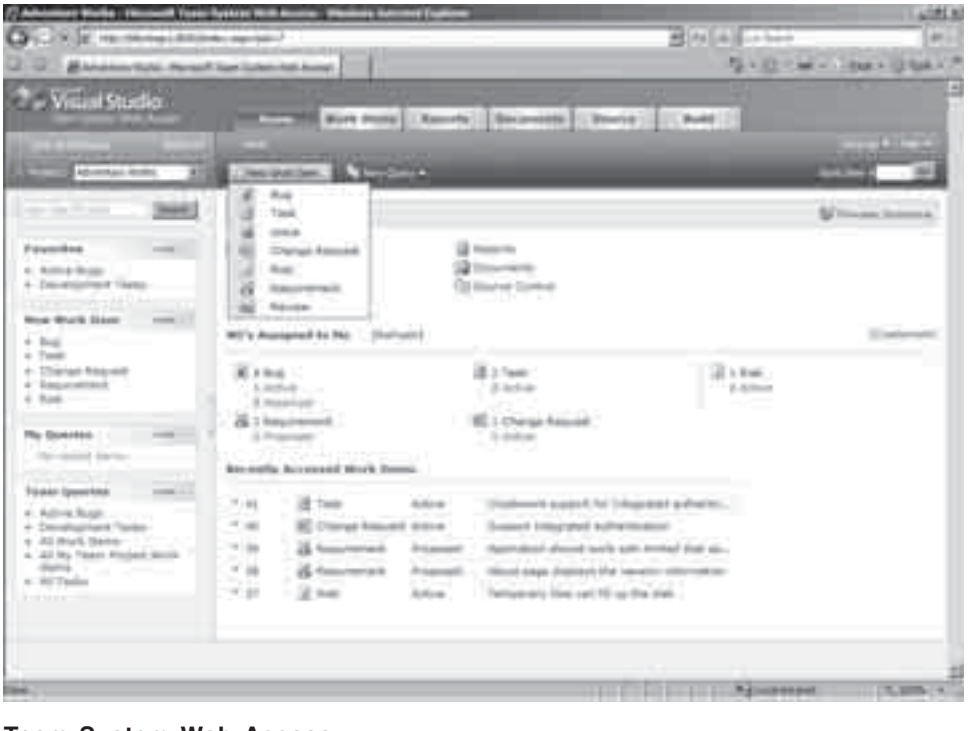
Управление Team Foundation Server — в новой версии сервера добавление большого числа пользователей не приводит к задержкам в работе и другим связанным с этим проблемам. Тогда как число поддерживаемых пользователей не изменилось, скорость синхронизации пользователей между Active Directory и Team Foundation Server существенно повысилась.

Как мы отметили выше, Team Foundation служит в качестве основы для работы различных участников проекта — менеджеров, архитекторов, разработчиков и тестировщиков, организации обмена информацией между ними. Для каждой конкретной роли — архитектора, разработчика, разработчика и администратора баз данных, тестировщика предлагается специализированная, ролевая версия Team System, которую мы и рассмотрим ниже.

Visual Studio Team System Web Access

Team System Web Access (продукт, ранее известный под названием TeamPlain Web Access и приобретенный у компании devBiz Business Solutions) представляет собой веб-интерфейс к Visual Studio 2005 Team Foundation Server. Team System Web Access доступен для бесплатной загрузки (http://go.microsoft.com/?linkid=7148426) всем пользователям Team Foundation Server и будет включен в Visual Studio 2008 Team System.

К ключевым характеристикам продукта относятся возможность добавления новых элементов программного проекта и редактирование существующих, использование любых типов элементов проекта, включая нестандартные, задаваемые пользователями, возможность добавления новых элементов в запросы и редактирование существующих запросов, наличие средств просмотра, загрузки, выгрузки, выполнения операций check-in и check-out над документами, расположенными на проектном портале на SharePoint Server, просмотр отчетов, экспорт отчетов в PDF или Excel, просмотр репозитариев системы контроля версий, загрузка файлов, просмотр наборов изменений (changesets), различий между файлами, истории изменений и представлений с аннотациями, возможность просмотра результатов сборки проекта, запуск и остановка процесса сборки, а также поиск по ключевым словам в элементах проекта.



Team System Web Access

Поддерживаются две модели аутентификации — Integrated Windows Authentication и Forms Based Authentication, последняя рекомендуется к использованию совместно с SSL.

Team System Web Access работает на Windows Server 2003, требует наличия на сервере Internet Information Services 6 с включенной поддержкой ASP.NET 2.0 и установленным Microsoft Visual Studio 2005 Team Explorer, на клиенте поддерживаются браузеры Internet Explorer 6+ и Mozilla Firefox 1+.

Microsoft Visual Studio 2005 Team Foundation Server Power Tool

Microsoft Visual Studio 2005 Team Foundation Server Power Tool (ранее назывался Power Toys) — это набор дополнений и утилит, улучшающих функциональность Team Foundation Server. В данный набор включены две новые пакетные утилиты для разработчиков и три утилиты с графическим интерфейсом — редактор шаблонов процессов, набор настраиваемых политик для операции check in и утилита, позволяющая выполнять модульные тесты.

Team Foundation Server Power Tool Commands (tfpt.exe) — это пакетная утилита, расширяющая функциональность Team Foundation Version Control. Некоторые команды, включенные в состав этой утилиты, имеют графический интерфейс. Помимо встроенных в утилиту команд, пользователям также доступны команды Annotate и Treediff, реализованные в Source Control Explorer (в Visual Studio) и Team Explorer.

В состав Team Foundation Server Power Tool включены следующие команды:

Команда	Описание
Unshelve	Используется для получения и объединения изменений в рамках workspace
Rollback	Используется для отмены изменений, внесенных на уровне Team Foundation Server
Online	Используется для создания изменений (pending edits) для файлов, у которых еще нет таких изменений
GetCS	Команда Get Changeset служит для получения изменений для определенного набора изменений (changeset)
UU	Команда Undo Unchanged служит для отмены операций над неизмененными файлами (unchanged files), включая операции добавления, редактирования и удаления
Annotate	Используется для загрузки всех версий указанных файлов и отображения построчной информации о том, кто и когда вносил изменения
Review	Данная команда используется для оптимизации процесса обзора кода путем отказа от использования операций check in и shelving

Команда	Описание
History	Служит для отображения истории ревизий для одного или более файлов или папок. Опция /followbranches позволяет получить историю для вложенных файлов (file branch)
Workitem	Используется для создания, обновления или просмотра элементов проекта
Query	Используется для запуска запроса к элементу проекта и отображения результатов выполнения этого запроса. Если не указан какой-то конкретный запрос, команда отобразит все элементы проекта, присвоенные данному пользователю
TreeDiff	Служит для визуального представления различий между файлами в двух серверных папках, в папке на сервере и локальной папке или двух локальных папках
Treeclean	Используется для отображения и опционального удаления файлов, которые не управляются системой контроля версий в текущем каталоге и подкаталогах. Эта команда может быть полезна в тех случаях, когда требуется удаление временных файлов из локальной области (workspace) — например файлов, созданных компилятором

Process Template Editor — при установке Team Foundation Server Power Tool в Microsoft Visual Studio Team System появляется Process Editor — интегрированное средство для создания новых типов элементов проекта и соответствующих компонентов шаблона процессов. Данная утилита требует установки компонентов Domain-Specific Language Tools for Visual Studio 2005 Redistributable Components, которые можно найти на сайте MSDN.

Check-In Policy Pack — это набор настраиваемых политик для операции check in, расширяющих стандартный набор, включенный в Team Foundation Server. В пакет включены следующие политики:

- Custom Path Policy — политика, предоставляющая механизм для указания маршрута к системе контроля версий, а также маршрутов, на которые распространяется данная политика. Данная политика поддерживает сценарий, когда можно применять различные наборы правил для различных папок системы контроля версий.
- Forbidden Patterns Policy — позволяет указать расширение имени файла или регулярное выражение, которые могут использоваться для указания на то, что определенные файлы не должны включаться в систему контроля версий. Это наиболее полезно для DLL-файлов, ряда файлов, порождаемых в процессе сборки проекта или автоматически создаваемых файлов для веб-сайтов, генерируемых в процессе разработки.
- Changeset Comments Policy — позволяет убедиться в том, что текстовое поле Comments в диалоговой панели Check In не является пустым —

если в даном текстовом поле отсутствует информация, файл не может быть включен в операцию check in.

- Work Item Query Policy — позволяет указать запрос, к которому относится тот или иной элемент проекта.

Test Tools Build Task (TestToolsTask) позволяет выполнять модульное тестирование простым указанием динамически загружаемых библиотек или указанием шаблона имени файла в файле TfsBuild.proj вместо использования файлов метаданных тестов (.vsmdi) для указания тестов, которые должны быть запущены. Данная функциональность аналогична запуску пакетной утилиты mstest.exe с опцией /testcontainer для указания тестового контейнера, все тесты из которого должны быть запущены.

«Ролевые» версии Visual Studio

В состав Visual Studio 2008 Team Suite входят четыре ролевые версии: для архитекторов — **Architecture Edition**, для разработчиков — **Development Edition**, для разработчиков баз данных — **Database Edition** и для тестировщиков — **Test Edition**.

Architecture Edition — Visual Studio для архитекторов

Продукт Team Architect Edition позволяет архитекторам программного обеспеченния и инфраструктуры создавать и описывать все компоненты сервис-ориентированных приложений. Работа архитекторов построена на основе использования специализированных дизайнеров:

- **Дизайнер распределенных приложений (Distributed Application Designer)** позволяет визуально создавать веб-сервисы и сервис-ориентированные архитектуры, обладает поддержкой расширений веб-сервисов (Web Services Enhancements, WSE) и возможностью проверки архитектуры на соответствие логической инфраструктуре. Включает следующие дизайнеры: Application Connection Designer, Logical Datacenter Designer, System Designer и Deployment Designer.
- **Дизайнер логической инфраструктуры (Logical Infrastructure Designer)** позволяет визуально описывать сетевую инфраструктуру, содержит редактор настроек и ограничений, включает генератор и компилятор System Definition Model (SDM)-модели, средства развертывания.
- **Дизайнер классов (Class Designer)** служит для визуального создания классов и генерации соответствующего кода.

Работая в связке, эти дизайнеры обеспечивают синхронизацию изменений с кодом, визуализацию изменений и т. п. операции. Важно то, что эти средства призваны пошагово решать задачу создания комплексных

систем и получать на выходе не только архитектуру, но и требования к инфраструктуре и общую модель системы.

На основе артефактов, созданных архитекторами, разработчики создают и тестируют код приложения. Для этих целей служит версия Team Edition для разработчиков.

В Visual Studio 2008 Team System Architecture Edition появился ряд дополнений и улучшений, которые мы рассмотрим ниже.

- Дизайн систем с использованием нисходящего подхода. Теперь у архитекторов появилась возможность использования нисходящего (top-down) подхода к дизайну приложений с использованием System Designer. Имеется возможность либо начать новый дизайн системы, либо продолжить работу над существующим — можно добавлять системы, приложения, точки входа непосредственно в описание системы. Точки входа добавляются прямо на границах описания системы и делегируют свое поведение членам системы.
- Синхронизация точек входа веб-сервисов с WSDL-файлами — теперь можно задавать или обновлять сигнатуры операций существующих веб-сервисов, написанных с использованием .NET, указав соответствующий WSDL-файл.
- Генерация проектов ASP.NET Web Application для приложений, использующих технологию ASP.NET за счет использования шаблона ASP.NET Web Application, который генерирует соответствующий тип проекта для приложения. Эта возможность доступна только для ASP.NET веб-проектов на основе файловой системы (свойство Project Location Type имеет значение File System). Также имеется возможность ре-инжиниринга ASP.NET веб-приложений.
- Сохранение, импорт и экспорт дополнительных прототипов — у архитекторов появилась возможность сохранять или устанавливать дополнительные прототипы, импортируя их, а не редактируя в реестр. Отметим, что редактирование реестра по-прежнему требуется для установки файлов .sdmdocument для прототипов приложений, созданных средствами System Definition Model (SDM) SDK.

В августе 2007 года была выпущена предварительная версия (Community Technology Preview) пакета утилит — Team Architect Power Tools, расширяющих функциональность Team System Architecture Edition путем добавления средств работы с библиотеками классов на уровне диаграмм приложений и системы. Этот пакет можно бесплатно загрузить с сайта MSDN.

Development Edition — Visual Studio для разработчиков

Помимо традиционных средств для разработки приложений, обеспечиваемых на уровне функциональности Visual Studio Professional Edition (написание и редактирование кода, компиляция, отладка, развертывание), в версию Team Edition для разработчиков входит ряд существенных дополнений, среди которых:

- **Средства статического анализа кода** выполняют нечто схожее с грамматическим анализом кода. Компилятор выполняет проверку синтаксиса и убеждается в том, что объекты, параметры и другие элементы кода допустимы для данного языка программирования. Средства статического анализа кода проверяют семантику и нарушения безопасности, стиля кодирования, руководств по производительности. Для проверки кода на C/C++ Microsoft предлагает PREFast — средство, работающее с .NET и обычным кодом, а также средство FxCop, которое базируется на наборе рекомендаций .NET Class Design Guidelines.
- **Средства профилирования** для того, чтобы получить максимальную производительность кода. Microsoft предлагает ряд профилировщиков кода, входящих в состав Visual Studio Team System и интегрированных в среду разработки. Доступны два типа профилировщиков:
  - ☐ Профилировщик на основе «проб» (Sampling profiler) — отслеживает выполнение приложения и по окончании сообщает данные о производительности.
  - ☐ Инструментальный профилировщик (Instrumented profiler) — во время компиляции вставляет в код определенные «метки» и сохраняет данные в хранилище Team System при достижении приложением определенной «метки».
- **Средства модульного тестирования** — модульное тестирование на основе атрибутов стало популярным способом проверки качества кода. В состав Visual Studio Team System входят интегрированные средства создания, слежения и выполнения модульных тестов непосредственно из среды разработчика. Модульное тестирование предполагает, что для каждого метода класса пишется определенный тест. Такой тест проверяет все возможные способы поведения класса (в соответствии со спецификацией) и сообщает об обнаруженных ошибках. Модульные тесты крайне полезны при регрессивном тестировании, т. к. они могут выполняться над каждым классом в проекте для того, чтобы убедиться, что изменения в одном классе не привели к изменениям в других.

В Visual Studio 2008 Team System Development Edition появился ряд дополнений и улучшений, которые мы рассмотрим ниже.

- **Метрики кода (Code Metrics)** — набор программных измерений, позволяющих разработчикам «заглянуть» в создаваемый ими код. Используя метрики кода, разработчики могут понять, какие типы и/или методы должны быть переработаны или дополнительно оттестированы. Помимо этого, команды разработчиков могут идентифицировать потенциальные риски, получать представление о текущем состоянии проекта и отслеживать прогресс в процессе работы над проектом.
- **Средства профилирования** пополнились рядом дополнительных возможностей, среди которых:
  - ☐ Поддержка Windows Vista — поддерживается полное соответствие требованиям безопасности Windows Vista, доступ к IIS по-прежнему требует прав администратора.
  - ☐ Сбор данных о распределении стека — включается на странице свойств Performance Session, отображается в отчете Allocation View.
  - ☐ Данные проб на уровне строк кода — включено по умолчанию, отображается в отчетах IP и Line View, информация о строках — в отчете Module.
  - ☐ Управление в режиме выполнения — автоматический запуск при запуске средства профилирования, возможность включения и отключения сбора данных из пользовательского интерфейса, запуск с отключенным сбором данных, добавление дополнительных пометок без редактирования кода.
  - ☐ Фильтрация аналитических данных — фильтрация по отметкам времени, процессам, потокам и пометкам. Функциональность, схожая с системой запросов в модуле слежения за элементами проекта (опция /filterfile позволяет сохранить фильтр в файле).
  - ☐ Поиск расхождений (Diff) — возможность управления отображаемыми данными, выбор двух отчетов в Performance Explorer и включение режима сравнения простым щелчком мышью, новый отчет о расхождениях в меню Tools, опция командной строки /diff.
  - ☐ Поддержка счетчиков производительности Windows — возможность фильтрации данных счетчиков, ассоциация со свойствами сессии подсчета производительности, опция командной строки /wincounter.
  - ☐ Компактные файлы отчетов — создание файлов отчетов небольшого размера на основе полных отчетов, команда Save Analyzed или опция командной строки /summaryfile.
  - ☐ Копирование в HTML — возможность копирования отчета и прикрепления его к электронному письму.

- Интеграция с нагрузочным и веб-тестированием — создание нагрузочного или веб-теста, создание сессии измерения производительности прямо из теста.
- Поддержка приложений, использующих технологию Windows Communication Foundation в средствах профилирования.

Следующее издание Visual Studio Team System — это издание, предназначенное для работы с базами данных.

Database Edition — Visual Studio для разработчиков и администраторов баз данных

Новым компонентом в Visual Studio Team System 2008 является издание, предназначенное для разработчиков и администраторов баз данных — Visual Studio Team System 2008 Database Edition. Это издание было доступно с ноября 2006 года в виде отдельного продукта для Visual Studio 2005 под названием Team Edition for Database Professionals и теперь этот компонент включен в состав Visual Studio Team System 2008.

К основным функциям версии для разработчиков баз данных можно отнести управление изменениями в схеме базы данных, модульное тестирование схемы базы данных, а также возможность полноценной интеграции в весь процесс разработки программного продукта с использованием платформы Visual Studio Team System.

В рамках нового типа проекта (Visual Studio Database Project) у разработчиков появляется возможность импортировать схему базы данных из SQL Server 2000 и SQL Server 2005 (версия SQL Server 2008 будет поддерживаться в следующей версии Visual Studio Team System, которая имеет кодовое название «Rosario») и поместить ее в систему контроля версий. Когда требуется внедрение изменений в схеме базы данных, новая система поддержки проектов позволяет создать скрипты обновления (update scripts) и предоставляет механизм для выполнения этих скриптов в выбранной базе данных.

Основной концепцией Visual Studio Team System 2008 Database Edition является работа с базами данных в отсоединенном, т. н. оффлайновом режиме. Проект представляет собой логически собранный в коллекцию набор файлов, которые содержат схему базы данных и ее различные представления. Проект может работать с любым SCCI-провайдером (например Team Foundation Server, VSS).

Новый **редактор T-SQL**, входящий в состав Visual Studio, существенно повышает продуктивность при написания кода и обеспечивает поддерж-

ку параллельного выполнения запросов и отображение клиентской статистики.

Механизм **Rename Refactoring** позволяет быстро переименовать объекты базы данных, включая все ссылки на эти объекты и все зависимые объекты (схема, генераторы данных, Unit Test, SQL-скрипты и т. п.). Обычно такие изменения называют «каскадными обновлениями».

Механизм **Schema Compare** позволяет быстро сравнить схемы двух баз данных (или одной базы данных — в системе контроля версий и в самой базе данных) и создать скрипты обновления для синхронизации схем. Схожий с описанным выше механизм Data Compare позволяет быстро сравнить данные в двух базах данных и создать скрипты обновления для синхронизации данных. Оба механизма генерации скриптов поддерживают выполнение как непосредственно из среды Visual Studio, так и из командной строки (SQLCMD), SQL Server Management Studio или в рамках исполняемого файла.

Механизм **Data Generator** служит для создания плана генерации повторяющихся наборов «осмысленных» данных на основе реальных данных, взятых из базы данных с поддержкой гистограмм и распределений. Такие данные используются для заполнения базы при выполнении модульных тестов или нагрузочного тестирования. Механизм генерации данных поддерживает модули расширения — т. о. есть возможность создания и подключения собственных генераторов.

И, наконец, поддержка **модульного тестирования базы данных** (Database Unit Testing) на основе Team Test Unit Test позволяет создавать тесты, используя язык T-SQL или управляемый код (Visual Basic .NET или C#). Можно тестировать следующие объекты баз данных: хранимые процедуры, функции, триггеры, любой код на языке T-SQL. Также имеется возможность тестирования базы данных на уровне всего разрабатываемого приложения.

Ролевой подход

При работе с Visual Studio Team System 2008 Database Edition предполагается использование двух ролей — администратора базы данных и разработчика базы данных. Функции по фазам работы над проектом — администрирование, разработка, внедрение — распределяются следующим образом.

На первом этапе — администрировании — администратор базы данных выполняет следующие действия:

- Создает в Visual Studio новый проект базы данных.
- Получает схему из базы данных.
- Создает план генерации данных.

Затем наступает этап разработки, на котором разработчик базы данных:

- Разрабатывает тесты.
- Пишет код базы данных и объектов.

- Выполняет рефакторинг.
- Запускает тесты.
- Вносит код и тесты в систему управления версиями.

На финальной стадии — внедрении — снова задействуется администратор базы данных, который:

- Проверяет внесенные изменения.
- Сравнивает модификации с рабочей базой данных.
- Создает скрипты для внедрения (скрипты для изменения схемы базы данных или самих данных).
- Внедряет скрипты в рабочую базу данных.

Выпущенный в июле 2007 пакет обновлений — **Team Edition for Database Professionals SR1**, в котором реализована поддержка ссылок на базы данных в рамках проекта (cross-database reference) и ссылок на метафайлы баз данных (.dbmeta). Помимо этого улучшена поддержка файлов в файловых группах SQL Server на уровне свойств проекта и к свойствам базы данных добавлена страница Variables, на которой можно задавать переменные типа setvar, используемые в скриптах развертывания. И, наконец, данный пакет обновлений полностью поддерживает пакет обновлений для SQL Server 2005 — Service Pack 2, и работу под управлением операционной системы Windows Vista.

Test Edition — Visual Studio для тестировщиков

В состав издания Visual Studio Team System для тестировщиков входят средства для нагрузочного тестирования и измерения производительности веб-приложений. Важно отметить то, что средства нагрузочного тестирования входят непосредственно в средство разработки и тесно интегрированы как со средой, так и с механизмами сбора данных и создания отчетов.

Запись и воспроизведение тестов

Такой подход позволяет обеспечить максимально тесное взаимодействие разработчиков и тестировщиков — при использовании Visual Studio 2005 Team System больше нет необходимости в обмене списком ошибок и последующим слежением за их исправлением — все данные (описания ошибок с необходимым контекстом) доступны из одной среды и соответствующие работы (Work Items) назначаются непосредственным исполнителям.

В Visual Studio Team System появилась возможность записи действий тестировщиков при работе с веб-приложениями и последующего воспроизведения таких записей. Такие записанные тесты хранятся в виде XML-файлов — они могут быть сохранены в системе контроля версий вместе с кодом приложения и другими артефактами периода разработки.

Нагрузочное тестирование

После того как тест веб-приложения записан и сохранен, он может быть воспроизведен на инфраструктуре веб-фермы с соответствующими нагрузочными данными. Условия загрузки настраиваются с помощью мастера a Load Test Creation Wizard и сохраняются в виде XML-файлов — они также могут быть сохранены в системе контроля версий вместе с кодом приложения и другими артефактами периода разработки.

Все нагрузочные тесты создаются и выполняются непосредственно из среды разработки Visual Studio. Это позволяет тестировщикам и разработчикам обмениваться информацией, используя одно и то же средство, доступное любой роли.

Тестировщики могут просматривать результаты тестов и создавать отчеты об ошибках на основе данных, полученных при нагрузочном тестировании. Такие отчеты содержат поля для разработчиков, позволяющие в точности повторить обнаруженную проблему, а также информацию, собранную в процессе выполнения приложения. Таким образом, разработчики не только видят отчет, но и получают возможность проанализировать данные, вызвавшие ошибку.

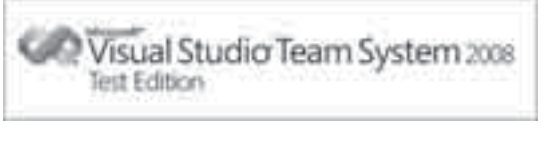
Помимо этого, как мы отметили выше, Visual Studio Team System поддерживает механизмы модульного тестирования, которое может выполняться и в нагруженном состоянии.

Управление тестированием

Все тесты, создаваемые в Visual Studio Team System, включая модульные тесты, воспроизводимые веб-тесты и нагрузочные тесты, управляются из окна Test Manager среды разработчика. Это позволяет разработчикам, тестировщикам и менеджерам продуктов получать информацию о статусе и полноте выполнения всех тестов для данного программного проекта. Помимо этого, все тесты могут быть сохранены в системе контроля версий Visual Studio Team Foundation Server.

Окно Test Manager позволяет создавать так называемые «списки тестов», которые полезны в том случае, когда из всего набора тестов для данного программного проекта при получении новой версии необходимо выполнить только часть тестов. Поддерживается возможность отображения тестов по «владельцам», типам и другим характеристикам.

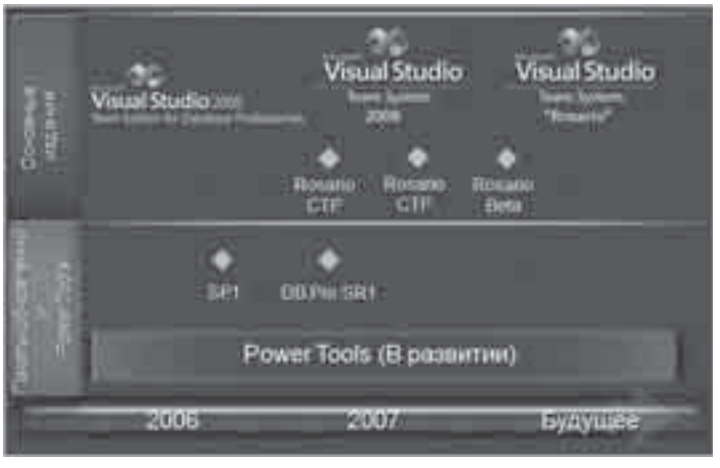
В Visual Studio 2008 Team System Test Edition появился ряд дополнений и улучшений, которые мы рассмотрим ниже.



- Возможность создания и выполнения модульного тестирования в Visual Studio Professional Edition.
- Упрощенное выполнение модульных тестов — поддерживается возможность запуска тестов непосредственно из кода.
- Использование наследования между тестовыми классами.
- Выполнение модульного тестирования на устройствах, разработка для которых поддерживается в рамках Visual Studio.
- Создание новых хост-адаптеров, позволяющих выполнять тесты в средах, отличных от сред тестирования по умолчанию.
- Улучшенная связь с источниками данных для модульного тестирования, включая CSV-файлы, XML-файлы и т. д.
- Улучшенная связь с источниками данных в веб-тестах.
- Улучшенные правила проверки результатов тестов при веб-тестировании — более гибкая система применения правил и использования результатов для управления выполнением веб-тестов.
- Более гибкое управление нагрузочным тестированием и моделированием нагрузок.
- Улучшенные средства просмотра результатов нагрузочного тестирования, включая новые диаграммы и представления.
- Улучшенные средства управления репозитарием результатов нагрузочного тестирования.
- Сохранение результатов тестирования в виде XML-файлов со схемой. Возможность программной обработки результатов тестирования, автоматически сохраняемых в XML-формате — файлы с расширением .trx (Test Result XML).

Visual Studio Team System в развитии — проект «Rosario»

Относительно недавно компания Microsoft раскрыла планы по развитию семейства продуктов Visual Studio Team System и предоставила доступ к ранней версии следующего поколения продуктов этого семейства, известного под кодовым названием «Rosario».



Планы по развитию Visual Studio Team System

Microsoft Visual Studio Team System «Rosario» представляет собой интегрированное средство для управления всем жизненным циклом приложения (Application Life-cycle Management, ALM) и состоит из инструментов, процессов и руководств по их применению. При разработке новой версии Visual Studio Team System Microsoft решает две ключевых бизнес-задачи: синхронизацию разработки приложений с требованиями бизнеса и улучшение качества приложений и их тестирования. К средствам, поддерживающим решение первой задачи, относятся:

- Средства планирования портфеля проектов — интеграция с Portfolio Server через Project Server.
- Проектное планирование — интеграция с Project Server и упрощенные средства управления расписаниями.
- Средства сбора требований и их проверки.
- Отслеживание требований за счет поддержки иерархических связей и улучшенной системы запросов.
- Улучшенная система генерации отчетов.

Для решения второй задачи — улучшения качества приложений и их тестирования — планируются следующие улучшения в продукте:

- Управление сценариями тестирования.
- Выполнение тестов — как функциональных, так и ручных.
- Интеграция средств тестирования и диагностики для облегчения воспроизведения ошибок.
- Приоритет тестов — возможность определения наиболее важных для выполнения тестов.

Часть из планируемых расширений и улучшений уже включена в предварительную версию продукта. Их использование позволит проектной команде:

- Более эффективно общаться между собой и заказчиками, а также выполнять совместные действия в рамках работы над программным проектом.
- Обеспечивать ожидаемое качество разрабатываемого программного обеспечения, используя соответствующие средства на каждом шаге создания продукта.
- Получать детальную информацию об активностях в рамках проекта и приоритетах на основе данных, которые позволят всем членам команды принимать обоснованные решения.

В версии, выпущенной для предварительного ознакомления в августе 2007 года, реализована поддержка нескольких сценариев работы над программным проектом:

- Создание расписания разработки и его отслеживание. Проектные команды могут более эффективно обмениваться задачами и отчетами о прогрессе в рамках проекта, используя механизмы слежения за элементами проекта, включая возможность создания дочерних соотношений между элементами проекта, определения зависимостей задач и отслеживания прогресса работы над требованиями (от определения до реализации) через актуальные отчеты и таблицы. Это возможно либо через Visual Studio Team Explorer, либо за счет более тесной интеграции с Microsoft Excel.
- Тестирование требований и ручное тестирование. Возможность определения соотношения между сценариями тестирования и требованиями к программному продукту дает тестировщикам возможность обнаружить пропущенные сценарии еще на ранних стадиях проекта, а также более точно определить причины неудачного выполнения тестовых сценариев. При выполнении ручных тестов, используя утилиты Manual Test Runner, позволяющей выполнять тестирование вне интегрированной среды, тестировщикам предоставляется возможность включения в отчеты большей информации об обнаруженных ошибках. С помощью Manual Test Runner можно загружать тесты с сервера Team Foundation Server, публиковать результаты тестирования, включать в отчеты копии экранов, автоматически собирать системную информацию, показывать шаги для воспроизведения ошибки, а также записывать видеоролики, показывающие эти шаги.
- Управление зависимостями. Разработчикам предоставляется возможность более простого определения зависимостей, которые основываются на нескольких задачах и, таким образом, упростить расстановку приоритетов для всех членов проектной группы.

Рассмотренные выше сценарии создания расписания разработки и отслеживания этого расписания, тестирования требований и ручного тестирования, а также управления зависимостями показывают ряд направлений развития Visual Studio Team System «Rosario». Новая функциональность позволяет более быстро создавать качественные приложения любого класса, более эффективно работать в команде вне зависимости от сложности проекта или размера проектной группы.

Механизмы расширения Visual Studio

Ниже мы кратко рассмотрим основные механизмы, используемые для расширения функциональности среды разработчика Microsoft Visual Studio. Интегрированная среда разработчика поддерживает различные способы создания расширений — от простых макросов, которые могут быть связаны либо с определенными комбинациями клавиш, либо с кнопками на панели инструментов, и дополнительных модулей (Add-Ins), позволяющих добавлять к среде новую функциональность, до механизмов VS Package, используя которые можно встраивать в среду разработчика целые программные продукты или существенно изменять или расширять функциональность среды.

Ниже мы рассмотрим каждый из трех доступных механизмов расширения функциональности интегрированной среды разработчика Visual Studio.

Макросы

Использование макросов — самый простой способ создания расширений для Visual Studio. Запись макроса позволяет сохранить последовательность использования практически всех команд среды и комбинаций клавиш. Помимо этого, механизм макросов дает вам доступ к объектной модели управления Visual Studio (Visual Studio Automation Object Model), содержащей более 140 различных объектов, а также доступ к классам .NET Framework. Несмотря на свою гибкость, макросы имеют ряд ограничений — для их написания используется язык Visual Basic, они не могут использоваться для создания новых окон (Tool Window), команд или страниц настройки опций (Tools Options).

Дополнительные модули

Создание дополнительных модулей (Add-Ins) — это второй механизм расширения функциональности Visual Studio. Дополнительные модули предоставляют возможность полного управления Visual Studio через Automation Object Model, что существенно облегчает взаимодействие с большинством инструментов и функций среды разработчика, включая Text Editor, Output Window, Task List и Code Model. Для создания модулей расширения в Visual Studio входит специальный «мастер» — Add-In Solution Wizard.

В отличие от макросов, модули расширения могут быть написаны на любом языке программирования, поддерживающем технологию COM. К таким языкам, в частности, относятся Visual C#, VB.Net и Visual C++. Среди других отличий модулей расширения от макросов отметим то, что модули сохраняются в откомпилированных DLL, что обеспечивает большую гибкость при распространении таких модулей, а также тот факт, что модули расширения могут использоваться для создания новых команд, страниц опций и окон инструментов (Tool Window). С другой стороны, используя модули расширения, нельзя создавать новые типы документов, новые типы проектов, новые средства отладки и т. п. — для решения этих задач используются т. н. VSIP-пакеты.

Механизмы VS Package

Механизмы VSIP-пакетов обеспечивают максимальную гибкость при решении задач, связанных с расширением функциональности среды разработчика в Visual Studio. VSIP-пакеты обеспечивают доступ к ряду программных интерфейсов (т. н. VSIP-интерфейсы), используя которые можно получить практически безграничный контроль над средой разработчика и интегрировать в Visual Studio практически любое программное обеспечение. К возможностям, предоставляемым VSIP-интерфейсами, относятся создание новых типов проектов, новых отладчиков, редакторов, расширений дизайнеров и новых языков программирования. Одним из примеров использования интерфейсов VSIP для интеграции нового языка программирования является IronPython — подробности интеграции описаны в Visual Studio SDK.

Начиная с Visual Studio 2005, у разработчиков появилась возможность использования библиотеки Managed Package Framework для более простого создания VSIP-пакетов на управляемых языках программирования.

Расширения Visual Studio Team System

Помимо рассмотренных выше механизмов, которые доступны во всех версиях Visual Studio за исключением версии Express, в Visual Studio Team System существует ряд дополнительных возможностей. Среди них — возможность расширения шаблонов процессов, расширение «мастера» создания проектов, расширение типов элементов работы (Work Items), расширение типов тестов, настройки системы построения проектов (Build System), а также управление системой контроля версий через Version Control Object Model и расширение механизма формирования отчетов.

Visual Studio 2008 Shell

Visual Studio 2008 Shell — это максимально «облегченная» версия Visual Studio, которую можно бесплатно использовать для создания собственных

средств разработки. Visual Studio 2008 Shell предоставляет основу, на базе которой можно реализовывать различные дизайнеры, средства проектирования, разработки, собственные языки программирования — пользователи таких средств будут работать в уже знакомой им по опыту использования Visual Studio среде.

Среди преимуществ использования Visual Studio Shell отметим быструю разработку за счет наличия среды, позволяющей включать в нее дополнительные инструментальные средства и языки программирования, знакомую среду программирования, использование которой существенно сокращает время на освоение и изучение, а также оптимизацию для хостинга языков программирования и различных программных инструментов.

Visual Studio Shell можно использовать в двух режимах — интегрированном и изолированном. В первом режиме расширения, созданные для Visual Studio Shell, объединяются (интегрируются) с любым другим изданием Visual Studio, установленным на компьютере, во втором — приложения работают параллельно с другими изданиями Visual Studio и изолированы от них. Интегрированный режим представляет интерес, в первую очередь, для компаний, создающих средства, расширяющие функциональность Visual Studio, тогда как изолированный режим — это отличное средство для компаний, создающих собственные инструменты.

Visual Studio 2008 Shell доступна в составе VS 2008 SDK, который можно загрузить с сайта MSDN. Более подробную информацию о механизмах расширения Visual Studio, включая Visual Studio 2008 Shell, можно получить в блоге http://blogs.msdn.com/vsxteam/.

Visual Studio Team System. Полезные ссылки

Общие ресурсы

- Российский раздел сайта Microsoft Developer Network (MSDN)
  - http://msdn2.microsoft.com/ru-ru/default.aspx
- Раздел сайта MSDN, посвященный Visual Studio Team System
  - http://msdn2.microsoft.com/teamsystem/

Team Foundation Server

- Team Foundation Server Team Center
  - http://msdn2.microsoft.com/en-us/teamsystem/aa718934

- Руководство «Team Development with Visual Studio Team Foundation»
  - http://www.codeplex.com/tfsguide
- Visual Studio Team Foundation Server — Project Server 2007 Connector
  - http://www.codeplex.com/pstfsconnector
- Microsoft Visual Studio 2005 Team Foundation Server Power Tool
  - http://go.microsoft.com/?linkid=5422499
- Project Manager Team Center
  - http://msdn2.microsoft.com/en-us/teamsystem/aa718827
- Ответы на часто задаваемые вопросы по Visual Studio Team System
  - http://msdn2.microsoft.com/en-us/teamsystem/aa718930.aspx

Для архитекторов

- Software Architect Team Center
  - http://msdn2.microsoft.com/en-us/teamsystem/aa718756
- Ответы на часто задаваемые вопросы по Team Edition for Software Architects
  - http://msdn2.microsoft.com/en-us/teamsystem/aa718909.aspx

Для разработчиков

- Software Developer Team Center
  - http://msdn2.microsoft.com/en-us/teamsystem/aa718765
- Ответы на часто задаваемые вопросы по Visual Studio 2005 Team Edition for Software Developers
  - http://msdn2.microsoft.com/en-us/teamsystem/aa718914.aspx
- Class Designer PowerToys
  - http://www.gotdotnet.com/workspaces/workspace.aspx?id=fe72608b-2b28-4cc1-9866-ea6f805f45f3

Для разработчиков и администраторов баз данных

- Database Developers Team Center
  - http://msdn2.microsoft.com/en-us/teamsystem/aa718764
- Ответы на часто задаваемые вопросы по Visual Studio Team Edition for Database Professionals
  - http://msdn2.microsoft.com/en-us/teamsystem/aa718913.aspx
- Visual Studio 2005 Team Edition for Database Professionals Power Tools
  - http://go.microsoft.com/?linkid=7244111

Для тестировщиков

- Software Tester Team Center
  - http://msdn2.microsoft.com/en-us/teamsystem/aa718941
- Ответы на часто задаваемые вопросы по Visual Studio 2005 Team Edition for Software Testers
  - http://msdn2.microsoft.com/en-us/teamsystem/aa718931.aspx
- patterns & practices Performance Testing Guidance
  - http://www.codeplex.com/perftesting

Visual Studio 2008

- http://msdn2.microsoft.com/en-us/vstudio/aa700830.aspx

Visual Studio «Rosario»

- http://msdn2.microsoft.com/en-us/teamsystem/bb725993.aspx

Механизмы расширения Visual Studio

- Visual Studio Extensibility Developer Center
  - http://msdn2.microsoft.com/en-us/vstudio/aa700819.aspx
- VSX Team Blog
  - http://blogs.msdn.com/vsxteam/
- Domain-Specific Language Tools Developer Center
  - http://msdn2.microsoft.com/en-us/vstudio/aa718368.aspx
- Visual Studio Industry Partner (VSIP) Program
  - http://msdn2.microsoft.com/en-us/vstudio/aa700860.aspx
- Visual Studio 2008 Shell
  - http://msdn2.microsoft.com/en-us/vstudio/bb510103
  - http://msdn.microsoft.com/vstudio/shell/

Партнерские решения

- Teamlook — организация эффективной командной работы на основе Outlook http://www.personifydesign.com/products/teamlook/teamlook.aspx
- TeamSpec — управление требованиями http://www.personifydesign.com/products/teamspec/teamspec.aspx
- Интеграция с Eclipse
  - http://www.teamprise.com/
- Бизнес-моделирование

- ☐ http://www.ravenflow.com/

- Project Server 2007 VSTS Connector
  - ☐ http://www.codeplex.com/pstfsconnector
- Requirements Authoring Starter Kit — решение по управлению требованиями на основе TFS в исходных кодах
  - ☐ http://www.microsoft.com/down-loads/details.aspx?FamilyID=E96CCC54-8759-452F-BF68-3A261C663 B66 &displaylang=en
- Шаблон процесса SCRUM для TFS
  - ☐ http://www.scrumforteamsystem.com/en/default.aspx

Приложение.

Аппаратные и программные требования

Издание Microsoft Visual Studio 2008 Team System Team Foundation Server представляет собой многозвенное приложение, компоненты которого могут быть установлены на различных компьютерах. Для установки Visual Studio Team Foundation Server на одном компьютере вам потребуется машина со следующими характеристиками:

- Процессор с частотой не ниже 2.2 ГГц;
- Минимум 2 Гбайт оперативной памяти;
- 8 Гбайт свободного пространства на жестком диске со скоростью не менее 5400 об/мин;
- Дисплей с разрешением не менее 1024 x 768;
- Привод DVD.

Пакет Visual Studio Team Foundation Server может быть установлен на следующие операционные системы:

- Windows Vista (x86 и x64) — все издания за исключением издания Starter Edition;
- Windows XP (x86 и x64) с установленным пакетом обновлений Service Pack 2 или более поздней версии — все издания за исключением издания Starter Edition;
- Windows Server 2003 (x86 и x64) с установленным пакетом обновлений Service Pack 1 или более поздней версии;
- Windows Server 2003 R2 (x86 или x64) или более поздней версии;
- Windows Server 2008 (x86 и x64).

Ниже мы более подробно рассмотрим аппаратные и программные требования для каждого компонента Visual Studio Team System.

Компонент	Программные требования	Аппаратные требования
Клиентские компоненты	32-битная операционная система: · Windows XP Professional/ Service Pack 2 · Windows Server 2003 · Windows Vista 64-битная операционная система: · Microsoft Windows Server 2003/SP1, Standard x64 Edition (режим WOW) · Microsoft Windows Server 2003/SP1, Enterprise x64 Edition (режим WOW) · Microsoft Windows Server 2003/SP1, Datacenter x64 Edition (режим WOW)	Минимум: процессор: 2.0 GHz, 512 MB памяти, 8 GB HDD. Рекомендуется: процессор: 2.6 GHz, 1 GB памяти, 20 GB HDD

Компонент	Программные требования	Аппаратные требования
	· Microsoft Windows Server 2003 R2, Standard x64 Edition (режим WOW) · Microsoft Windows Server 2003 R2, Enterprise x64 Edition (режим WOW) · Microsoft Windows Server 2003 R2, Datacenter x64 Edition (режим WOW) · Microsoft Windows XP Professional x64 Edition (режим WOW) · Windows Vista Требуемые программные компоненты · Internet Explorer 6.0/ Service Pack 1 · Microsoft Office 2003/Service Pack 1 или · Microsoft Office 2007 · MDAC 9.0 · .NET Framework 2.0	
TFS Слой данных	32-битная операционная система: · Windows Server 2003, Standard Edition/ Service Pack 1 (SP1) · Microsoft Windows Server 2003 R2 Standard Edition · Windows Server 2003, Enterprise Edition/ Service Pack 1 (SP1) · Microsoft Windows Server 2003 R2 Enterprise Edition · Windows Server 2003 Datacenter Edition/ Service Pack 1 (SP1) · Microsoft Windows Server 2003 R2 Datacenter Edition · Windows Server 2008 TFS Слой данных не поддерживает 64-битные операционные системы при развертывании на одном сервере с прикладным слоем 64-битная операционная система: · Microsoft Windows Server 2003/Service Pack 1 (SP1), Standard x64 Edition · Microsoft Windows Server 2003/Service Pack 1 (SP1), Enterprise x64 Edition · Microsoft Windows Server 2003/ Service Pack 1 (SP1), Datacenter x64 Edition · Microsoft Windows Server 2003 R2, Standard x64 Edition · Microsoft Windows Server 2003 R2, Enterprise x64 Edition Microsoft Windows Server 2003 R2, Datacenter x64 Edition	Минимум: процессор: 2.2 GHz, 8 GB HDD, 1 GB памяти. Сервер должен обладать надежным сетевым соединением с минимальной скоростью 1 Mbps и максимальной задержкой в 350 ms. Рекомендованная конфигурация: см. следующий раздел

Компонент	Программные требования	Аппаратные требования
TFS Прикладной слой	32-битная операционная система: · Windows Server 2003 Standard Edition/ Service Pack 1 (SP1) · Microsoft Windows Server 2003 R2 Standard Edition · Windows Server 2003 Enterprise Edition/ Service Pack 1 (SP1) · Microsoft Windows Server 2003 R2 Enterprise Edition · Windows Server 2003 Datacenter Edition/ Service Pack 1 (SP1) · Microsoft Windows Server 2003 R2 Datacenter Edition · Windows Server 2008 TFS Прикладной слой не поддерживает 64-битные операционные системы	Минимум: процессор: 2.2 GHz, 8 GB HDD, 1 GB памяти. Сервер должен обладать надежным сетевым соединением с минимальной скоростью 1 Mbps и максимальной задержкой в 350 ms. Рекомендованная конфигурация: см. следующий раздел
Team Build	32-битная операционная система: · Windows Server 2003/ Service Pack 1 (SP1), Service Pack 2 (SP2) · Windows Server 2003 R2 · Windows XP Professional, .NET 2.0 64-битная операционная система: · Microsoft Windows Server 2003/SP1, Standard x64 Edition (режим WOW) · Microsoft Windows Server 2003/SP1, Enterprise x64 Edition (режим WOW) · Microsoft Windows Server 2003/SP1, Datacenter x64 Edition (режим WOW) · Microsoft Windows Server 2003 R2, Standard x64 Edition (режим WOW) · Microsoft Windows Server 2003 R2, Enterprise x64 Edition (режим WOW) · Microsoft Windows Server 2003 R2, Datacenter x64 Edition (режим WOW) Microsoft Windows XP Professional x64 Edition (режим WOW)	Минимум: процессор: 766 GHz, 8GB HDD, 256MB памяти. Рекомендованная конфигурация: см. следующий раздел
TFS Proxy	32-битная операционная система: · Windows Server 2003 Standard Edition/ Service Pack 1 (SP1) · Microsoft Windows Server 2003 R2 Standard Edition · Windows Server 2003 Enterprise Edition/Service Pack 1 (SP1) · Microsoft Windows Server 2003 R2 Enterprise Edition · Windows Server 2003 Datacenter Edition/Service Pack 1 (SP1)	Минимум: процессор: 2 GHz, 40 GB HDD, 2 GB памяти. Сервер должен обладать надежным сетевым соединением с минимальной скоростью 1 Mbps и максимальной задержкой в 350 ms.

Компонент	Программные требования	Аппаратные требования
	• Microsoft Windows Server 2003 R2 Datacenter Edition • Windows Server 2008	
Удаленное тестирование	TFS Proxy не поддерживает 64-битные операционные системы Windows Server 2003/ Service Pack 1 (SP1), Windows XP Profession/ Service Pack 2 или Windows 2000/Service Pack 4, SQL Server 2005 Express Edition, .NET Framework 2.0	Процессор: 600 MHz, 1 GB HDD, 512 MB памяти. Рекомендованная конфигурация: см. следующий раздел

Рекомендованные аппаратные конфигурации

Team Foundation Server. Прикладной слой

Конфигурация	CPU	HD	Память	Комментарии
Менее 20 пользователей	2.2 GHz	8 GB	1 GB	Установка на том же компьютере, что и слой данных (один сервер)
От 20 до 50 пользователей	2.2 GHz	30 GB	1 GB	Установка на том же компьютере, что и слой данных (один сервер)
От 50 до 100 пользователей	3.4 GHz	30 GB	1 GB	Установка на том же компьютере, что и слой данных (один сервер)
От 100 до 200 пользователей	3.4 GHz	40 GB	2 GB	Установка на том же компьютере, что и слой данных (один сервер)
От 200 до 400 пользователей	3.4 GHz/ Dual	80 GB	2 GB	Установка на том же компьютере, что и слой данных (один сервер)
От 400 до 800 пользователей	2.8 GHz	20 GB	1 GB	Установка на отдельном от слоя данных компьютере (два сервера)
От 800 до 2000 пользователей	2.8 GHz/ Dual	40 GB	4 GB	Установка на отдельном от слоя данных компьютере (два сервера)

Team Foundation Server. Слой данных

Конфигурация	CPU	HD	Память	Комментарии
Менее 20 пользователей	2.2 GHz	8 GB	1 GB	Установка на том же компьютере, что и прикладной (один сервер)
От 20 до 50 пользователей	2.2 GHz	30 GB	1 GB	Установка на том же компьютере, что и прикладной (один сервер)
От 50 до 100 пользователей	3.4 GHz	30 GB	1 GB	Установка на том же компьютере, что и прикладной (один сервер)
От 100 до 200 пользователей	3.4 GHz	40 GB	2 GB	Установка на том же компьютере, что и прикладной (один сервер)
От 200 до 400 пользователей	3.4 GHz/ Dual	80 GB	2 GB	Установка на том же компьютере, что и прикладной (один сервер)

Конфигурация	CPU	HD	Память	Комментарии
От 400 до 800 пользователей	2.8 GHz/ Dual	80 GB	2 GB	Установка на отдельном от прикладного слоя компьютере (два сервера)
От 800 до 2000 пользователей	2.8 GHz/ Quadruple	150 GB	4 GB	Установка на отдельном от прикладного слоя компьютере (два сервера)

Team Build. Рекомендованные аппаратные конфигурации

Конфигурация	Время сборки	CPU	HD	Память
Минимальная, 1-2 проекта, 5-20 пользователей	30 мин.	766 GHz	8 GB	256 MB
Небольшая, 2-20 проектов, 20-100 пользователей	От 30 мин. до 2 часов	1.5 GHz	30 GB	512 MB
Средняя, более 20 проектов, от 100 до 250 пользователей	От 2 до 5 часов	2.6 GHz	50 GB	1 GB
Большая, более 50 проектов, от 250 до 500 пользователей	От 3 до 7 часов	2.8 GHz/ Dual	80 GB	2 GB

Конфигурации для тестирования на удаленных компьютерах

Конфигурация	Компонент	CPU	HD	Память
Минимальная, 1-2 проекта, 5-20 пользователей	test agent	600 GHz	1 GB	256 MB
Минимальная, 1-2 проекта, 5-20 пользователей	test controller	600 MHz	1 GB	256 MB
Минимальная, 1-2 проекта, 5-20 пользователей	Оба компонента	600 MHz	1 GB	512 MB
Небольшая, 2-20 проекта, 20-100 пользователей	test agent	2.0 GHz	5 GB	512 MB
Небольшая, 2-20 проекта, 20-100 пользователей	test controller	1.0 GHz	8 GB	512 MB
Небольшая, 2-20 проекта, 20-100 пользователей	Оба компонента	2.0 GHz	8 GB	1 GB
Средняя, более 20 проектов, от 100 до 250 пользователей	test agent	2.6 GHz	5 GB	2 GB

Об авторе

Алексей Федоров осваивал азы компьютерной грамотности на СМ-4 (клон PDP-11, созданный странами соцлагеря) и ЕС-1840/41 (советский клон IBM PC). В зрелые годы занимался русификацией принтеров Epson FX-80, локализацией операционной системы DR-DOS (компании Digital Research) и средствами разработки Borland C++, читал лекции, писал статьи для российских и международных компьютерных изданий (более 400 статей), книги по различным вопросам программирования (на русском и английском языках). Работал руководителем Управления разработки программного обеспечения, техническим директором швейцарской веб-компании.

В Microsoft — 5 лет, в настоящее время возглавляет отдел по работе с партнерами в Департаменте стратегических технологий. В 2007 г. был награжден как победитель конкурса «Лучший по профессии».

Windows Communication Foundation
http://www.infoq.com/news/2007/05/wcf-web-programming-model

Алексей Федоров

Microsoft® Visual Studio® 2008.
Краткий обзор ключевых новинок

Алексей Федоров — сотрудник отдела стратегических технологий
ООО «Майкрософт Рус» (alexeif@microsoft.com)

Подготовлено к печати издательством «Русская Редакция»
123290, Москва, Шелепихинская наб., д. 32
тел.: (495) 256-6691, тел./факс: (495) 256-7145
e-mail: info@rusedit.com, http://www.rusedit.com

РУССКАЯ РЕДАКЦИЯ