**Windows Forms: Современная модель программирования для создания GUI приложений**

Jeff Prosise, Microsoft Corp.

**Введение**

Чтобы создать GUI приложения в Microsoft .NET нужно использовать Windows Forms. Windows Forms - новый стиль построения приложения на базе классов .NET Framework class library. Они имеют собственную модель программирования, которая более совершеннее, чем модели, основанные на Win32 API или MFC, и они выполняются в управляемой среде .NET Common Language Runtime (CLR). Эта статья дает представление о том, что такое Windows Forms, рассматривая ее от модели программирования до Microsoft Intermediate Language и JIT-транслятора.

Вы уже много слышали, что Microsoft .NET - новая платформа, которая основана на Windows. Это целая новая парадигма программирования, которая изменит путь, которым вы сейчас думаете о написании программ для Windows. Она реализована на библиотеке классов .NET Framework class library и содержит более единую модель программирования, улучшенную защиту и более богатые возможности для написания полнофункциональных веб-приложений. И это только начало.

Windows Forms - одна из наиболее интересных возможностей Microsoft .NET. Если вы знакомы с MFC (или Windows API), то Windows Forms хорошее начало для работы с .NET Framework class library, потому что она позволяет писать традиционные GUI приложения с окнами, формами и т.п. вещами. Однажды, начав работать с Windows Forms вы сможете быстро понять .NET Framework.

Главная выгода от написания Windows-приложений с использованием Windows Forms - это то, что Windows Forms гомогенизируют (создают более однородную (гомогенную) структуру) программную модель и устраняют многие ошибки и противоречия от использования Windows API. Например, каждый опытный программист под Windows знает, что некоторые стили окна могут применяться только к окну, когда оно уже создано. Windows Forms в значительной степени устраняют такое противоречие. Если вы хотите существующему окну задать стиль, который может быть присвоен только в момент создания окна, то Windows Forms спокойно уничтожит окно и вновь создаст его с указанным стилем. Кроме того, .NET Framework class library намного более богаче, чем Windows API, и когда вы будете писать приложения, используя Windows Forms, вы получите в распоряжение больше возможностей. Написание приложения с использованием Windows Forms потребует меньшего количества кода, чем приложения, которые используют Windows API или MFC.

Другая выгода от Windows Forms - вы используете тот же самый API, независимо от языка программирования, который вы выбрали. В прошлом, выбор языка программирования управлял выбором API. Если вы программировали в Visual Basic, вы использовали один API (реализованный на языке Visual Basic), в то время как программисты C использовали Win32 API, а программисты C++, вообще говоря, использовали MFC. MFC-программисту было трудно переключиться на Visual Basic и наоборот. Но теперь такого больше нет. Все приложения, которые используют Windows Forms, используют один API из .NET Framework class library. Знание одного API достаточно позволит программисту писать приложения фактически на любом языке, который он выберет.

Windows Forms ни как не меньше, чем современная модель программирования для GUI приложений. В отличие от модели программирования Win32, в которой многое идет еще от Windows 1.0, новая модель была разработана с учетом всех современных требований. Цель этой статьи состоит в том, чтобы познакомить читателя с моделью программирования Windows Forms. Чтобы компилировать и выполнять примеры кода, приведенного далее, на вашем компьютере должнен быть установлен пакет Microsoft .NET Framework SDK (.NET Framework SDK Beta 1 доступен на сайте Microsoft).

**Модель программирования Windows Forms**

В Windows Forms термин "форма" - синоним окна верхнего уровня. Главное окно приложения - форма. Любые другие окна верхнего уровня, которые имеет приложение - также формы. Окна диалога также считаются формами. Несмотря на название, приложения, использующие Windows Forms, не выглядят как формы. Подобно традиционным Windows-приложениям приложения осуществляют полный контроль над событиями в собственных окнах.

Программисты видят Microsoft .NET через линзу .NET Framework class library. Представьте MFC на порядок больше и вы получите точную картину о ширине и глубине .NET Framework class library. Чтобы облегчить противоречия в обозначениях и придать организацию многим сотням классов, .NET Framework class library разбита на иерархические разделы по именам. Корневой раздел, System, определяет фундаментальные типы данных, используемые всеми приложениями .NET.

Приложения, использующие Windows Forms используют классы System.WinForms. Этот раздел включает такие классы, как Form, который моделирует поведение окон или форм; Menu, который представляет меню; Clipboard, который дает возможность приложениям Windows Forms использовать буфер обмена. Он также содержит многочисленные классы, предоставляющие средства управления, например: Button, TextBox, ListView, MonthCalendar и т.д. Эти классы могут быть включены в приложение либо с использованием только имени класса, либо с использованием полного имени, например: System.WinForms.Button.

В основе почти каждого приложения, написанного с применением Windows Forms, - производный класс от System.WinForms.Form. Образец этого класса представляет главное окно приложения. System.WinForms.Form имеет множество свойств и методов, которые имеют богатый программный интерфейс к формам. Хотите знать размеры клиентской области формы? В Windows вы вызвали бы функцию API GetClientRect. В Windows Forms нужно использовать свойства ClientRectangle или ClientSize.

Приложения, основанные на Windows Forms, которые используют кнопки, списки и другие типы компонентов Windows, используют классы управления System.WinForms, значительно упрощающие программирование управления. Хотите создать стилизованную кнопку с изображением в виде фона? Нет проблем. Включите требуемое изображение в объект System.Drawing.Bitmap и назначьте его свойству кнопки BackgroundImage. Как насчет управления цветом? Вы когда-либо пробовали настраивать цвет фона текстового поля? В Windows Forms это просто: нужно просто присвоить цвет свойству BackColor, все отстальное система сделает сама.

Другой важный "строительный" блок приложения, который использует Windows Forms - класс System.WinForms по имени Application. Этот класс содержит статический метод Run, который загружает приложение и отображает окно.

Вы скажете: если приложения, которые являются Windows Forms, не обрабатывают сообщения, как они отвечают на пользовательский ввод или знают когда рисовать? Много классов имеют виртуальные методы, которые можно переопределить... Например, System.WinForms.Form содержит виртуальный метод OnPaint, который вызывается когда клиентская область формы нуждается в обновлении. OnPaint - один из многих виртуальных методов, который можно переопределить в производном классе для формирования интерактивных форм.

Другая важная грань модели программирования Windows Forms - механизм, который формы используют для ответа на ввод в меню, средств управления и других элементов GUI приложения. Традиционные Windows-приложения обрабатывают сообщения WM\_COMMAND и WM\_NOTIFY используя события процесса Windows Forms. В C# и на других языках, которые поддерживают .NET Common Language Runtime (CLR), события - члены типа первого класса наравне с методами, полями и свойствами. Фактически все управляющие классы (control classes) Windows Forms (а также и многие неуправляющие классы) создают события. Например, кнопка (экземпляр System.WinForms.Button) после нажатия создает событие Click. Форма, которая должна ответить на нажатие кнопки может использовать следующий код, чтобы соединить кнопку на обработчиком события Click:

MyButton.Click += new EventHandler (OnButtonClicked);

...

private void OnButtonClicked (object sender, EventArgs e)

{

 MessageBox.Show ("Click!");

}

EventHandler - специальный обработчик событий, который выполняет метод OnButtonClicked когда MyButton создает событие Click. Первый параметр OnButtonClicked идентифицирует объект, который вызвал событие. Второй параметр в основном бессмысленен для события Click, но используется некоторым другие типами событий, чтобы передать дополнительную информацию.

**Приложение "Hello World" с Windows Forms**

Самый простой способ начать изучение новой платформы - создание приложения "Hello World". В Листинге 1 показана версия, созданная при помощи Windows Forms. Все примеры в этой статье написаны на C#, но вы можете писать приложения Windows Forms на любом языке, для которого есть компилятор .NET. Сегодня это: C#, Visual Basic, JScript и C++.

ЛИСТИНГ 1

using System;

using System.WinForms;

using System.Drawing;

public class MyForm : Form

{

 public MyForm ()

 {

 Text = "Windows Forms Demo";

 }

 protected override void OnPaint (PaintEventArgs e)

 {

 e.Graphics.DrawString ("Hello, world", Font,

 new SolidBrush (Color.Black), ClientRectangle);

 }

 public static void Main (string[] args)

 {

 Application.Run (new MyForm ());

 }

}

Начнем сначала. Слово "using" вверху файла позволяют сослаться на классы в пакетах System, System.WinForms и System.Drawing. Например,

using System.WinForms;

вы пишите

public class MyForm : System.WinForms.Form

вместо

public class MyForm : Form

В приложении, использующем Windows Forms, каждое окно - или форма - представлено экземпляром класса, производного от System.WinForms.Form (Листинг 1 - класс MyForm). Конструктор MyForm устанавливает текст заголовка формы "Windows Forms Demo" используя свойство Text. Text - одно из более чем 100 свойств, которые форма наследует от System.WinForms.Form, но пока единственное, в котором вы нуждаетесь.

Как вы знаете, окна получают сообщения WM\_PAINT и большинство перерисовок экрана выполнено в ответ на эти сообщения. В Windows Forms эквивалент сообщения WM\_PAINT - виртуальный метод по имени OnPaint. Производный класс формы может переопределить этот метод в случае надобности выполнять собственную перерисовку в ответ на сообщения WM\_PAINT.

Обратите внимание на ключевое слово в Листинге 1, которое компилятор C# интерпретирует как подтверждение, что вы хотите переопределить виртуальный метод, унаследованный от базового класса. Перопределенный OnPaint записывает "Hello, world" в клиентской области формы. OnPaint вызывает объект PaintEventArgs (System.WinForms.PaintEventArgs), который содержит свойства Graphics и ClipRectangle. Свойство Graphics ссылается к объекту Graphics (System.Drawing.Graphics), который является эквивалентом контекста устройства (device context) в Windows Forms. ClipRectangle производное объекта Rectangle (System.Drawing.Rectangle), который описывает какая часть формы является недопустимой.

Метод OnPaint из MyForm использует Graphics.DrawString, чтобы выполнить вывод на экран. Первый параметр DrawString - непосредственно само сообщение (строка) "Hello, world". Второй - объект Font (System.Drawing.Font), который описывает шрифт для вывода текста. MyForm.OnPaint использует шрифт формы (которой установлен в свойстве Font формы). Третий параметр - Brush (System.Drawing.Brush) - объектное определение цвета текста. Четвертый и заключительный параметр - прямоугольник области, куда требуется вписать текст.

Заключительная часть MyForm - статический метод Main. Main - точка входа в приложение. Каждое приложение .NET должно иметь этот метод. Main может быть объявлен любым из следующих способов:

public static void Main ()

public static int Main ()

public static void Main (string[] args)

public static int Main (string[] args)

Параметр args метода Main - строковый массив параметров командной строки (задаваемых при вызове программы). Элемент args[0] хранит первый параметр командной строки, args[1] - второй и т.д. Как правило, в каждом приложении метод Main выполняется только однажды (компилятор Microsoft C# допускает использование ключа /main, указывающего в каком классе содержится метод Main, если приложение имеет несколько классов с методами Main). Main может находиться в любом классе, определенном в приложении.

Отображение нашей формы на экране - простой пример выполнения MyForm и передачи действия Application.Run. Application - другой класс, определенный в System.WinForms. Метод Run создает форму, отображает ее на экране и обрабатывает сообщения к ней. Следующая инструкция в Листинге 1

Application.Run (new MyForm ());

обрабатывает MyForm и показывает форму.

Сохраните текст программного кода (Листинг 1) в файле Hello.cs и откомпилируйте его. Чтобы это сделать, откройте окно командной строки, перейдите в каталог файла Hello.cs и введите:

csc /target:winexe /out:Hello.exe /reference:System.dll

/reference:System.WinForms.dll /reference:System.Drawing.dll

/reference:Microsoft.Win32.Interop.dll Hello.cs

Команда csc вызывает компилятор Microsoft C#. "Hello.cs" указывает на файл, который требуется откомпилировать. Ключ "/target:winexe" сообщает компилятору, что нужно создать GUI-приложение для Windows, а "/out:Hello.exe" задает имя файла программы (этот ключ можно опустить, т.к. в данном случае по умолчанию все равно будет создан Hello.exe, т.к. CS-файл назван Hello.cs). Ключи "/reference" указывают ссылки на внешние классы, например, для System.WinForms.Form и System.Drawing.Size. Для краткости допускается заменять "/target" и "/reference" на "/t" и "/r".

Hello.exe не обычный EXE-файл, это .NET-программа, содержащая следующие важные элементы:

Microsoft Intermediate Language (MSIL), сгенерированный при помощи C#

Метаданные, описывающие типы (классы), определенные в приложении, и типы (например, System.WinForms.Form), на которые ссылается приложение, находящиеся в другом месте (например, в MsCorLib.dll и System.WinForms.dll)

Декларация, описывающая требуемые файлы для сборки приложения

В языке .NET, сборка - это коллекция из одного или более файлов, создающих модуль. Наша сборка содержит только один файл — Hello.exe — и этот факт отмечен в декларации внутри выполнимой программы. Декларация физически сохранена как часть метаданных. Каждая управляемая выполнимая программа — это есть любой PE-файл, который содержит MSIL - часть сборки и каждая управляемая выполнимая программа имеет метаданные внутри. Один из файлов в .NET-сборке содержит декларацию идентификации файлов, которые должны быть включены при сборке и общедоступные типы. Компилятор C# производит всю необходимую инфраструктуру.

Теперь когда вы откомпилировали Hello.exe, можете набрать в командной строке:

Hello

Посмотрите на рис.1, чтобы увидеть результат работы нашей программы.