5-6[3]

В номере:

Уважаемый читатель!

Сейчас у вас перед глазами третий номер журнала Алгоритм, последний номер, полностью выложенный в Сети. Начиная с четвёртого номера некоторые материалы будут доступны только тем, кто подписался на журнал в твёрдой копии. Причина одна и она вполне очевидна - подписчик, заплативший деньги за издание, должен иметь преимущество перед тем, кто читает журнал бесплатно. еПоэтому щё раз напоминаю, что подписной индекс журнала в Украине 91132, подписаться можно также и через редакцию, что будет несколько дешевле. Также в этом случае отправка издания по почте будет происходить раньше, т.е. вы получите свой экземпляр раньше, чем те, кто подписался через почту. Документы на подписку лежат по адресу

http://dotnetgrains.sql.ru/alg/algsubscribe.htm.

В этом номере вы сможете прочесть материал Скотта Ханесльмана в переводе Никиты Зимина о том, что должен знать настоящий разработчик на платформе .Net. В следующих номерах журнала мы будем публиковать ответы на эти вопросы и подробно разбирать их.

Так же в номере появилась первая статья из цикла о ключевых словах языка C#, этот цикл продолжится в следующих номерах и, думаю, будет полезен не только начинающим, но и более опытным программистам, использующимэтот язык.

В номере сознательно сделан упор не на новости, а на статьи. В следующем, четвёртом номере, распределение статей и новостей будет более сбалансированным, но, опять же, полностью они будут доступны только подписчикам.

С этого номера редакция начинает оплату авторских статей и переводов. Если вы хотите стать автором - пишите на адрес hdrummer@gmail.com, требования к авторам можно прочесть здесь -

http://dotnetgrains.sql.ru/alg/algauthors.htm.

На этом всё, приятного чтения!

С уважением,

Главный редактор журнала “Алгоритм”
Чужа Виталий Ф.,
hdrummer@gmail.com

Сегодня мы представляем вашему вниманию, уважаемый читатель, интервью с человеком-легендой главного форума сайта SQL.RU – Вячеславом Брылёвым aka Glory.

Здравствуйте, Вячеслав! Спасибо за то, что согласились дать интервью. Уже традиционно, для начала расскажите немного о себе.

Добрый день, Виталий! Здравствуйте, уважаемые читатели! Ну, о себе так, о себе. Родился я 25-го мая 1970г. в Эстонии, где проживаю и работаю до сегодняшнего дня. Сам я, как и мои родители, русский, хоть и гражданин теперь уже Эстонской Республики. Правда, родители в отличие от меня сюда приехали самостоятельно. Остальные пункты биографии вроде как стандартные – школа, институт, армия (еще Советская), опять институт, работа.

Вячеслав, как шутил один комик, вы широко известны в узких кругах программистов и администраторов Microsoft SQL Server'a. Тем не менее, именно на сайте SQL.RU стали широко известны ваши таланты, как вы впервые встретились с сайтом SQL.RU?

Наверное, так же, как и большинство посетителей форума. В связи с началом плотной работы на MSSQL искал соответствующие ресурсы в Интернет. Просто понравился сам форум, его аудитория и т.д. Становиться «человеком-легендой» именно на этом форуме в мои планы не входило :) Наверное, совпали звезды, как говорят астрологи, не исключая конечно влияние поисковых движков Google и Yandex :). А потом уже просто не оставалось времени искать что-то другое, да и не очень-то хотелось. Хотя время от времени я посещаю и другие форумы, в том числе не русскоязычные. С создателями ресурса sql.ru до последнего времени был знаком исключительно заочно.

Чем является для вас общение на профессиональном форуме?

Хм... Думаю, что это уже некий образ жизни, в котором переплетено многое. И самовыражение, и желание помочь другим, и самому поучится, и свои знания показать. Иногда у любого разработчика/администратора наступает момент зашорености, когда пытаешься, как тот Сизиф, толкать камень проблемы исключительно в одном направлении. В этот момент очень ценен, по-моему, взгляд на проблему постороннего человека. Т.к. позволяет тебе понять, какой у тебя есть выбор в решении задачи и сделать этот выбор осознанно. Так собственно и накапливается опыт решения.
К тому же свои знания тоже надо освежать – ведь ничто не вечно под луной. Отследить самому все, что поменялось в самом сервере или около него, трудно. А тут в форуме можно узнать иногда вещи, которые даже не задумывался проверить.

Как удаётся поддерживать хорошую «спортивную» форму?

Сейчас уже на это влияет и чувство долга – все-таки звания модератора и MVP так сказать обязывают. С другой стороны, на мой взгляд, профессионализм в любой области, включая разработку и администрирование БД, это именно ежедневные тренировки. По-моему, профессионал в большей степени не тот, кто знает о продукте его редчайшие особенности, а тот, кто знает о продукте 1000 и 1 документированную мелочь и может сопоставить их влияние друг на друга. Поэтому, как и спортсмену для поддержания физической формы нужны ежедневные нагрузки, так профи какого-то ПО нужно все время проверять актуальность своих знаний. Особенно, если это ПО не статично, а развивается.
Так что получается, что сам форум и помогает мне держать «форму».

Не надоедает ли отвечать на одни и те же вопросы? Ведь наверняка они повторяются?

Про ежедневные «тренировки» я уже упомянул выше. Второй аспект по-моему в том, что форум должен выполнять и некоторые образовательные функции. Конечно же, не в том объеме как учебные заведения или курсы, но отчасти. А там ведь преподаватели читают один и тот же курс много-много раз. И некоторые из них даже находят это занятие увлекательным :). Тем более что в форуме достаточно порой просто указать вопрошающему, откуда он может почерпнуть ответы на свои вопросы. Ответы на многие повторяющиеся вопросы не являются моим ноу-хау, просто я знаю, где про это написано. Отчего

бы не поделится этими знаниями? Тем более что житейская истина о том, что все сделанное тобой тебе же и вернется, работает. Вот уже начал интервью давать журналам, толи еще будет :) Кстати, не я один не устаю отвечать на повторяющиеся вопросы на форуме. Хотелось бы им тоже сказать «спасибо» и «так держать».
С третьей стороны марку сайта и форума надо же поддерживать. Тем более что понятие сообщество подразумевает поддержку своих участников.
Разумеется, мне не нравятся вопросы вроде «дайте мне» или «сделайте за меня». Я считаю таких людей, мягко говоря, халявщиками. Возможно, я и излишне резок в своих оценках; но все равно считаю, что для получения чего-то нужно сначала что-то вложить. Хотя у меня, как и у всех нормальных людей просто бывает плохое настроение :)

Как вы пришли к Microsoft SQL Server'у? Чем занимались до этой, наверное, важной в вашей жизни встречи?

С базами данных «познакомился» на 3-ем курсе института, когда устроился на практику (а потом на работу) в маленькую фирму, которая создавала программы складского и бухгалтерского учетов для входящих в моде ПК. Разумеется, это были еще не SQL базы, а простые dbf-ы. Ну а дальше все пошло по накатанной колее – популярность и необходимость всяческих учетов на ПК росла, соответственно росли мой опыт и знания в области СУБД. Мест работы, как ни странно, у меня было не много – всего четыре, включая текущее. Просто, наверное, везло с профилем компаний – учет в страховой фирме намного интереснее простого складского учета :)
Пять лет назад с переходом на последнее место работы основным «орудием» моего производства стал MS SQL Server. Встреча, как видится сейчас, была действительно важной.

Что вы считаете основным преимуществом и основным недостатком этого продукта?

Дааа... Сложные Вы вопросы задаете :)

Основным преимуществом я бы назвал внутренний движок сервера. Особенно радует его улучшение от версии к версии. Видно, что Microsoft поставила себе целью сделать добротный продукт. И видно, что эта цель планомерно осуществляется.

К недостаткам я бы отнес то, что MS SQL Server требует для полноценной работы наличие хоть какого-то клиентского приложения. Т.е. нельзя на TSQL создать полностью автономный самодостаточный код. Я очень надеюсь и верю, что уже в этом году в новой версии SQL Server картина кардинально поменяется.

Есть ли у вас проекты, которыми вы гордитесь как профессионал, проекты, выполненные на высоком уровне и высокой сложности?

К сожалению, (а может к счастью) все проекты в нашей фирме сугубо внутренние, т.к. основная деятельность компании не связана с производством ПО. Она связана с оказанием услуг сотовой и стационарной связи.
У нас как бы один бесконечный проект со стандартным названием data warehouse. Смысл его в том, чтобы собрать из различных программных и аппаратных средств абсолютно все данные, которые ими генерируются. И связать их все между собой. А так как список услуг фирмы все время пополняется, то окончание проекта вроде как и не предвидится :)

Что особенно запомнилось при реализации такого проекта?

Выбор средств реализации :) Было несколько пилотных проектов, потому что на начальном этапе использовался аутсорсоринг, т.е. привлекались программистские фирмы, реализовавшие подобные проекты.

В прошлом году вам был присвоен статус Microsoft MVP, какие преимущества есть у такого статуса и помогли ли они вам в вашей работе?

Преимущества в основном выражаются в большем объеме доступной информации. Разумеется, использование этой информации регламентируется. Информация включает в себя как доступ, к примеру, к бета-версиям статей в MS KnowledgeBase, так и к ньюс-группам, онлайн-чатам и презентациям непосредственно с разработчиками из Microsoft. Есть и более так сказать материальные выгоды в виде MSDN подписки и получения бета-версий ПО. Все это, правда требует дополнительного времени, которого не хватает.

Лично мне выгод от статуса MVP вот так прямо непосредственно в работе пока не было. Все проблемы с текущими версиями MS SQL Server и так можно решить через службу поддержки. А вот возможность повлиять своими комментариями на функционал следующих версий того же SQL Server-а, греет душу, так сказать. Посмотреть на «кухню» создания подобного рода ПО если не изнутри, то хотя бы с более близкого расстояния тоже очень интересно.

Ваши планы на будущее?

Планов, как говорится, просто громадьё. Есть несколько задумок по расширению сайта sql.ru, по написанию своих статей, переводу чужих. Не хочу сильно распространяться на эту тему, т.к. многие проекты еще только в стадии задумки. За форумом опять же надо приглядывать. Да и родная фирма не даст без дела засиживаться – грядет внедрение новой биллинговой системы

Большое спасибо за интересный рассказ, желаю Вам успехов во всех ваших начинаниях.

Спасибо Вам, Виталий. Вам и всем читателям также желаю всего наилучшего.

Что новенького...

Вышел Portable.NET 0.7.0

Portable.NET – набор инструментов, разработанный для построения и запуска веб-сервисов и приложений .NET.

Проект содержит полный набор инструментария для разработки, библиотеку времени выполнения, библиотеки, сфокусированные на совместимости со спецификациями ECMA. В новой версии 0.7 присутствуют десятки улучшений производительности, исправлены ошибки в работе с XML и WinForms и т.д. Изменения таковы:

Библиотека времени выполнения

- *Реализован PPC анроллер (Gopal V).*
- *Новые внутренние обращения в классе StringBuilder (Klaus Treichel).*
- *Исправлен isinst – теперь он действует подобно castclass, как указано в спецификации (Klaus Treichel).*
- *Внутренние вызовы получают информацию о номере ошибки errno для Process (Hermann Weiss).*

Исправлена возможная блокировка движка в коде нити (Klaus Treichel).

- *Оптимизирован конвертер CVM – теперь он вызывается только при необходимости (Klaus Treichel).*
- *Исправлено использование регистра x86 с оператором memmove в интерпретаторе (Klaus Treichel).*
- *Оптимизирован FindInRange для IndexOf и LastIndexOf (Klaus Treichel).*
- *Улучшена поддержка профилирования движка (Marc Haisenko).*

Компилятор C#

- *Исправлена ошибка, вызывавшая переполнение при обработке оператора switch (Gopal V).*

Документация

- *Улучшена документация для режима построения файлов csant (Stephen Compall).*

Поддержка платформы

- *Отключен SIGPIPE для операция с сокетами (Klaus Treichel).*
- *Введена широковещательная поддержка UDP (Doru Budai).*

Другое

- *Обновления пакетирования для Debian (Russell Stuart).*
- *Доступен пакет времени выполнения (всего 2MB!), с полным SDK.*

Для того, чтобы узнать больше, посетите веб-сайт Portable.NET

http://www.autoline.com.au/labs#pnet.

Ключевые слова в C# : A

Автор : Чужа В.Ф.

В этом цикле статей мы попробуем разобраться с ключевыми словами языка C#, пройдя по ним прямо по алфавиту и разобрав несколько примеров к каждому ключевому слову. Также потом посмотрим, какие изменения были внесены при появлении C# 2.0. Думаю, что такой подход будет полезен новичкам, начинающим программировать на этом языке.

Abstract

Использование в классах

Модификатор ***abstract*** может быть использован с классами, методами, свойствами, индексаторами и событиями.

Используется с классом, если нужно указать, что класс будет являться базовым классом для других классов, т.е. объекты этого класса не могут быть созданы.

Абстрактные классы обладают такими особенностями:

• Объект такого класса не может быть создан

```
using System;

namespace _abstract
{
class _absrtactMain
{
[STAThread]
static void Main(string[] args)
{
// ошибка компиляции
myAbstract a = new myAbstract();
}
}
abstract class myAbstract
{}}
```

Листинг 1. Ошибка компиляции при попытке создания объекта абстрактного класса Class1. cs(10): Cannot create an instance of the abstract class or interface '_abstract.myAbstract'

• Абстрактный класс может содержать абстрактные методы и акцессоры

```
abstract class myAbstract
{
//абстрактный метод
public abstract void Demo();
}
```

Листинг 2. Абстрактный класс, содержащий абстрактный метод – метод без реализации

• К абстрактному классу нельзя применять модификатор sealed, который запрещает наследовать от класса

```
//ошибка компиляции
abstract sealed class myAbstract
{
//абстрактный метод
//- не имеет реализации
public abstract void Demo();
}
```

Листинг 3. Попытка запретить наследование от абстрактного класса заканчивается ошибкой компиляции: Class1.cs(15): '_abstract.myAbstract' cannot be both abstract and sealed

• Неабстрактный класс, получившийся путём наследования от абстрактного класса, должен включать реализацию всех унаследованных абстрактных методов и акцессоров

```
abstract class myAbstract
{
//абстрактный метод
public abstract void
DemoInheritedImplemented();
//абстрактный метод
```

```
public abstract void
DemoInheritedNotImplemented();
}
class myAbstractChild : myAbstract
{
public override void
DemoInheritedImplemented()
{

}
}
```

Листинг 4. Попытка создания класса-наследника, нереализующего все абстрактные методы приводит к ошибке компиляции: Class1.cs(24): '_abstract.myAbstractChild' does not implement inherited abstract member '_abstract.myAbstract.DemoNonInherited()'

Абстрактный класс также должен содержать объявления всех членов интерфейса, если он наследуется от него.

```
interface IMyAbstractInterface
{
void CalculateGas();
}
abstract class Calculator:
IMyAbstractInterface
{
public abstract void CalculateGas();
}
```

Листинг 5. Обязательное соответствие между членами интерфейса и абстрактного класса, наследуемого от него.

Использование в методах и свойствах

Модификатор ***abstract*** используется в объявлении метода или свойства для индикации того, что метод или свойство не имеют реализации.

Абстрактные методы обладают следующими особенностями:

- Абстрактный метод неявно является виртуальным методом

```
abstract class myAbstract
{
//попытка объявления виртуальным
//абстрактного метода
public virtual abstract void
DemoInherited();
}
```

Листинг 6. Хотя при объявлении его виртуальным выдаётся ошибка: Class1.cs(18): The abstract method '_abstract.myAbstract.DemoInherited()' cannot be marked virtual

- Объявления абстрактных методов разрешены только в абстрактных классах.

```
class MakeMyDay
{
//ошибка компиляции
public abstract void MakeItNow();
}
```

Листинг 7. Попытка объявить абстрактный метод в неабстрактном классе успехом не увенчалась: Class1.cs(40): '_abstract.MakeMyDay.MakeItNow()' is abstract but it is contained in nonabstract class '_abstract.MakeMyDay'

- Поскольку объявление абстрактного метода не предполагает какой-либо реализации, то заканчивается оно просто точкой с запятой. Например:

```
public abstract void
DemoInheritedNotImplemented();
```

- Реализация абстрактного метода находится в перекрывающем методе, который является членом неабстрактного класса (см. Листинг 4.).
- Использование модификаторов static или virtual в абстрактном методе приведёт к ошибке.

```
//ошибка компиляции
abstract class myAbstract
{
//абстрактный метод
public abstract void DemoInherited();
//попытка объявить статический
//абстрактный метод
public static abstract void
DemoNonInherited();
}
```

Листинг 8. Попытка объявить статический абстрактный метод приводит к ошибке: Class1.cs(20): A static member '_abstract.myAbstract.DemoNonInherited()' cannot be marked as override, vir-

tual, or abstract

Абстрактные свойства ведут себя также, как и методы, за исключением разницы в объявлении и использовании.

• Попытка использования модификатора abstract для статического свойства также приведёт к ошибке компиляции.

```
public abstract static string Name
{
get {return name;}
set {name = value;}
}
```

Листинг 9. Попытка объявить статическое свойство абстрактным приводит к ошибке компиляции: Class1.cs(38): A static member '_abstract. myAbstractChild.Name' cannot be marked as override, virtual, or abstract

• Абстрактное свойство может быть перекрыто в производном классе путём создания свойства, использующего модификатор override.

```
abstract class myAbstractProperty
{

public abstract string Name
{
get;
set;
}
}

class myAbstractPropertyChild :
myAbstractProperty
{
public override string Name
{
get { throw new
NotImplementedException(); }
set { throw new
NotImplementedException(); }
}
}
```

Листинг 10. Реализация абстрактного свойства в производном классе.

Пример использования абстрактных классов.

```
using System;

namespace _abstract
{
class _absrtactMain
{
[STAThread]
static void Main(string[] args)
{
DellPC dpc = new DellPC(300, 19);
dpc.PrintHDDMonitorCharacteristics();
}
}

/// <summary>
/// Некий абстрактный
///персональный компьютер
/// </summary>
public abstract class PC
{
public abstract int HDDVolume
{
get;
set;
}
public abstract int MonitorSize
{
get;
set;
}
public abstract void
PrintHDDMonitorCharacteristics();
}

/// <summary>
/// Конкретный ПК,
///например от компании Dell
/// </summary>
public class DellPC : PC
{
int hddvolume;
int monitorsize;

public DellPC(int hdd, int monitor)
{
hddvolume = hdd;
monitorsize = monitor;
}
public override int HDDVolume
{
get { return hddvolume; }
set { hddvolume = value; }
}

public override int MonitorSize
{
get { return monitorsize; }
```

```
set { monitorsize = value; }
}

public override void
PrintHDDMonitorCharacteristics()
{
Console.WriteLine("HDD volume : {0} GB,
Monitor size : {1} inches.", HDDVolume,
MonitorSize);
}
}
```

Листинг 11. Пример использования абстрактных классов, методов и свойств.

As

Оператор ***as*** используется для осуществления преобразований между совместимыми типами. Используется в виде выражения:

выражение as *тип*,

где:

выражение
выражение ссылочного типа
тип
ссылочный тип

Примечания

Оператор ***as*** похож на операцию приведения типов, за исключением того, что при ошибке приведения генерируется исключение, а при использовании оператора as - null. Формально, выражение вида
`expression as type`

эквивалентно выражению

```
expression is type ? (type)expression :
(type)null
```

за исключением того, что expression вычисляется только один раз.

Заметьте, что оператор as выполняется только для ссылочных и упакованных типов.

```
using System;

namespace _as
{

class Class1
{
[STAThread]
static void Main(string[] args)
{
int x = 200;
byte y = 0;
y = x as byte;
}
}
}
```

Листинг 1. Попытка приведения двух значимых типов приводит к ошибке: Class1.cs(13): The as operator must be used with a reference type ('byte' is a value type)

Оператор as не работает для значимых типов, определённых пользователем. Вместо этого оператора в таких случаях нужно использовать операции приведения, как показано в Листинге 2.

```
using System;

namespace _as
{

class Class1
{
[STAThread]
static void Main(string[] args)
{
int x = 200;
byte y = 0;
y = (byte)x;
}
}
}
```

Листинг 2. Таким образом всё работает.

Однако в таком случае нужно помнить о том, что может быть выход за границы диапазона типа – например, если x = 256, то y будет равен 0 и никакого исключения в этом случае не произойдёт. Если же вы хотите обезопасить себя от такого рода ситуаций, то необходимо воспользоваться оператором ***checked***, речь о котором пойдёт ниже. Применив его, мы получим исключение переполнения OverflowException, если таковое произойдёт.

Реализация простого WinSock приложения с использованием технологии .Net...

...или пишем шпиона своими руками

Автор: А.Лиман

Часть 1

Учимся говорить

Автор не имеет цели написать полноценное клиент-серверное приложение. Для этого у него просто нет всех необходимых знаний. Но, тем не менее, он надеется, что статья получилась интересной.

Введение

Технологию .Net использую уже третий год. В связи со спецификой задач нашей фирмы писали в основном клиент-серверное программное обеспечение для доступа к базам данным. Не могу сказать, что работа была неинтересная или скучная, но все равно я чувствовал, что развиваюсь только в одном направлении. Понимая, что если не заняться другими направлениями .Net самостоятельно, то так и останусь «сидеть» на СУБД. А хотелось освоить и другие, не менее интересные, направления программирования, которые предоставляет технология .Net.
Одноизнаправлений,котороеменязаинтересовало – это программирование WinSock приложений под .Net. Ясное дело, что осваивать что-то новое всегда легче, если у тебя есть реальная задача, так как ты знаешь, что должно получиться и время от времени сталкиваешься с разными проблемами, решения которых помогает в освоении темы.

И тут однажды мой коллега подкинул мне идею создания одного интересного приложения. Суть приложения проста, но процесс его реализации, как мне показалось, помог бы мне в изучении интересовавшей меня темы. Кроме того, программка должна была получиться интересной и с практической точки зрения её применения.

Постановка задачи

Итак, подошло время рассказать о сути приложения. Как я уже говорил, она проста. Программа должна состоять из двух частей.

Одна часть, назовем её „клиент", должна уметь:
- работать бесшумно в фоновом режиме на компьютере потенциальной "жертвы";
- находить в локальной сети другую часть программы,назовемее„сервер",ирегистрироваться в ней;
- по просьбе сервера пересылать изображение экрана пользователя (фактически сымитировать нажатие PrintScreen)

Серверная часть должна уметь:
- регистрировать все клиентские части программы;
- отсылать им и принимать от них некоторые сообщения;
- принимать изображения, поступившие от них;
- отображать полученные изображения на экране;
- взаимодействовать с пользователем.

Немного теории

Надеюсь, всем примерно стало ясно, что у нас должно получиться в результате? Тогда продолжим. Итак, теперь нужно определиться с технологиями, которые мы будем применять, решая нашу задачу. Как я уже говорил, в то время меня интересовала тема сокетов в .Net. Поэтому, долго не колеблясь, взялся изучать пространства имен и классы .Net Framework, которые бы мне

пригодились при разработке.
Весь функционал для работы с сокетами в библиотеке .Net Framework сосредоточен в пространстве имен System.Net.Sockets. Вот что нам о нем говорит MSDN:

MSDN

Пространство имен System.Net.Sockets предоставляет управляемую реализацию интерфейса Windows Sockets (Winsock) для тех разработчиков, которые нуждаются в надежном управлении доступом к сети. Классы TCPClient, TCPListener и UDP-Client инкапсулируют детали создания TCP и UDP подключений к Интернету.

Описание ключевых классов можно также посмотреть в MSDN:

MSDN

Класс Socket реализует интерфейс сокетов Berkeley. Обеспечивает широкий набор методов и свойств для сетевых взаимодействий. Класс Socket позволяет выполнять как синхронную, так и асинхронную передачу данных с использованием любого из коммуникационных протоколов, имеющихся в перечислении ProtocolType.
Класс TcpClient обеспечивает простые методы для подключения, а также приема и передачи потоков данных в сети в синхронном блокирующем режиме.
Класс TcpListener обеспечивает простые методы, предназначенные для ожидания и приема в блокирующем синхронном режиме входящих запросов на подключения.

Если подсмотреть реализацию классов TCPClient и TCPListener через рефлектор, то можно заметить, что и тот и другой является оберткой над классом нижнего уровня, а именно над классом System. Net.Sockets.Socket. В том же MSDN'е пишут:

MSDN

Если разрабатывается относительно простое приложение, в котором применяется только синхронная передача данных, рассмотрите возможность использования классов TcpClient, Tc-pListener и UdpClient. Эти классы предоставляют простой и удобный для пользователя интерфейс взаимодействия с объектом Socket.

Ну что ж, следуя совету из MSDN, попробуем реализовать логику нашего приложения через классы TCPClient, TCPListener. Рассмотрим более детально эти классы.

Итак, класс TCPClient поможет нам просто и быстро соединиться с сервером и передать ему данные. Соединиться можно либо вызвав один из перегруженных конструкторов класса, указав параметры соединения, либо создать объект класса TCPClient через дефолтный конструктор и вызвав метод Connect. Вот простой пример соединения с сервером с использованием этого класса взятый из MSDN:

```
//Пример соединения с сервером через
конструктор
try{
TcpClient tcpClient = new TcpClient
("www.contoso.com", 11000);
}
catch (Exception e ) {
Console.WriteLine(e.ToString());
}

//Пример соединения с сервером через
//метод Connect
TcpClient tcpClient = new TcpClient();
try{
tcpClient.Connect("www.contoso.com",
11002);
}
catch (Exception e ) {
Console.WriteLine(e.ToString());
}
```

Метод Connect будет выполнять блокирование до установления соединения или до отказа. Если подключиться по какой либо причине не получилось, то можно отловив исключение SocketException, воспользоваться его свойством SocketException.ErrorCode для получения кода ошибки. Получив этот код, можно обратиться за подробным описанием ошибки к документации по кодам ошибок Windows Socket Version 2 API в библиотеке MSDN.

После успешного установления подключения к удаленному хосту можно воспользоваться методом TcpClient.GetStream, чтобы получить основной объект NetworkStream. Класс NetworkStream несет в себе функционал для обеспечения записи/

считывания данных между двумя соединенными объектами Socket. Он наследуется от класса System.IO.Stream, который является абстрактным базовым классом всех потоков.

MSDN
Поток — это абстракция последовательности байтов, например файл, устройство ввода-вывода, канал взаимодействия процессов или сокет TCP/IP.

```
try{
TcpClient tcpClient = new
TcpClient("www.contoso.com", 11000);

NetworkStream networkStream =
tcpClient.GetStream();

if(networkStream.CanWrite){
Byte[] sendBytes = Encoding.ASCII.
GetBytes("Есть там кто?");
networkStream.Write(sendBytes, 0,
sendBytes.Length);
}
else{
Console.WriteLine("Невозможно записать
данные в этот поток.");
tcpClient.Close();
return;
}
if(networkStream.CanRead){
// Считываем данные NetworkStream в
//байтовый буфер.
byte[] bytes = new
byte[tcpClient.ReceiveBufferSize];
// Этот метод блокируется до тех пор,
//пока не будет считан
//хотя бы один байт
networkStream.Read(bytes, 0,
(int) tcpClient.ReceiveBufferSize);

// Преобразовываем полученные данные
//от хоста в строку и отображаем ее на
//консоли

string returndata = Encoding.ASCII.
GetString(bytes);
Console.WriteLine("Вот что хост нам
переслал: " + returndata);
}
else{
Console.WriteLine("Невозможно считать
данные из этого потока.");
tcpClient.Close();
return;
}
}
catch (Exception e ) {
Console.WriteLine(e.ToString());
}
```

Для синхронной записи/считывания данных в поток используются методы Write и Read. За их детальным описанием можно обратиться в MSDN. Стоит лишь отметить, что метод Read при отсутствии доступных для чтения данных будет выполнять блокирование до тех пор, пока не появятся данные. Чтобы предотвратить блокирование, можно использовать свойство DataAvailable, чтобы определить, имеются ли данные в очереди для чтения во входном сетевом буфере. Если свойство DataAvailable имеет значение true, вызов метода Read немедленно возвратит результат.

Итак, как соединиться с хостом стало более-менее ясно, теперь разберемся, как можно реагировать на подключения TcpClient'ов со стороны сервера. Как уже говорилось, для этой цели служит класс TcpListener, который дает возможность прослушивать входящие запросы на подключение.

В конструктор класса TcpListener можно передать локальный IP-адрес и номер порта для прослушивания входящих попыток подключения. Метод TcpListener.Start запускает ожидание входящих запросов на установление соединения. Вот что по этому поводу пишет MSDN:

MSDN
Метод Start инициализирует основной объект Socket, связывает его с локальной конечной точкой и выполняет прослушивание входящих попыток подключения. Если получен запрос на подключение, метод Start поставит этот запрос в очередь и продолжит ожидание последующих запросов, пока не будет вызван метод Stop. Чтобы удалить подключение из входной очереди, воспользуйтесь методом AcceptTcpClient или AcceptSocket. Метод AcceptTcpClient удалит подключение из очереди и возвратит объект TcpClient, который может быть использован для передачи и приема данных. Если необходимо убедиться в наличии доступного подключения перед попыткой выполнением его приема, воспользуйтесь методом Pending.

Пример использования класса TcpListener пока не буду приводить, чтобы сильно не нагружать статью примерами, так как по ходу разработки приложения мы еще столкнемся с этим классом.

Думаю, на этом небольшой экскурс в пространство System.Net.Sockets можно закончить. Если же что-то еще осталось неясным, то за помощью можно обратиться в MSDN.

Начинаем проектировать

Получив кое-какое представление о реализации работы с сокетами в .NET Framework приступим к проектированию нашего приложения. Прежде всего, необходимо набросать примерный алгоритм работы клиентской и серверной частей программы , и определиться, как они будут между собой взаимодействовать.

Работу клиентской части программы можно описать примерно так:
- Программа запускается незаметно для пользователя.
- Инициализируется процесс поиска серверной части.
- Если серверная часть программы найдена – регистрация в ней и инициализация процесса ожидания команды от сервера.
- Если поступила команда от сервера – немедленное ее выполнение и отправка ответа серверу.

Работу серверной части программы можно описать примерно так:
- Запускается серверная часть, которая должна иметь графический пользовательский интерфейс (GUI)
- Инициализируется процесс прослушивания запросов на подключение.
- Инициализируется процесс анализа очереди клиентов
- Если поступил запрос на подключение – принятие клиента и добавление в очередь соединенных клиентов.
- Анализируется очередь соединенных клиентов, если такие есть, то запускается обработка данных, которые передал клиент.
- Инициализируется процесс проверки соединения с клиентом.
- В зависимости от типа сообщения, которое передал клиент, принимается решение о дальнейших действиях.
- GUI приложение должно давать возможность пользователю взаимодействовать с подключенными клиентскими частями программы в интерактивном режиме.

На Рис. 1-2 изображена примерная схема работы клиентской и серверной части приложения.

Теперь определим типы команд, которые может послать сервер клиентской части программы:
- проверка соединения – эта команда, по сути, условна, так как реально она не будет передаваться нашему клиенту, потому что для того, чтобы проверить связь, достаточно просто попытаться соединиться;
- получить одно изображение – просим зарегистрированного клиента прислать нам одно изображение экрана;
- получить серию изображений - просим зарегистрированного клиента прислать нам серию изображений, для более пристального наблюдения;
- прекратить передачу серии изображений
- команда на прекращение передачи серии изображений.

Исходя из списка типов команд сервера, формируем типы ответных сигналов клиента:
- запрос на авторизацию либо просто проверка связи;
- отправлено изображение в ответ на запрос о получении одного изображения;
- отправлено изображение в ответ на запрос о получении серии изображений.

Программным способом команды и типы ответов клиента можно представить в виде перечислений. Вот как они могут выглядеть:

```
// типы сигнала, которые могут
поступить от сервера
public enum ServerSignalTypes
{
IsAlive, GetImage, StartMovie,
StopMovie
}
// типы сигналов, которые могут быть
//получены
public enum ClientSignalTypes
{IsAlive, OneImageArrived,
MovieImageArrived
}
```

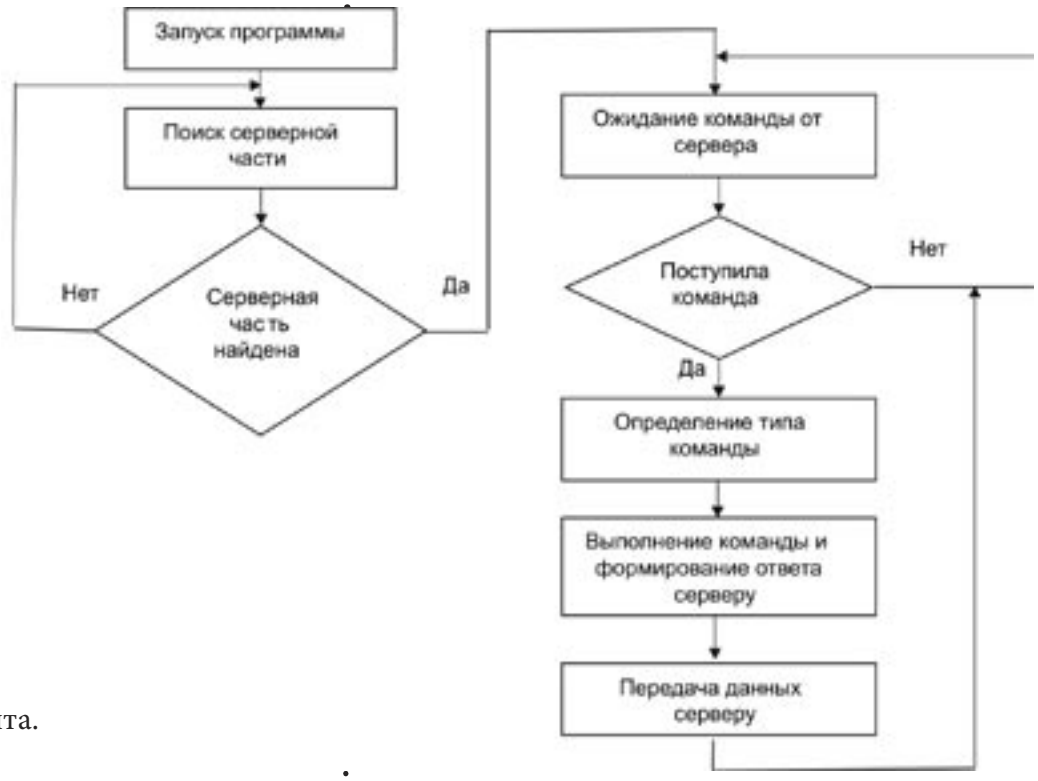

Рис. 1 Схема работы клиента.

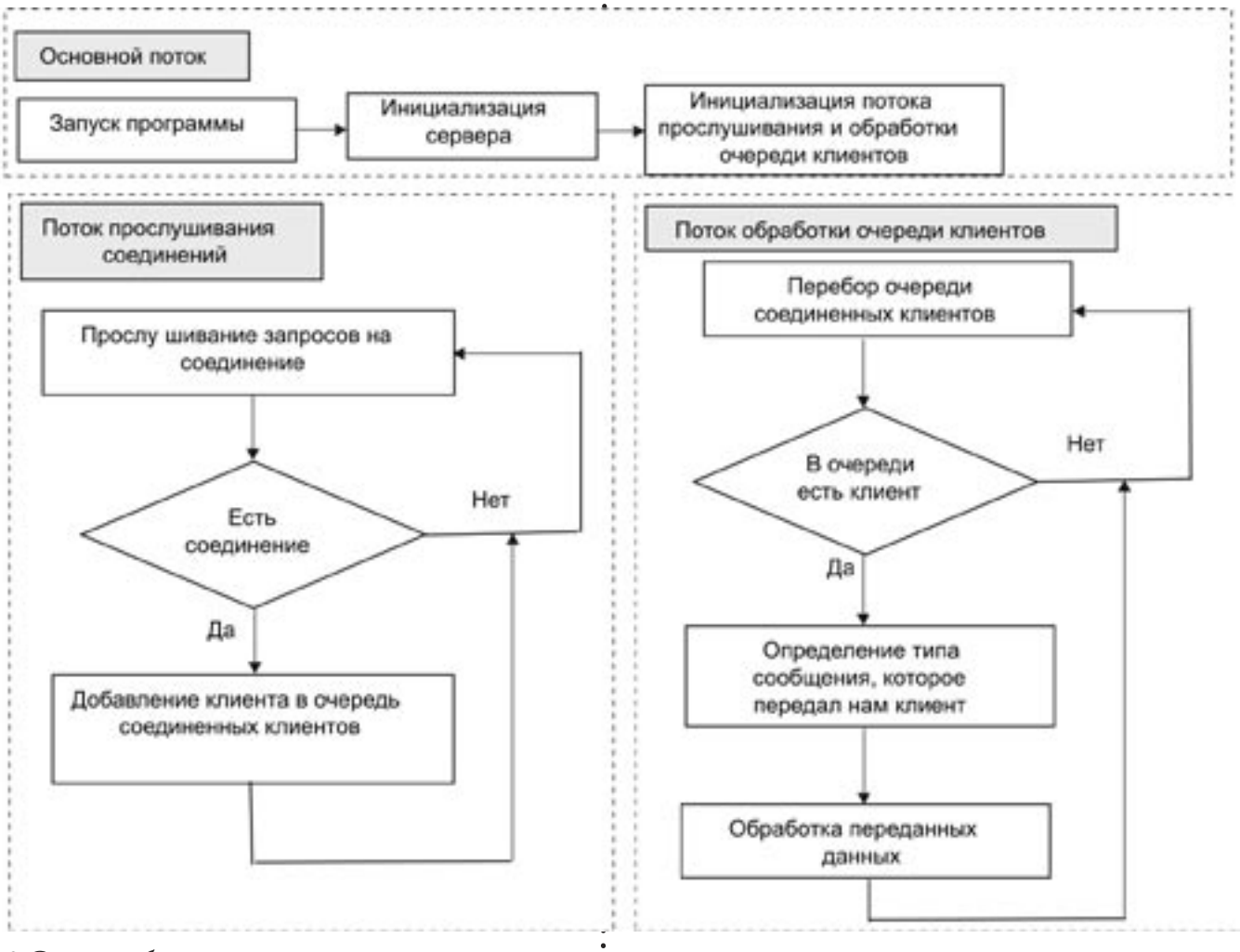

Рис. 2 Схема работы сервера.

Итак, определив типы сигналов, которыми будут обмениваться клиентская и серверная часть приложения, мы тем самым определили язык их общения. Теперь перед нами стоит задача наладить общение наших программ.
Прежде чем браться за конкретную реализацию следует спланировать главные узлы программ.
Например, для серверной части можно рассмотреть три сущности.

Первая – отвечает за получение сигналов от клиентов и оповещение об их получении.

Вторая – реагирует на полученные сигналы, обрабатывает входные данные (в нашем случае это будет изображение экрана), также может служить для передачи команд клиенту.

Третья – по сути, графическая оболочка, для взаимодействия с пользователем программы.

Для клиентской части программы, думаю, достаточно определить две сущности.

Первая – принимает команды от серверной части и оповещает об их получении.

Вторая – занимается непосредственно выполнением запрошенных команд и передачей результатов серверу, указывая тип послания.

Определим общие сущности, которые помогут нам реализовать как клиентскую, так и серверную часть. Общей может быть сущность, отвечающая за получение сигналов. Очевидно, что данный тип сущности должен опираться на объект типа TcpListener, так как нам необходимо прослушивать подключения и получать данные. Аналогично, сущность, отвечающая за отправку команд/результатов, опирается на класс TcpClient, так как нам необходимо пересылать данные.
Получается, что у нас понятия клиент и сервер немногоразмыты. Так, клиент, зарегистрировавшись на сервере, начинает прослушивание на получение команд от сервера, тем самым становясь сервером. В то же время сервер, передавая команду клиенту, становится на время клиентом, так как пытается соединиться с инициализированным слушателем команд на стороне клиентской части. Я, надеюсь, никого не запутал? :)

Теперь попробуем изобразить взаимодействие наших программ схематически.
Итак, теперь, надеюсь, картина более-менее прояснилась. Осталось только все это дело запрограммировать.

В данной части статьи остановимся только на налаживании общения наших программ, то есть реализовывать снятие изображений экрана и GUI для ее отображения сейчас мы не будем. Хоть и эта часть приложения тоже очень важна, так как, если вы еще не забыли, наша цель – подсмотреть, что делается на другом компьютере (например, у шефа) :)
Для начала реализуем класс, который позволил бы нам:
• прослушивать определенный порт;
• обрабатывать все соединения в отдельном потоке;
• добавлять всех соединенных клиентов в очередь для обеспечения последовательной обработки;
• в отдельном потоке вытягивать из очереди соединенных клиентов.

Класс должен реализовывать некий уровень абстракции, то есть он не должен знать ни о типах сигналов, которые передаются, ни вообще о том, что передается. На основе этого класса, в дальнейшем, мы реализуем сущности, которые отвечают за получение сигналов как со стороны серверной части программы, так и со стороны клиентской.
Вот приблизительный набросок основных членов этого класса:

```
namespace NetSpy
{
public class TcpListenerEx
{
/// <summary>
/// наш слушатель
/// </summary>
protected TcpListener _listner;
/// <summary>
/// очередь соединенных клиентов
/// </summary>
protected Queue _clients = new Queue();
/// <summary>
/// конструктор
/// </summary>
/// <param name="Port">
///прослушиваемый порт</param>
///<param name="Started">
```

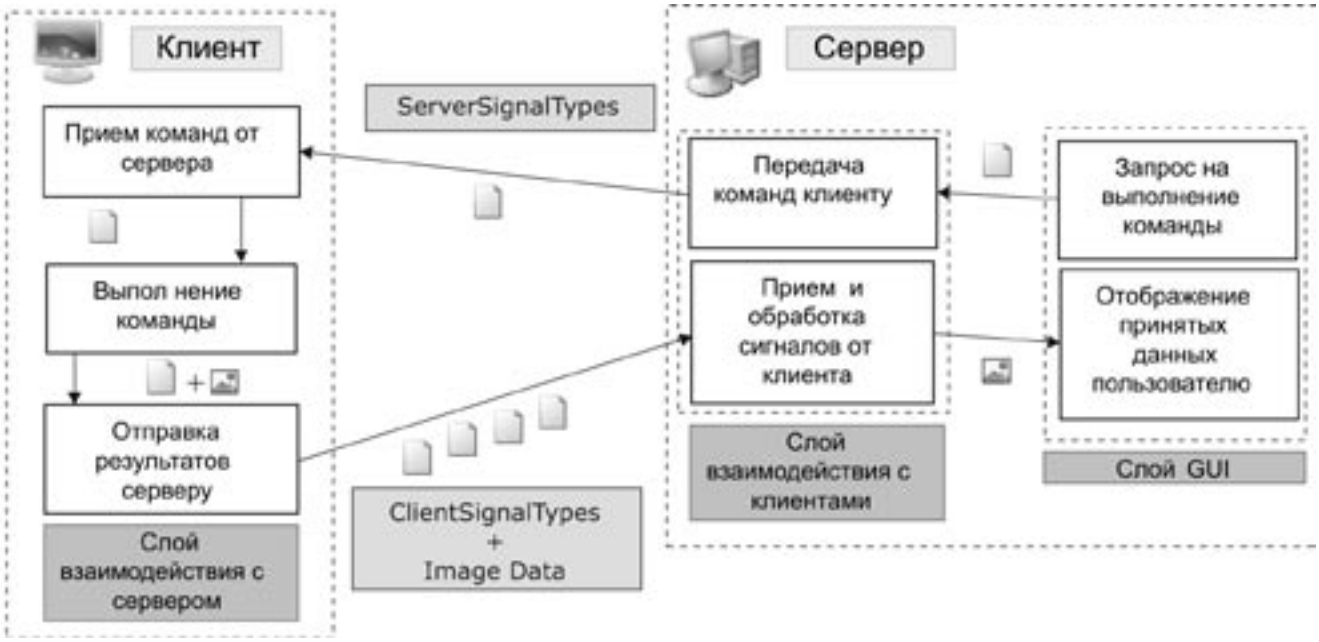

Рис. 3 Схематическое изображения взаимодействия клиента и сервера

```
///начать прослушивание</param>
public TcpListenerEx(int Port, bool
Started){}
/// <summary>
/// стартует сервер
/// </summary>
public void Start(){}
/// <summary>
/// остановка прослущивания
/// </summary>
public void Stop(){}
/// <summary>
/// эту функцию должен перекрыть
///наследник для
/// реализации своей
///собственной обработки клиента
/// </summary>
/// <param name="client">обеъкт клиента
///</param>
protected virtual void OnProcessClient(
TcpClient client){}
/// <summary>
/// функция потока проверки имеются
///ли ожидающие запросы на подключение
/// </summary>
/// <param name="target"></param>
private void Pending(object target){}
/// <summary>
/// функция потока обработки
///очереди клиентов
/// </summary>
private void ProcessClientQueue(
object target){}
}
```

```
}
```

Метод Start инициализирует прослушивание порта, переданного как параметр в конструктор класса. Кроме того, стартуют потоки проверки наличия ожидающих запросов на подключение и обработки очереди клиентов.

```
public void Start()
{
      Stop();
try
{
_listner = new
TcpListener(IPAddress.Any, _port);

// начинаем слушать клиентские запросы
_listner.Start();
// поток обработки соединений
ThreadPool.QueueUserWorkItem
(new WaitCallback(Pending));
// поток обработки очереди сигналов
ThreadPool.QueueUserWorkItem(
new WaitCallback(ProcessClientQueue));
}
catch (SocketException e){}
finally{}
}
```

Функция Pending в цикле проверяет наличие запросов на подключение, и если такие есть, то принимает их и добавляет в очередь соединенных клиентов.

```
private void Pending(object target)
{
try
{
//главный цикл, пока не будет
//остановлен сервер
while (_StopListening == false)
{
// принимаем клиента
if (_listner.Pending())
{
//добавляем клиента в очередь
//для последовательной обработки
//поступивших клиентов
lock (_clients) _clients.Enqueue(
_listner.AcceptTcpClient());
}
//делаем задержку
Thread.Sleep(_delay);
}
}
catch (Exception e){}
finally
{
_listner.Stop();
_StopListening = false;
}
}
```

Функция ProcessClientQueue просматривает очередь соединенных клиентов и, в порядке их поступления в очередь, вызывает виртуальную функцию OnProcessClient , логика которой должна быть определена в классах-наследниках.

```
private void ProcessClientQueue(
object target)
{
//главный цикл обработки очереди,
//пока TcpListener слушает
while (_StopListening == false)
{
TcpClient client = null;
lock (_clients)
{
if (_clients.Count > 0)
//получаем первого клиента в очереди
client = (TcpClient)
_clients.Dequeue();
}
//если клиент извлечен из очереди,
//то передаем управление
// в функцию обработки клиента
if (client != null)
OnProcessClient (client);
Thread.Sleep(_delay);
}
_StopListening = false;
}
```

Ну вот, по сути, основной функционал нашего класса TcpListenerEx заложен. Теперь мы умеем подключать к себе клиентов. Далее перед нами стоит задача реализовать конкретные сущности для каждой из частей программы.

MSDN

Событие представляет собой сообщение, посылаемое объектом, чтобы сигнализировать о совершении какого-либо действия. Это действие может быть вызвано в результате взаимодействия с пользователем, который, например, выполнил щелчок мышью, или же может быть обусловлено логикой работы программы. Объект, вызывающий (запускающий) событие, называется отправителем события. Объект, который захватывает событие и отвечает на него, называется получателем события.
При обмене событиями классу отправителя событий не известен объект или метод, который будет получать (обрабатывать) вызванные отправителем события. Необходимо, чтобы между источником и получателем события имелся посредник (или механизм подобный указателю). В среде .NET Framework определяется специальный тип (делегат), который предоставляет функциональные возможности указателя функции.
Делегат является классом, который может хранить ссылку на метод. В отличие от других классов класс делегата имеет подпись и может хранить ссылки только на методы, соответствующие этой подписи. Таким образом, делегат эквивалентен указателю функции безопасного типа или обратному вызову. Хотя делегаты имеют и другие направления использования, здесь будут рассматриваться только функциональные возможности делегатов по обработке событий.

Напомню, что сейчас мы реализуем сущности, которые предназначены только для получения сигналов и уведомления об этом.

Реализуем два класса ServerMessanger, ClientMes-

sanger. Оба будут наследоваться от класса TcpListenerEx и определять логику для выяснения типа сообщения и уведомления о его получении. ServerMessanger – берет на себя ответственность за получение ответов от клиентов, ClientMessanger – за получение команд от сервера.
Для того, чтобы уведомлять внешний мир о том, что у нас что-то произошло, был придуман очень удобный механизм, который называется – события. В .Net Framework события тесно связаны с делегатами.

Определим делегаты и события получения сигнала для классов ServerMessanger, ClientMessanger.

```
public class ServerMessanger :
TcpListenerEx
{
public delegate void SignalDelegate(
SignalEventArgs e);
public event SignalDelegate Signal;
       ...
}

public class ClientMessanger :
TcpListenerEx
{
public delegate void SignalDelegate(
ServerSignalTypes type);
 public event SignalDelegate Signal;
        ...
}
```

Делегату ServerMessanger.SignalDelegate в качестве параметра передается объект типа SignalEventArgs – это сделано для обеспечения более гибкого кода. Вот упрощенная реализация этого класса:

```
public class SignalEventArgs :
EventArgs
{

public SignalEventArgs(
ClientSignalTypes type, string
remoteAddress,
NetworkStream data){}

/// <summary>
/// тип сигнала
/// </summary>
public ClientSignalTypes SignalType
{get {}}
/// <summary>
/// адрес клиентсокой машины
/// </summary>
public string RemoteAddress
{get { }}
/// <summary>
/// поток данных, полученных с сервера
/// </summary>
public NetworkStream Data
{get{ }}
}
```

Теперь приступим к реализации функций обработки данных клиентов. Забыл упомянуть о формате пересылаемых данных - он прост до смешного. Первый байт полученного потока – это всегда тип сообщения, далее будут идти непосредственно передаваемые данные. То есть всё, что наши классы должны сделать, так это - получить поток данных, считать первый байт, определить тип сигнала, и уведомить о произошедшем внешний мир. Так и сделаем.

```
public class ServerMessanger :
TcpListenerEx
{
protected override void OnProcessClient
(TcpClient client)
{
lock (client)
{
//получаем данные
NetworkStream stream =
client.GetStream();
try
{
if (stream.DataAvailable)
{
//считываем первый байт из потока,
//для определения типа
//сигнала
ClientSignalTypes type =
(ClientSignalTypes) stream.ReadByte();
//оповещаем о получении сигнала

OnSignal(type, client, stream);
}
}
catch (Exception e)     {}
finally{}
}
/// <summary>
/// генерируем событие, которое
извещает ///подписчика о том,
/// что поступил сигнал от одного
///из клиентов
```

```
/// </summary>
///<param name="type">тип сигнала
///</param>
/// <param name="client">клиент</param>
protected void OnSignal(
ClientSignalTypes type, TcpClient
client, NetworkStream data)
{
if (Signal != null)
Signal(new SignalEventArgs(type,
Utils.GetClientAddress(client),data));
}
}
```

Для определения хоста присоединенного клиента мне пришлось воспользоваться технологией отражения, так как другого способа я не нашел. Функцию, которая это делает я вынес в отдельный класс.

```
public sealed class Utils
{
/// <summary>
/// полчение IP адреса клиента,
///с использованием рефлексии
/// </summary>
/// <param name="client">соединенный
///объект клиента</param>
/// <returns></returns>
public static string GetClientAddress(
TcpClient client)
{
//получаем защищенное свойство Client
PropertyInfo pi =
client.GetType().GetProperty ("Client",

BindingFlags.NonPublic |
BindingFlags.GetProperty |
BindingFlags.Instance );

Socket s =  (Socket)pi.GetValue (
client, new object[]{});

//вытягиваем адрес

string RemoteAddress =
s.RemoteEndPoint.ToString();
return RemoteAddress.Substring (
0, RemoteAddress.IndexOf (":"));
}
}
```

Реализацию класса ClientMessanger приводить не буду, так как она практически идентична выше описанной.

Ну что же, принимать сигналы мы научились, теперь нам необходимо их обработать и отреагировать должным образом. Классы XXXMessanger являются датчиками, которые регистрируют сигналы и уведомляют внешний мир о их получении. Опишем классы, которые будут должным образом реагировать на эти сигналы.

Класс ClientController берет на себя такие полномочия:

- использует класс ClientMessanger для получения уведомления о получении команд от серверной части;
- в зависимости от команды генерирует действия.

```
public class ClientController
{
/// <summary>
/// объект, через который
///будут приниматься
/// команды от сервера
/// </summary>
ClientMessanger _cm;

/// <summary>
/// конструктор
/// </summary>
///<param name="Host">имя хоста</param>
///<param name="OutPort">номер порта
///для отправки данных</param>
///<param name="InPort">номер порта
///для приема сигнала</param>

public ClientController(string Host,
int OutPort, int InPort)
{
      ...
_cm = new ClientMessanger(InPort);
_cm.Signal += new NetSpy.
ClientMessanger.SignalDelegate(
_cm_Signal);
...
}

public void Start()
{
//с самого начала пытаемся найти сервер
this.SendMessageIsAlive ();
//и в любом случае
//инициализируем таймер
//проверки подключения
_seekServerTmr.Start();

}
/// <summary>
/// обработчик команды от сервера
/// </summary>
```

```
/// <param name="type">тип команды,
///которая поступила</param>

private void _cm_
Signal(ServerSignalTypes type)
{
switch (type)
{
case ServerSignalTypes.GetImage:
//сервер попросил прислать одно
//изображение экрана
SendMessageGetImage();
break;
case ServerSignalTypes.StartMovie:
//сервер попросил прислать
///серию изображений
//экрана, запускаем таймер
_movieTmr.Start();
break;
case ServerSignalTypes.StopMovie:
//сервер попросил прекратить передачу
//серии изображений,
//останавливаем таймер
_movieTmr.Stop ();
break;
}
}
private void SendMessageGetImage()
{
SendMessageGetImage(
ClientSignalTypes.OneImageArrived);

}
/// <summary>
/// отправка изображения серверу
/// </summary>
/// <param name="type"></param>
private void SendMessageGetImage(
ClientSignalTypes type)
{
//инициализируем поток, в котором будем
//хранить наше изображение
MemoryStream imgStream = new
MemoryStream();
//делаем снимок экрана и сохраняем его
//в поток
ScreenShoter.ScreenShot().Save
(imgStream, ImageFormat.Jpeg );
//отправляем серверу данные
ResponseToServer(type ,
imgStream.ToArray());
}

/// <summary>
/// отправка ответа серверу
/// </summary>
```

```
/// <param name="message">
///тип сигнала</param>
/// <param name="data">данные</param>
/// <returns>true, если удалось
отправить ///</returns>
private bool ResponseToServer(
ClientSignalTypes message, byte[] data)
{
TcpClient client = new TcpClient();
try
{
//пытаемся соединиться с сервером
client.Connect(_host, _outPort);
//получаем поток для передачи данных
NetworkStream stream =
client.GetStream();
//в первый байт потока записываем
//тип сигнала
stream.WriteByte((byte)message );
//далее идут сами данные
stream.Write( data, 0, data.Length );
return true;
}
catch (Exception ex)
{
return false;
}
finally
{
client.Close();
}
}

private bool SendMessageIsAlive()
{
//пытаемся послать сообщение серверу
if(ResponseToServer(
ClientSignalTypes.IsAlive,new
byte[]{}))
{
//если отослать удалось, то
//инициализируем прослушивание
//команд от сервера
if(!_cm.Started)_cm.Start();
return true;
}
return false;
}

private void
_seekServerTmr_Elapsed(
object sender, ElapsedEventArgs e)
{
//останавливаем таймер, что бы избежать
//повторного входа
_seekServerTmr.Stop();
```

```
//если проверка связи с сервером была
//неудачной, то
//прекращаем передачу серии изображений
if(!SendMessageIsAlive())
_movieTmr.Stop();
//возобновляем работу таймера
_seekServerTmr.Start();
}

private void _movieTmr_Elapsed(
object sender, ElapsedEventArgs e)
{
//отправляем изображение серверу
SendMessageGetImage(
ClientSignalTypes.MovieImageArrived);
}
}
```

Класс ClientController создает экземпляр класса ClientMessanger и подписывается на его событие Signal. Метод Start запускает таймер, который с определенным интервалом проверяет связь с сервером. Если соединение удалось установить, то вызывается метод ClientMessanger.Start для инициализации прослушивания поступления команд от серверной части. Метод-обработчик события Signal в зависимости от типа поступившей команды выполняет определенные действия.

При запросе на получение изображения экрана, используется класс ScreenShoter и его метод ScreenShot. Реализацию этого методаприводить не буду, нам пока достаточно знать, что он возвращает объект типа Bitmap, который и есть наше изображение экрана. Метод ResponseToServer принимает в качестве аргумента тип сигнала и пересылаемые данные и отправляет эти данные серверной части в виде ответа на запрошенное действие. Если передача удалась, то возвращается true, иначе false.

Как мы видим, ничего сложного в реализации клиентской части нет, все, что нам осталось – так это инициализировать ClientController и мы можем начинать отвечать нашему серверу на запросы.

Подобный класс-контроллер для серверной части будет выглядеть немного сложнее, так как на него будет возложено немного больше обязанностей. Вот что нам нужно получить от этого класса:

- регистрировать новых клиентов, при поступлении нового клиента извещать об этом;
- следить за статусом присоединенных клиентов, при изменении статуса извещать об этом;
- на запрос пользователя посылать клиентам команды;
- принимать от клиентов ответы на запрошенные команды, извещать о получении команды и передавать полученные данные наружу для дальнейшей обработки.

Для более гибкого взаимодействия перенесем часть обязанностей класса ServerController на другой класс - ClientBinder. Объекты этого класса будут представлять интересы каждого отдельного клиента (неважно в каком состоянии он находится, главное, что бы он хотя бы один раз зарегистрировался на сервере). Этот класс будет предоставлять возможность отслеживать состояние каждого клиента в отдельности, а так же взаимодействовать с ним – посылать ему команды и принимать ответы.

Вот немного упрощенная реализация класса ServerController:

```
public class ServerController
{
/// <summary>
/// Событие, извещающее о том,
///что пришел ответ от
/// клиентской части
/// </summary>
public  delegate void
SignalArrivedDelegate(
SignalArrivedEventArgs e);

/// <summary>
/// Класс для передачи параметров
///в событие SignalArrivedDelegate
/// </summary>
public class SignalArrivedEventArgs:
EventArgs
{
public ClientBinder Client{get{}}

public  ClientSignalTypes  SignalType{
get{}}

public NetworkStream Data{get{}}
}
//ссылка на ServerMessanger
private ServerMessanger _sm;
//коллекция инициализированных клиентов
private Hashtable _clients =
new Hashtable();
```

```
/// <summary>
/// Событие - получен ответ от клиента
/// </summary>
public event
SignalArrivedDelegate SignalArrived;
/// <summary>
/// Событие - зарегистрирован
///новый клиент
/// </summary>
public event
SignalArrivedDelegate NewClient;

public ServerController( int Port )
{
_sm = new ServerMessanger( port );
_sm.Signal += new
ServerMessanger.SignalDelegate(
_sm_Signal );
}

public void Start()
{
_sm.Start();
}

  private void _sm_Signal(
ServerMessanger.SignalEventArgs e )
{
if(e.SignalType ==
ClientSignalTypes.IsAlive)

ProcessIsAlive(e);

OnSignalArrived(e);
}

private void ProcessIsAlive(
ServerMessanger.SignalEventArgs e)
{

if (!_clients.ContainsKey(
e.RemoteAddress  ) )
{
//если это первая связь с клиентом,
//то необходимо
//добавить объект для обратной связи
//с клиентом
//в коллекцию инициализированных
//клиентов _clients
ClientBinder cb = new
ClientBinder( e.RemoteAddress,  _port
);
lock(_clients)_clients.Add(
e.RemoteAddress, cb);
OnNewClient(e);
}

(ClientBinder)_clients[
e.RemoteAddress]).Pulse();
}
}
```

Как видно, класс ServerController использует класс ServerMessanger для получения сообщений от клиентов. Если тип сообщения инициализационный(ClientSignalTypes.IsAlive) , то идет проверка: нет ли уже зарегистрированного клиента (ключом служит хост клиента). Если такого нет, то создается объект-оболочка для данного клиента, то есть инициализируется новый объект класса ClientBinder.

А вот и класс ClientBinder, тоже в упрощенном виде:

```
public class ClientBinder
{
/// <summary>
/// таймер для периодической проверки
///соединения с клиентом
/// </summary>
Timer _getStatusTmr = new
Timer(DEF_PENDING_DELAY);
/// <summary>
/// делегат, для обеспечения
асинхронной
///передачи данных клиенту
/// </summary>
private delegate void
PostMessageDelegate(
ServerSignalTypes message,
int Tryes );

/// <summary>
//событие изменения статуса клиента
/// </summary>
public event
EventHandler StatusChanged;

public ClientBinder( string HostName,
int Port )
{
      ...
_getStatusTmr.Elapsed+=
new ElapsedEventHandler(
_getStatusTmr_Elapsed);
_getStatusTmr.Start();
}

public ClientStatus Status
{get{}}
```

```
public string Host
{get{}}

/// <summary>
/// переопределенная базовая функция
///для определения хэша по
/// хосту
/// </summary>
/// <returns>хэш</returns>
public override int GetHashCode()
{
return _host.GetHashCode();
}

/// <summary>
/// синхронная посылка
///сообщения клиенту
/// </summary>
public bool SendMessage(
ServerSignalTypes message, int Tryes )
{
return SendMessageInternal ( message,
Tryes);
}

/// <summary>
/// асинхронная отправка сообщения
/// </summary>
public IAsyncResult PostMessage(
ServerSignalTypes message, int Tryes )
{
return new PostMessageDelegate(
SendMessageInternalForDelegate).
BeginInvoke( message, Tryes, null, null
);
}


protected void
SendMessageInternalForDelegate(
ServerSignalTypes message,
int Tryes )
{
SendMessageInternal(message, Tryes);
}

/// <summary>
/// Отправка сообщения
/// </summary>
/// <param name="message">сообщение
///</param>
/// <param name="Tryes">кол-во
///попыток соединиться</param>
/// <returns></returns>
protected bool SendMessageInternal(
ServerSignalTypes message, int Tryes )
{

TcpClient client = new TcpClient();
try
{
//имеем Tryes попыток соединиться
for ( int i = 0; i < Tryes; i++ )
{
try
{
client.Connect( _host, _port );
//удалось соединиться, выходим из цикла
break;
}
catch
{
Debug.WriteLine(string.Format(
"Can't connect. {0} try.", i ) );
}
}
//пытаемся послать данные,
//возможно возникновение
//ошибки, если так и не удалось
//установить соединение
//либо поток не доступен для записи
client.GetStream().WriteByte(
(byte) message );
return true;
}
catch ( Exception ex )
{
Debug.WriteLine( "Ошибка при
посылке сообщения (SendMessage) "
+ ex.ToString() );
return false;
}
finally
{
client.Close();
}

}

protected void SetStatus(
ClientStatus NewStatus)
{ ...}

private void _getStatusTmr_Elapsed(
object sender, ElapsedEventArgs e)
{
      CheckConnection();
}

/// <summary>
/// проверка соединения
/// </summary>
private void CheckConnection()
{
```

```
SetStatus(ClientStatus.Pending);
//проверяем соединение
if(SendMessage(
ServerSignalTypes.IsAlive , 3))
{
SetStatus( ClientStatus.Online );
}
else //соединиться не удалось :(
{
SetStatus( ClientStatus.Offline );
//чтобы зря не расходовать ресурсы на
//отсоединенного клиента -
//выключаем таймер
//проверки соединения
_getStatusTmr.Stop();
}
}

public void Pulse()
{
_getStatusTmr.Start();
SetStatus( ClientStatus.Online );
}
}
```

Класс ClientBinder использует таймер, который с определенным интервалом вызывает проверку состояния соединения с клиентом. Этот таймер запускается непосредственно в конструкторе класса и останавливается лишь тогда, когда соединение с клиентом отсутствует. Если клиентская часть хочет возобновить связь с серверной, то ей необходимо вновь пройти регистрацию, но, при этом, новых объектов ClientBinder создаваться не будет. Вместо этого, достаточно вызывать функцию Pulse и старый объект ClientBinder опять будет следить за состоянием вновь подсоединенного клиента.

Состояние клиента в данный момент времени можно узнать из свойства Status типа ClientStatus. ClientStatus – это перечисление, описанное ниже.

```
public enum ClientStatus {Online, Offline,
Pending}
```

У класса ClientBinder есть функции для отправки команды клиенту в синхронном или асинхронном режиме. Синхронная передача предназначена для клиентов (имеется в виду код, который вызывает), которым нужно удостовериться в доставке или недоставке сообщения клиентской программе. Асинхронная передача - для тех, кому эта информация не важна, поэтому с использованием делегатов синхронная функция передачи сообщения вызывается асинхронно.
Как видно, для серверной части, все клиенты идентифицируются по хосту. Это немного неправильно, так как на одной клиентской машине может быть запущено несколько копий клиентской части программы, и все они должны распознаваться как разные соединения. Но наша программа этого делать не будет уметь - оставлю эту информацию к размышлению.

Ну что же, я считаю, что мы реализовали взаимодействие наших частей программы, клиентской и серверной, между собой – научили их общаться :) Теперь осталось дело за взаимодействием пользователя с нашей системой, но это уже тема для следующей статьи.

Что новенького...

Вышла экстремально оптимизированная математическая библиотека Extreme

Optimization Mathematics Library для .NET v1.1

Extreme Optimization выпустила версию 1.1 библиотеки Extreme Optimization Mathematics Library для .NET. Новая версия поддерживает кубические сплайны, линейную интерполяцию, модифицированные функции Бесселя и т.п.

Библиотека Extreme Optimization Mathematics для .NET включает классы для работы с комплексными числами, полиномами, классы оптимизации кривых, решения уравнений, числового интегрирования и дифференцирования. Также есть классы для работы с векторами, матрицами, проведения матричной декомпозиции. Код, использующий линейную алгебру, использует оптимизированные версии стандартных пакетов BLAS и LAPACK.
Бесплатная, полнофункциональная в течение 15 дней версия доступна на веб-сайте

http://www.extremeoptimization.com/

Visual Studio Beta 1 – первый взгляд на среду разработчика

Автор : В.Чужа

В статье описываются первые впечатления от работы с инструментом разработки программного обеспечения Visual Studio 2005. Статья не претендует на полный и всеохватывающий обзор этой среды, а предназначена лишь для тех, кому не посчастливилось ещё её увидеть.

Ставлю студию 2005 beta 1, свежескачанную из Интернета. После того, как я убрал несколько языков – C++, VB.Net и что-то ещё она затребовала «всего» 1.6 гигабайта на диске.

Послеединственной,необязательнойперезагрузки (видимо, она необходима для установки второй беты каркаса .Net), процесс спокойно пошёл дальше. А я налил себе бокальчик красного сухого вина и сел наблюдать за процессом.

Периодически сменяющаяся картинка вещала о преимуществах нового инструмента – более 50 новых элементов управления, возможность создания масштабируемых и – о боже! – управляемых данными интерфейсов приложений, переработанные инструменты для веб-дизайнеров, новые сервисы на веб-платформе, управляющие персонифицированием и членством, установка на веб-сервер одним кликом, визуальное моделирование кода, использование сниппетов кода в часто используемых задачах, визуальная работа с веб-сервисами – ожидание было приятным. Однако достаточно продолжительным – установка заняла около 20 минут на машине с 1 GB оперативной памяти и процессором Celeron 2.8 GHz. Не мало, однако, надеюсь, оно того будет стоить :)

Далее был получен так называемый Office Security Warning – вы его можете увидеть на рисунке 2. Говорилось в нём следующее:

Предупреждение о зависимости от Office

Вы успешно установили Microsoft Visual Studio Tools для Microsoft Office System, но для создания проекта у вас не установлены необходимые компоненты Office. Необходимо установить версию Microsoft Of-

Рис. 1 Начало установки VS.2005 Beta1

Рис. 2 Предупреждение о зависимости от Office

fice 2003, которая поддерживает Visual Studio Tools для Office. Также необходимо установить основные интероперабельные сборки Office 2003, а именно: Word, Excel, Forms, Graph.

Плюс ещё и насчёт безопасности предупредили:

Замечания по безопасности:

Настоятельно рекомендуем обновить ПК с помощью последних патчей для ОС, которые можно найти на сайте Windows Update, http://www.windowsupdate.com. Обновления для Windows 2000 доступны также по адресу http://www.microsoft.com/windows2000/downloads/default.asp, а для Windows XP - http://www.microsoft.com/windowsxp/pro/downloads/default.asp.

Проанализировать состояние безопасности компьютера можно с помощью утилиты Microsoft Baseline Security Analyzer (MBSA), адрес - http://www.microsoft.com/technet/treeview/default.asp?url=/technet/Security/tools/tools/MBSAHome.ASP. MBSA обладает графически интерфейсом и командной строкой, может производить как локальный, так и удалённый анализ систем Windows. MBSA работает под управлением Windows 2000 и Windows XP.

Если Microsoft Internet Information Services (IIS) установлен на вашей машине, установка автоматически зарегистрирует ASP.NET.

Если вы инсталлируете программу в каталог, отличный от каталога по умолчанию, или в раздел, имеющий файловую систему FAT, существует возможность доступа к вашим файлам со стороны других пользователей.

Далее предложили установить MSDN, который идёт в поставке со студией (рис.3), что и было успешно выполнено, однако заняло ещё около 10 минут (при выборе полной инсталляции – больше гигабайта на диске, хотя были и варианты). Ан нет, не обошлось без ошибки – memory could not be «written», что, однако, не помешало успешному завершению установки.

Посмотрим, что появилось у нас в главном меню. А появилось аж три пункта – Microsoft .Net Framework SDK 2.0, Microsoft Visual Source Safe, Microsoft Visual Studio 2005 Beta.

При старте студия предлагает выбрать настройки для среды – я, конечно, выбрал Visual C# Settings. При следующей загрузке попросили поучаствовать в программе тестирования (рис.5) – пришлось невежливо отказаться. Ну, куда мне с моим домашним Интернетом…

Ну что ж, пойдём дальше. Среда загрузилась, в глаза сразу бросилось оформление в стиле Office 2003. Теперь посмотрим, какие проекты мы можем создать с её помощью. На самом деле оказалось, что проектов просто несметное множество. Все они разбиты на 4 подменю – «Проект», «Веб-сайт», «Файл» и «Проект из существующего кода». С помощью первого подменю, мы можем создать Windows-приложения (собственно приложение Windows, библиотеку элементов управления Windows, консольное приложение, пустой проект, библиотеку классов, библиотеку элементов управления для веб, сервис Windows, а также приложение Crystal Reports – см. рис.6.), приложение для Office (приложение и шаблон приложения для MS Excel и Word), поддерживается создание приложений для Pocket PC 2003, Smartphone 2003, Windows CE, не забыты и проекты для баз данных. Другие типы доступных для создания проектов включают себя пакеты-инсталляторы, добавки к среде VS.Net и пустое решение для VS.Net. В целом, по сравнению с VS.Net 2003, возможностей гораздо больше, всё выглядит продуманнее и изящнее, что, несомненно, радует. К тому же, ко всем инсталлированным шаблонам

Рис. 3 Устанавливаем MSDN

можно добавить ещё и шаблоны, размещённые в Сети.

Веб-приложения

Далее. Обратим свой взгляд на веб-проекты (см. рис.7). Как вы помните, теперь не обязательно иметь установленный IIS на машине для того, чтобы разрабатывать веб-проекты. Дело в том, что в VS.Net 2005, которая, кстати, избавилась от приставки .Net и теперь называется просто Visual Studio 2005, имеется встроенный веб-сервер, которым и можно воспользоваться. Так вот, типы доступных по умолчанию проектов – веб-сайт ASP.Net, стартовый пакет для персонального веб-сайта, веб-сайт ASP.Net Crystal Reports, пустой веб-сайт и веб-сервис ASP.Net.

Следующая остановка – подменю «Файл» (см. рис.8). Тут мы видим, что в проект можно добавить текстовые файлы, файлы стилей, файл схемы XML, файл bmp, курсор, почему-то отдельно выделен класс Visual C#, логическую диаграмму данных, HTML-файл, файл XML, файл XSLT, файл иконки, шаблон ресурсов, диаграмму подсоединения приложения к данным, диаграмму системы.

Вы видите, какие обширные возможности предоставляет VS.2005 для разработчика. И последняя возможность – создание проекта на основе существующих файлов (см. рис.9), она пригодится, наверное, тогда, когда необходимо несколько частей одного проекта слить в один, более крупный, проект.

Ну что ж, настало время попробовать создать какой-нибудь проект. Начнем, пожалуй, с проекта веб-сайта ASP.Net, назовём его Test1Article. После создания проекта создался файл default.aspx, содержимое которого представлено в Листинге 1.

```
<%@ Page Language="C#"
CompileWith="Default.aspx.cs"
ClassName="Default_aspx" %>

<!DOCTYPE html PUBLIC "-//W3C//DTD
XHTML 1.1//EN" "http://www.w3.org/TR/
xhtml11/DTD/xhtml11.dtd">

<html xmlns="http://www.w3.org/1999/
xhtml" >
<head runat="server">
    <title>Untitled Page</title>
</head>
<body>
    <form id="form1" runat="server">
    <div>
    </div>
    </form>
</body>
```

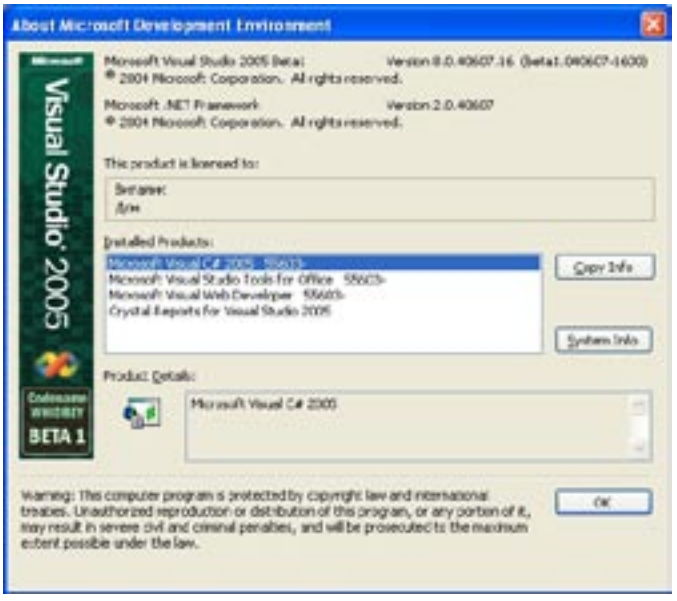

Рис. 4 Информация об установленном ПО

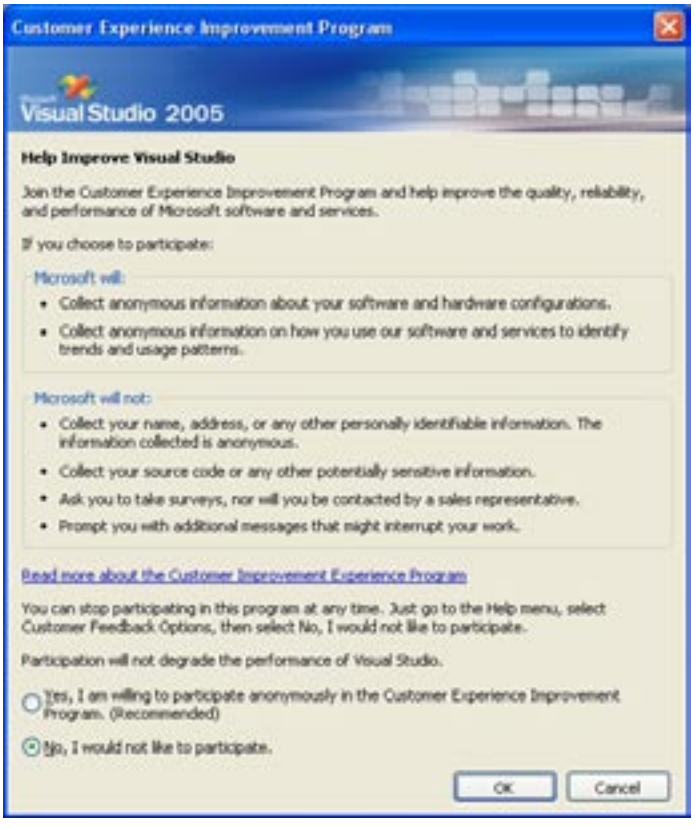

Рис. 5 Участвуем в программе тестирования?

```
</html>
```

Листинг 1. Содержимое файла default.aspx

Что сразу бросается в глаза?
Во-первых, атрибут CompileWith директивы Page. К сожалению, во встроенной документации описания этого атрибута нет. Отсюда можно сделать вывод, что это атрибут, специфичный для самой среды VS.2005, указывающий на то, что страница будет совместно компилироваться с одноимённым файлом кода C#. Далее можно заметить, что изменилась спецификация документа. Ранее тип документа указывался как <!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.0 Transitional//EN" >, что говорит о том, что от стандарта HTML 4.0 мы перешли к использованию стандарта XHTML 1.1, что указано также в тэге html, только несколько иным способом – с помощью атрибута xmlns. Далее бросается в глаза тэг head, который теперь исполняется на стороне сервера, что, скорее всего, говорит о том, что теперь мы сможем на стороне сервера менять свойства этого элемента. Например – заголовок страницы. Ранее, для того чтобы сделать это во время выполнения, необходимо было прибегнуть к некоторым ухищрениям. И последнее, что бросается в глаза – наличие тэга div в форме.

Перейдём к дизайну страницы default.aspx и щёлкнем на ней два раза для того, чтобы посмотреть на код-за-кулисами (см.Листинг 2.)

```
using System;
using System.Data;
using System.Configuration;
using System.Web;
using System.Web.Security;
using System.Web.UI;
using System.Web.UI.WebControls;
using System.Web.UI.WebControls.WebParts;
using System.Web.UI.HtmlControls;

public partial class Default_aspx
{void Page_Load(object sender,
```

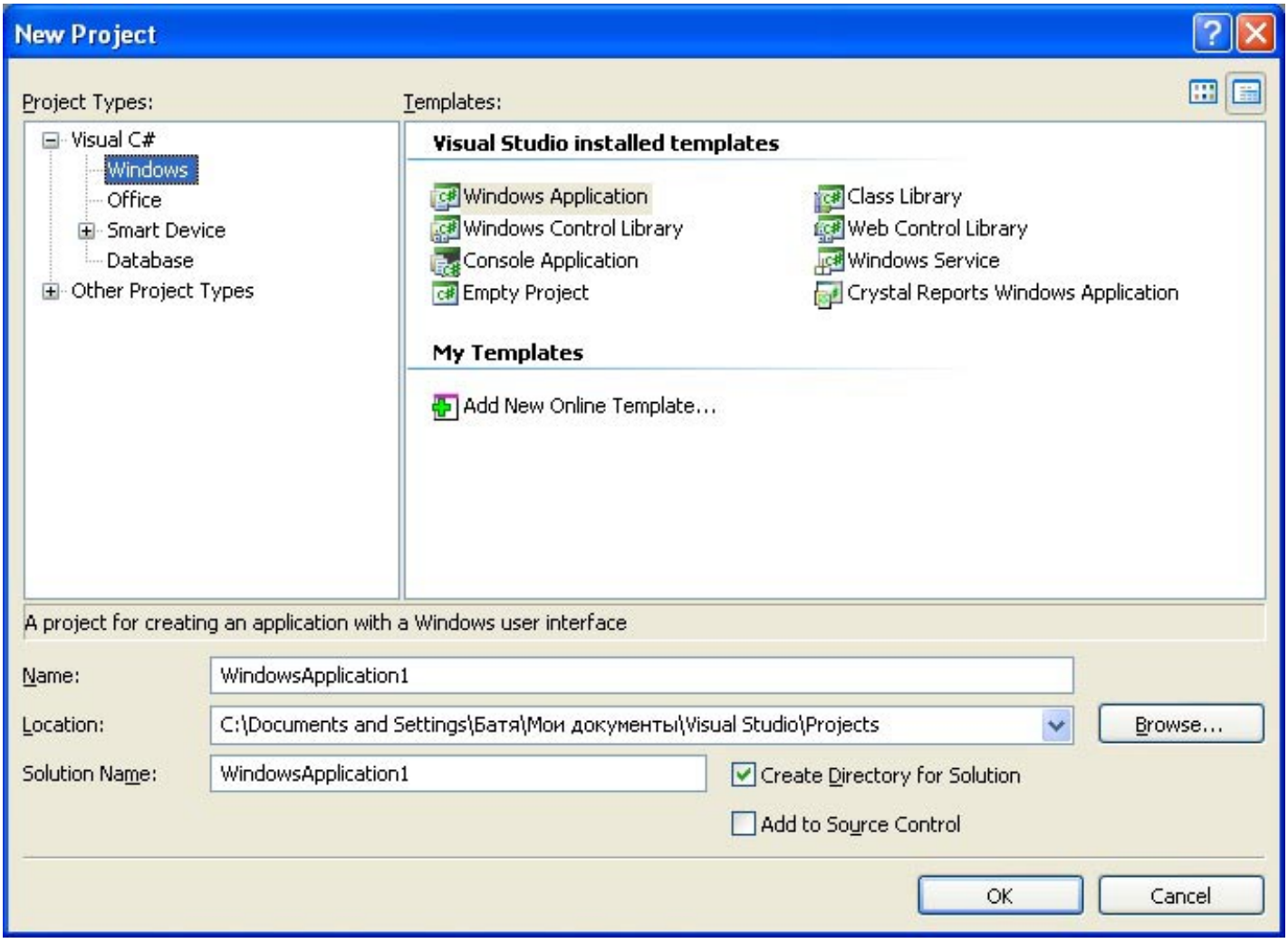

Рис. 6 Варианты Windows проектов

```
EventArgs e){}}
```

Листинг 2. Код-за-кулисами default.aspx.cs

Здесь по-прежнему сразу добавляется куча ненужных пространств имён, а также – частичный класс, нововведение второй версии каркаса .Net. Больше мы не увидим здесь различного кода, сгенерированного системой по умолчанию – только то, что мы написали сами. Из незнакомых пространств имён наблюдается только System. Web.UI.WebControls.WebParts.В документации сказано, что это пространство имён содержит интегрированный набор классов и интерфейсов для создания веб-страниц, чей вид и поведение может быть изменён (персонализирован) конечным пользователем. Установки, определённые пользователем для каждой страницы, сохраняются для последующих сессий.

Ну что ж, попробуем скомпилировать? После нажатия на F5 система выдала очень интересное предупреждение (см. рис.10) – «Отладка недоступна». Что удивило, так это, что это было именно предупреждение, а не сообщение об ошибке. Предупреждение с возможностью исправить ситуацию безотлагательно – запустить приложение в режиме релиза или добавить файл web.config с прописанной возможностью отладки веб-приложения. Выберем второе – заодно заглянём внутрь файла web.config. Запустился Visual Developer Web Server (см. рис.11), а затем запустилась и пустая страница. Ну и замечательно. Теперь взглянем на web.config – Листинг 3.

```
<?xml version="1.0"?>
<!-- Note: As an alternative to hand
editing this file you can use the web
admin tool to
configure settings for your application.
Use the Website->Asp.Net Configuration
option
in Visual Studio.
A full list of settings and comments
can be found in machine.config.comments
usually
located in \Windows\Microsft.Net\
Frameworks\v2.x\Config -->
```

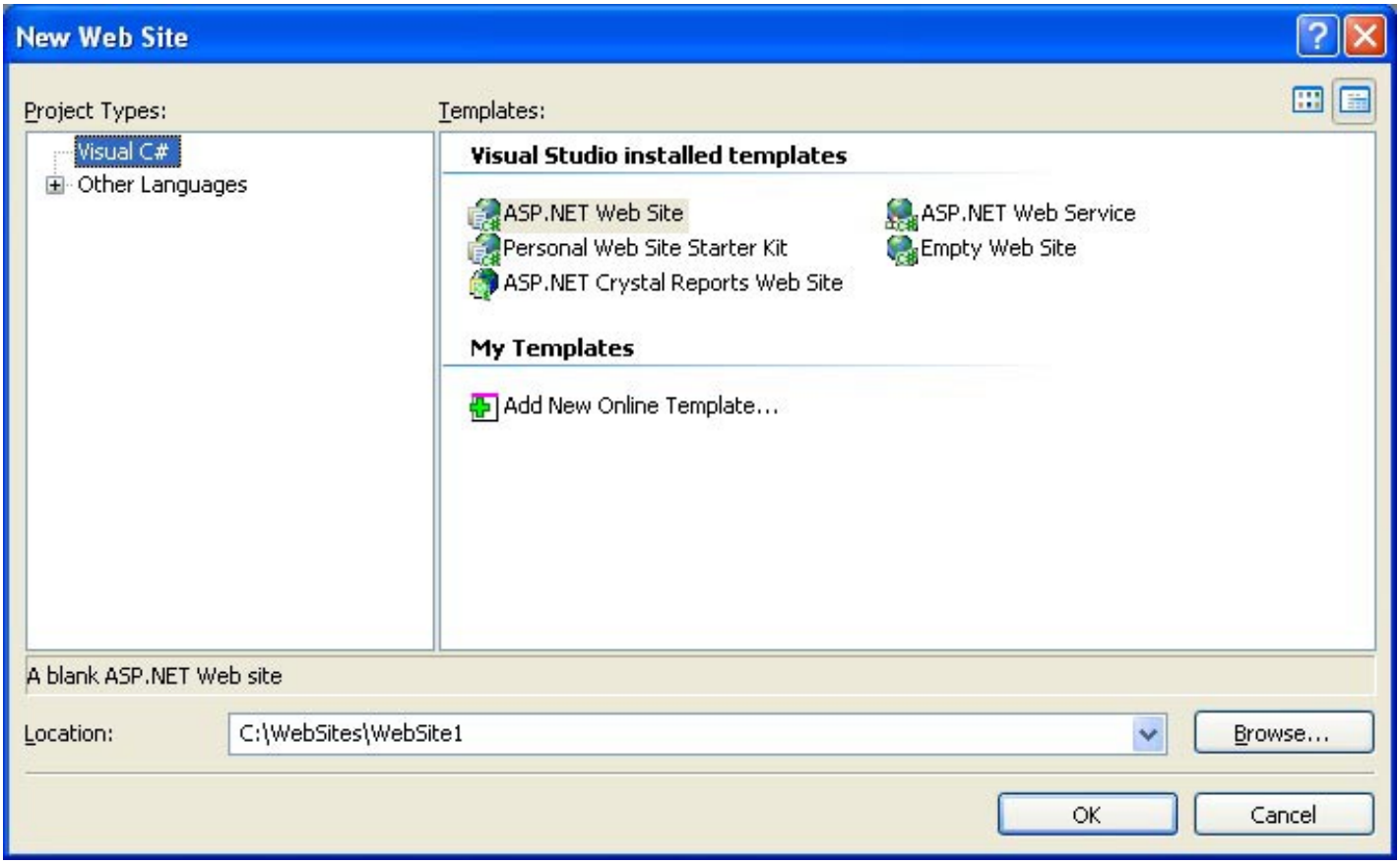

Рис. 7 Варианты веб-проектов

```
<configuration xmlns="http://schemas.
microsoft.com/.NetConfiguration/v2.0">
<appSettings/>
<connectionStrings/>

<system.web>
<!--
Set compilation debug="true" to insert
debugging symbols into the compiled
page.
Because this affects performance,
set this value to true only during
development.
-->

<compilation debug="true"/>
<!--
The <authentication> section enables
configuration of the security
authentication
mode used by ASP.NET to identify an
incoming user.
-->

<authentication mode="Windows"/>
<!--
The <customErrors> section enables
configuration of what to do if/when an
unhandled
error occurs during the execution of
a request.  Specifically, it enables
developers
to configure html error pages to be
displayed in place of a error stack
trace.
-->

<customErrors mode="RemoteOnly" default
Redirect="GenericErrorPage.htm">

<!--
<error statusCode="403"
redirect="NoAccess.htm"/>
<error statusCode="404"
redirect="FileNotFound.htm"/>
```

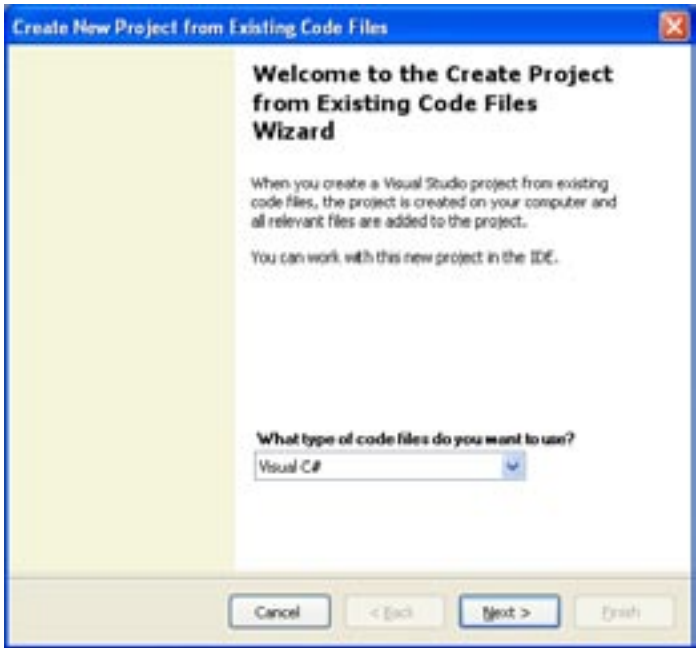

Рис. 9 Волшебник создания проектов из существующего кода

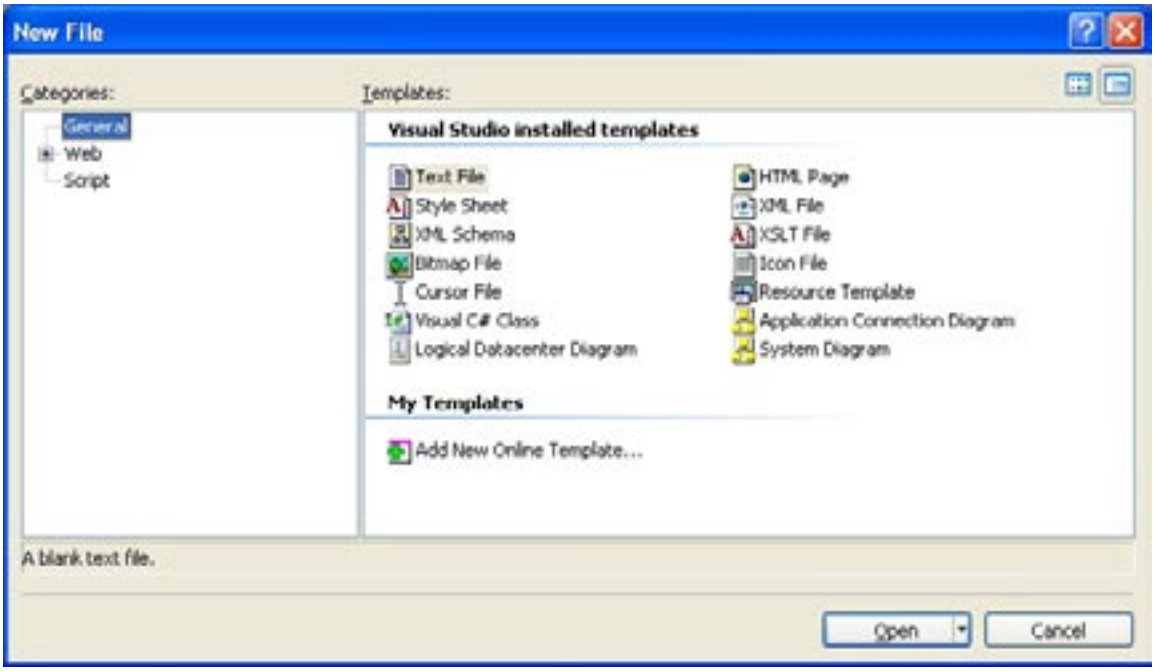

Рис. 8 Варианты файлов

```
-->

</customErrors>
</system.web>
</configuration>
```

Листинг 3. Содержимое web.config.

Очень интересное предложение воспользоваться для альтернативного редактирования файла конфигурации неким встроенным инструментом. Пожалуй, попробуем. Оп-па!!! Да тут целая куча возможностей! Кстати, по умолчанию у меня стоит FireFox версии 1.0.2, пока что никаких возмущений со стороны студии по этому поводу не было. Итак, что же тут можно сделать? Можно пойти по ссылке Security (Безопасность), которая предоставляет возможность добавлять и редактировать пользователей, роли, разрешения по доступу к сайту. Следующая ссылка – Profile (Профиль), позволяющая управлять набором свойств профиля, значения которого хранятся отдельно для каждого пользователя сайта. Ссылка Application Configuration приведёт нас к странице управления конфигурационными настройками приложения. А ссылка Provider Configuration позволит указать, где и как хранятся административные данные, используемые нашим веб-сайтом. В общем, после ручного конфигурирования – выше всяких похвал. Хотя, я уверен, что в итоге без него всё равно не обойтись. Что ещё бросается в глаза – предустановленные страницы, на которые может перенаправляться пользователь при обработке ошибок 403 и 404. Такая предусмотрительность приятно радует глаз :)

Теперь вскользь взглянем на панель инструментов

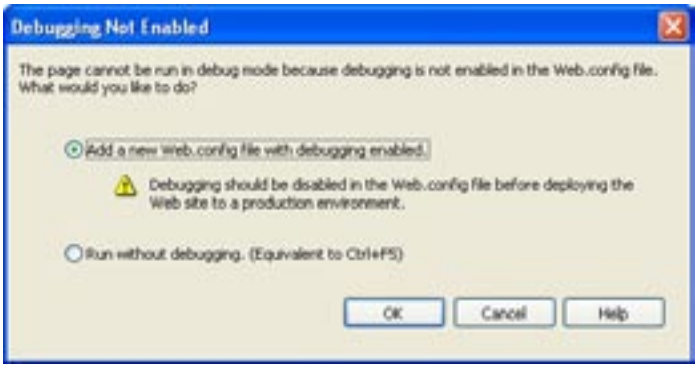

Рис. 10 Автоматическое добавление конфигурационного файла

и перейдём к другому типу проектов. Подробнее разбираться с каждым из них будем в последующих, специально посвящённых для этих целей, статьях. Так, вот. Если вы помните, то все компоненты в VS.2003 были размещены на закладках Data, WebForms и HTML. Теперь же их стало больше – первая закладка называется Standart, наиболее примечательные компоненты на ней – это FileUpload, PhoneLink, View и Substitution. Далее старая закладка Data – GridView, DetailsView, FormView, XMLDataSource, SiteMapDataSource. На отдельную закладку Validation были вынесены компоненты проверки пользовательского ввода. На закладке Navigation всего три компонента – Pointer, SiteMapPath и Menu. Далее идёт закладка Login с компонентами, отвечающими за авторизацию пользователя, закладка WebParts, соответствующее пространство имён мы уже немного обсуждали, закладка Crystal Reports с компонентами для вывода отчёта и последняя закладка HTML с соответствующими компонентами. На этом краткий обзор веб-приложений закончим и перейдём к Windows-приложениям.

Windows-приложения

Закроем предыдущий проект и создадим новое приложение из подменю Projects – Windows Application. Назовём его Test2ArticleWinApp. После создания проекта мы можем увидеть в Solution Explorer'е набор файлов, изначально составляющих проект. Теперь к проекту автоматически добавляется папка Properties, в которой хранятся файлы AssemblyInfo.cs и Resources.resx. В главном каталоге проекта, кроме файла формы Form1.cs, теперь находится и файл Program.cs. В нём находится собственно текст программы, который выглядит так – см. Листинг 4.

```
#region Using directives

using System;
using System.Collections.Generic;
using System.Windows.Forms;

#endregion

namespace WindowsApplication1
{
static class Program
{
/// <summary>
```

```
/// The main entry point
///for the application.
/// </summary>
[STAThread]
static void Main()
{
Application.EnableVisualStyles();
Application.EnableRTLMirroring();
Application.Run(new Form1());
}
}
}
```

Листинг 4. Текст программы приложения Windows

А текст первой формы выглядит так, как показано в Листинге 5.

```
#region Using directives

using System;
using System.Collections.Generic;
using System.ComponentModel;
using System.Data;
using System.Drawing;
using System.Windows.Forms;

#endregion

namespace WindowsApplication1
{
partial class Form1 : Form
{
public Form1()
{
InitializeComponent();
}
}
}
```

Листинг 5. Код Form1.cs

Рис. 11 Встроенный веб-сервер

Как вы видите, он существенно отличается от того кода, который генерировала VS.2003 по-умолчанию – смотрим Листинг 6.

```
using System;
using System.Drawing;
using System.Collections;
using System.ComponentModel;
```

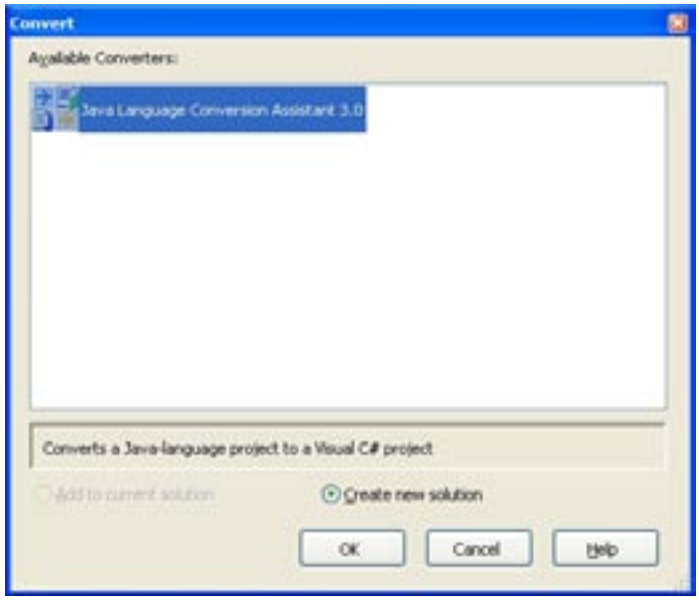

Рис. 12. Конвертор Java-проектов

```
using System.Windows.Forms;

namespace Test.VS.2003
{
/// <summary>
/// Summary description
///for Form0.
/// </summary>
public class Form0 : System.Windows.
Forms.Form
{
/// <summary>
/// Required designer variable.
/// </summary>
private System.ComponentModel.Container
components = null;

public Form0()
{
//
// Required for Windows
//Form Designer support
//
InitializeComponent();
```

```
//
// TODO: Add any constructor code
//after InitializeComponent call
//
}

/// <summary>
/// Clean up any resources
///being used.
/// </summary>
protected override void Dispose( bool
disposing )
{
if( disposing )
{
if(components != null)
{
components.Dispose();
}
}
base.Dispose( disposing );
}

#region Windows Form Designer generated
code
/// <summary>
/// Required method for Designer
support - do not modify
/// the contents of this method with
the code editor.
/// </summary>
private void InitializeComponent()
{
this.components = new System.
ComponentModel.Container();
this.Size = new System.Drawing.
Size(300,300);
this.Text = "Form0";
}
#endregion
}}
```

Листинг 6. Код, сгенерированный по умолчанию для приложения Windows VS.2003

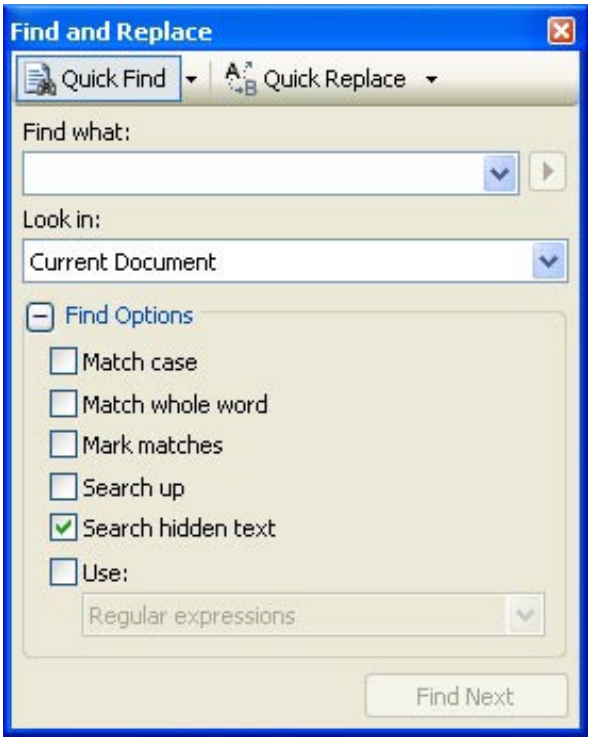

Рис. 13. Диалог поиска и замены

Нужно признать, что код, сгенерированный VS.2005 выглядит менее захламленным и более понятным. Теперь обратим внимание на закладку с компонентами, а затем рассмотрим инструменты, призванные упростить работу разработчика. Если помните, в VS.2003 компоненты были разбиты на три закладки – Data, Components и Windows Forms.

Теперь же добавилась закладка под названием Crystal Reports. На «старых» закладках заметны такие новые компоненты как DataGridView, DataConnector, DataNavigator, SerialPort, PerfomanceCounter, DataConnector, MaskedTextBox. По названиям многих из них можно догадаться о выполняемой функциональности. Впрочем, в одной из следующих статей мы поговорим о ней подробнее.

Посмотрим на набор инструментов, доступных для облегчения нашей непростой жизни.

Первый инструмент – конвертер проектов Java в приложение C# - в меню File-Open-Convert. Ну что ж, вполне достойное предложение для тех, кто хочет перейти на новую платформу (см. рис.12).

Существенно переработана функция поиска и замены в решении и проектах. Теперь доступен быстрый поиск, быстрая замена, поиск и замена в файлах, поиск символа (см. рис.13) – в целом всё стало более функциональным и удобным. Доступно через меню Edit-Find and Replace.

Появилось окно просмотра ошибок, предупреждений и сообщений – доступно через меню View-Error List.

В режиме просмотра кода доступны два новых пункта меню – Intellisense и Refactor. Функции, доступные посредством задействования данных технологий, уже не раз освещались в различных статьях. Поэтому я просто их перечислю – вставка конструкции или окружение конструкцией участка кода, создание пустой реализации метода

при нажатии на его имя, просмотр списка членов данного пространства имён, класса и т.д., просмотр информации о параметрах метода, быстрая информация о методе, классе и т.п., вызов функции завершения оператора, имени или типа переменной и т.п. Всё это функции Intellisense, доступные посредством меню или через горячие клавиши. Функции, доступные через встроенный рефакторинг таковы: экстракция метода, переименование метода, переменной и т.п., инкапсуляция поля, экстракция интерфейса, преобразование внутренней переменной в параметр, удаление и изменение порядка следования параметров. В целом можно сказать одно – необходимый минимум есть, но функциональность куда беднее, чем у уже ставшего привычным в VS.Net 2003 Resharper'a.

Пойдём далее. В меню «Проект» появилось подменю свойств проекта с массой настроек – все они вынесены в отдельную страницу с закладками. Закладок всего десять. Первая из них – Приложение (Application). Здесь можно установить имя сборки, имя корневого пространства имён, тип приложения – консольное, обычное windows, ,библиотека классов, устанавливается стартовый объект, иконка приложения, даётся информация о каталоге проекта, имени файла в который он будет скомпилирован и имени самого проекта. Следующая закладка – Подпись (Signing). С её помощью можно подписать сборку – файл ключа можно создать здесь же или использовать уже готовый.

Третья закладка – Reference Paths или Ссылочные пути. В ней можно добавить пути, по которым приложение будет искать сборки или файлы ресурсов.

Четвёртая – Build Events или События постройки. Здесь можно указать некие действия, которые будут выполняться перед компиляцией проекта и после завершения компиляции, причём посткомпиляционные действия можно выполнять или не выполнять в зависимости от успешности завершения самой компиляции.

Пятая закладка – Security или Безопасность. В ней можно настроить степень доверия коду, т.е. указать, какими правами будет обладать компилируемая сборка. Вариантов четыре – локальный компьютер (все права), локальный интранет (прав поменьше), Интернет (минимум прав) и пользовательская настройка – раздача прав на ваше усмотрение.

Шестая закладка – Publish или Публикация. С её помощью можно настроить каталоги, в которых будет перемещаться приложения для запуска конечным пользователем. Здесь же можно задать список файлов для публикации, набор приложений, необходимых для корректной работы нашего проекта (например, наличие MDAC 2.8), возможность проверки приложением обновлений, а также присутствует волшебник публикации, существенно облегчающий сам процесс публикации.

Далее идёт закладка Build – Постройка. Здесь можно указать целевую платформу – x86, x64, Itanium или любой ЦПУ, разрешить использование неуправляемого кода, включить оптимизацию кода, отредактировать список условных слов – таких как DEBUG и TRACE, установить путь размещения откомпилированных программ, файла документации в формате XML и многое другое.

На восьмой закладке – Debug (Отладка) можно установить опции для режимов компиляции – отладки, релиза или для всех сразу. Настроить можно действие при старте компиляции в данном режиме, указать настройки при старте, а также разрешить использование *отладчиков при отладке неуправляемого или SQL кода.*

Девятая закладка является редактором ресурсов, поэтому так и называется – Resources(Ресурсы).

На десятой же можно отредактировать профиль – как приложения, так и пользователя. Тоже вещь, достойная отдельного исследования.

Как видно из этого небольшого обзора, настройкам Windows-приложения уделено очень много внимания, все они теперь сконцентрированы в одном месте и позволяют гибко реагировать на требования к разработке программ.

В меню Tools появилось подменю Import|Export Settings (Импорт/Экспорт Настроек), теперь не придётся каждый раз, при новой установке студии, все настройки создавать заново. Более богатыми стали и настройки в подменю Options, они также достойны отдельного, более внимательного взгляда.

Кстати…

В ASP.Net 2.0 появилась новая возможность компиляции – прекомпиляция проекта. Что это такое и зачем она нужна? Прекомпиляция (или предварительная компиляция) призвана бороться с задержкой отображения страницы при первом обращении к ней – эта задержка относительно велика и присутствовала в предыдущих версиях ASP.Net. Некоторые программисты писали специальные утилиты, обращавшиеся «первый раз» ко всем страницам проекта для устранения эффекта задержки. Также прекомпиляция поможет вам в том случае, если вы не хотите выкладывать исходные коды в Сеть – в этом случае весь проект можно скомпилировать в одну библиотеку и выложить только её (да -да, без файлов .aspx!). Прекомпиляция выполняется с помощью утилиты aspnet_compiler.exe, размещённой в каталоге %windir%\Microsoft.NET\Framework\version.

Заключение

В целом Visual Studio 2005 даже в виде первой беты производит очень благоприятное впечатление – стабильно и быстро работает, обладает мощной и многофункциональной средой для написания программного обеспечения для самых разных целей и платформ. Также особое внимание уделено встроенной помощи, которая также обзавелась некоторыми новшествами – например, появилось «Избранное» и «Как мне сделать…». Настораживает только одно – по пока ещё непроверенным слухам вторая бета работает куда медленнее и куда менее стабильна в работе. Также было бы совсем неплохо, если бы с помощью студии можно было разрабатывать программы, использующие Mono, альтернативную реализацию платформы .Net с открытым исходным кодом. Однако, так или иначе, этот продукт наверняка будет лучшим инструментом разработчика в индустрии как минимум несколько лет – поэтому, начав его изучение сейчас, вы обеспечиваете себе более стабильное будущее, соответствующее духу времени.

Что новенького…

Вышел конвертор VB.Net на C# 1.3

В новой версии 1.3 улучшена аккуратность конвертации, онлайновая помощь, появилась версия для командной строки для пакетных конвертаций. Конвертор от VBConversions VB.Net на C# может конвертировать индивидуальные проекты, фрагменты кода и даже группы проектов в случае необходимости массовой конвертации.

Версия 1.3 успешно конвертировала 650,000 строк публично доступного кода примеров на VB.Net, включая Microsoft 101 Пример на VB.Net (версий 2002 и 2003 года), Microsoft Enterprise Library, а также сотни примеров программ третьих производителей.

Как уже говорилось, версию 1.3 отличают улучшенная аккуратность конвертации, онлайновая помощь и уроки, а также версия для командной строки для пакетных конвертаций. Ваш код проверяется на потенциальные проблемы с конверсией перед самой конверсией и новые Conversion Notes (Заметки о конверсии) расскажут вам о происходящем во время конверсии, чтобы вы были в курсе событий.
Бесплатная пробная версия доступна на веб-сайте

http://www.vbconversions.com/

- она не ограничена по времени работы и никакой необходимости в регистрации тоже нет.

Вышел Likemedia ClassBuilder - версия 3.7

Новая версия известного генератора кода. Самый простой в использовании генератор кода доступа к данным стал ещё лучше после редизайна стратегии обработки ошибок. Теперь исключения проходят сквозь слои кода, сообщая вам о том, где они реально возникли. Отладка становится настоящим удовольствием, когда вы знаете, где искать ошибку! Скачайте бесплатную пробную версию на сайте

http://www.likemedia.com/

Что должен знать правильный .NET-разработчик

Автор: С.Хансельманн
Перевод: Н.Зимин

Каждый кто пишет код

• *Объясните разницу между нитью (Thread) и процессом (Process)*

• *Что такое сервис (Windows Service) и как его жизненный цикл отличается от «стандартного» EXE?*

• *Какой максимальный объем памяти может адресовать один процесс? Отличается ли он от максимального объема виртуальной памяти, доступной системе? Как это влияет на структуру системы?*

• *В чем различие между EXE и DLL?*

• *Что такое строгая типизация (strong-typing) в сравнении со слабой типизацией (weak-typing)? Какая предпочтительнее? Почему?*

• *Некий продукт называют «контейнером компонентов» ("Component Container"). Назовите по крайней мере 3 контейнера компонентов, поставляемых с семейством продуктов Windows Server Family.*

• *Что такое PID? Чем он полезен при выявлении неисправностей системы?*

• *Сколько процессов могут слушать один и тот же порт TCP/IP?*

• *Что такое GAC? Какую проблему он разрешает?*

.NET-разработчик среднего уровня

• *Объясните разницу между интерфейсно ориентированным (Interface-oriented), объектно ориентированным и аспектно ориентированным (Aspect-oriented) программированием*

• *Объясните что такое «интерфейс» и чем он отличается от класса*

• *Что такое Reflection?*

• *В чем различие между XML Web Services с использованием ASMX и .NET Remoting с использованием SOAP?*

• *Являются ли системы типов, представленные в XmlSchema и в CLS — изоморфными?*

• *Концептуально, в чем различие между ранним и поздним связыванием (early-binding и late-binding)?*

• *Использование Assembly.Load — это статическая или динамическая ссылка?*

• *Когда уместно использование Assembly.LoadFrom, а когда Assembly.LoadFile?*

• *Что такое «Asssembly Qualified Name»? Это имя файла? В чем различие между ними?*

• *Правильно ли так писать?*
Assembly.Load("foo.dll");

• *Чем отличается «strongly-named» сборка от «НЕ strongly-named» сборки?*

• *Может ли DateTime равняться null?*

• *Что такое JIT? Что такое NGEN? Каковы преимущества и ограничения каждого из них?*

• *Как основанный на поколениях сборщик мусора в .NET CLR управляет жизненным циклом объекта? Что такое «non-deterministic finalization»?*

• *В чем различие между Finalize() и Dispose()?*

• *Чем полезен using()? Что такое IDisposable? Как*

он поддерживает deterministic finalization?

• Что делает эта полезная команда?
tasklist /m "mscor"*

• В чем разница между «in-proc» и «out-of-proc»?

• Какая технология позволяет выполнять out-of-proc взаимодействие в .NET?

• Когда вы запускаете компонент из под ASP.NET, в каком процессе он работает под Windows XP? Windows 2000? Windows 2003?

Ведущий разработчик

• Что не так вот в следующей строке?
DateTime.Parse(myString);

• Что такое PDB? Где они должны находится, чтобы можно было выполнять отладку?

• Что такое «цикломатическая сложность» (cyclomatic complexity) и почему она важна?

• Напишите стандартный lock() плюс «двойную проверку» для создания критической секции вокруг доступа к переменной.

• Что такое «FullTrust»? Имеют ли FullTrust сборки, помещенные в GAC?

• Какие преимущества получает ваш код, если вы декорируете его атрибутами, относящимися к особым Security permissions?

• Что делает эта команда?
gacutil /l | find /i "Corillian"

• Что делает эта команда?
sn -t foo.dll

• Какие порты брандмауэра должны быть открыты для DCOM? Каково назначение порта 135?

• Сопоставьте OOP и SOA. Каковы принципы каждого из них?

• Как работает XmlSerializer? Каких ACL permissions требует использующий его процесс?

• Почему catch(Exception) почти всегда — плохая мысль?

• В чем разница между Debug.Write и Trace.Write? Когда должен быть использован каждый из них?

• В чем различие между компиляцией в Debug и в Release? Есть ли значительная разница в скорости? Почему или почему нет?

• Как работает JIT — по сборке целиком или по методу? Как это влияет на working set?

• Сравните использование абстрактного базового класса и использование интерфейса?

• В чем различие между a.Equals(b) и a == b?

• В контексте сравнения, что такое идентичность объектов по сравнению с эквивалентностью объектов?

• Как можно выполнить глубокое копирование (deep copy) в .NET?

• Изложите ваше понимание IClonable.

• Что такое «упаковка» (boxing)?

• string — это тип значений (value type) или ссылочный тип?

• В чем значимость паттерна "PropertySpecified", используемого в XmlSerializer? Какую проблему он пытается разрешить?

• Почему в .NET выходные параметры (out parameters) не стоит применять? Действительно ли это так?

• Может ли атрибут быть установлен на один из параметров метода? Чем это полезно?

Разработчик компонентов на C#

• Сопоставьте использование override и new. Что такое «shadowing»?

• Объясните использование virtual, sealed, override и abstract.

• *Объясните использование и значение каждого компонента строки:*
Foo.Bar, Version=2.0.205.0, Culture=neutral, PublicKeyToken=593777ae2d274679d

• *Объясните различия между public, protected, private и internal.*

• *Какое преимущество вы получаете от использования первичной сборки взаимодействия (Primary Interop Assembly, PIA)?*

• *Благодаря какому механизму NUnit узнает, какой метод протестировать?*

• *В чем различие между:*
catch(Exception e){throw e;} и
catch(Exception e){throw;}

•*В чем разница между typeof(foo) и myFoo.GetType()?*

• *Объясните что происходит в первом конструкторе:*
public class c{ public c(string a) : this() {;};
public c() {;} }
Чем полезна такая конструкция?

• *Что такое «this»? Может ли this использоваться в статическом методе?*

Разработчик на ASP.NET (UI)

• *Объясните, как POST-запрос формы из браузера становится на серверной стороне событием — таким как Button1_OnClick.*

• *Что такое «PostBack»?*

• *Что такое «ViewState»? Как он кодируется? Является ли он шифрованным? Кто использует ViewState?*

• *Что такое <machinekey> element и для чего используются эти две технологии ASP.NET? (В оригинале: What is the <machinekey> element and what two ASP.NET technologies is it used for?)*

• *Какие три Session State providers доступны в ASP.NET 1.1? Какие преимущества и недостатки у каждого из них?*

• *Что такое «Web Gardening»? Как его использование влияет на проект?*

• *В заданном ASP.NET-приложении, сколько объектов-приложений имеется, если это однопроцессорная машина? двухпроцессорная? двухпроцессорная с включенным Web Gardening? Как это отражается на проекте?*

• *Используются ли нити (threads) ASP.NET приложения повторно для различных запросов (requests)? Получает ли каждый HttpRequest свою собственную нить? Должны ли вы в ASP.NET использовать Thread Local storage?*

• *Полезен ли атрибут [ThreadStatic] в ASP.NET? Есть ли побочный эффект? Это хорошо или плохо?*

• *Дайте пример того, как использование HttpHandler может упростить существующий проект, который обслуживает Check Images на .aspx-странице.*

• *На события какого вида может подписываться HttpModule? Какое влияние они могут оказать на реализацию? Что может быть сделано без перекомпиляции ASP.NET-приложения?*

• *Опишите способы представления «arbitrary endpoint (URL)» и направьте запросы к этой endpoint в ASP.NET.*

• *Объясните как работают cookies. Дайте пример злоупотребления Cookie.*

•*Объясните важность*
HttpRequest.ValidateInput()?

• *Какого рода данные передаются в заголовках HTTP (HTTP Headers)?*

• *Сравните HTTP-запросы вида GET и POST. Что такое «HEAD»?*

• *Назовите и опишите по крайней мере 6 статус-кодов HTTP (HTTP Status Codes) и объясните о чем они говорят клиенту, давшему запрос.*

• *Как работает «if-not-modified-since»? Как это*

может быть программно реализовано на ASP.NET?

• *Объясните <@OutputCache%> и использование «VaryByParam», «VaryByHeader».*

• *Как работает «VaryByCustom»?*

• *Как можно реализовать кэширование готового HTML в ASP.NET, кэшируя отправляемые версии страниц, полученные по всем значениям q= кроме q=5 (например, http://localhost/page.aspx?q=5)?*

Разработчик, использующий XML

• *В чем назначение XML Namespaces?*

• *Когда уместно использование DOM? Когда неуместно? Есть ли ограничения по размеру?*

• *Что такое «WS-I Basic Profile» и почему он важен?*

• *Напишите простой XML-документ, использующий пространство имен (namespace) по умолчанию, а также qualified (prefixed) namespace. Добавьте элементы из обоих пространств имен.*

• *В чем основное фундаментальное различие между элементами (Elements) и атрибутами (Attributes)?*

• *В чем различие между «Well-Formed XML» и «Valid XML»?*

• *Как бы вы валидировали XML используя .NET?*

• *Почему такое использование — почти всегда неудачно. В каких случаях такое уместно? myXmlDocument.SelectNodes("//mynode");*

• *Объясните различие между «pull-style parsers» (XmlReader) и «eventing-readers» (Sax)*

• *В чем различие между XPathDocument и Xml-Document? Опишите ситуацию когда один из них может быть использован над другим.*

• *В чем различие между XML "Fragment" и XML "Document"*

• *Что означает — «каноническая» форма XML?*

• *Почему спецификация XML InfoSet отличается от Xml DOM? Что пытается решить InfoSet?*

• *Сравните DTD и XSD. В чем они схожи, в чем различны? Что предпочтительнее и почему?*

• *Поддерживаются ли DTD в System.Xml? Как именно?*

• *Всякая ли XML Schema может быть представлена в виде графа объектов? А наоборот?*

Что новенького...

Передовая технология построения GUI для .NET

9Rays.Net объявила о выходе версии Beta1 нового революционного продукта в области инструментов и графических компонентов для .NET Instrumentation Model Kit. Этот продукт специально разработан для существенного упрощения и оптимизации процесса создания пользовательского интерфейса.
У вас есть прекрасная возможность сэкономить деньги – компания предлагает 30% скидку на версию Beta1. Этот продукт специально разработан для упрощения и оптимизации разработки пользовательского интерфейса.

Подробнее здесь ***http://www.9rays.net/***

Вышел VG.net 2.4: анимированная векторная графика для .NET

VG.NET – ядро векторной графики для .NET, включающее графический дизайнер, интегрированный в Visual Studio. С версии 2.4 можно создавать текстовые объекты, позиционированные относительно точки, обводить текст, интегрировать свой код отрисовки в класс CustomElement, работать внутри групп в новом режиме Active Group.

http://weblogs.asp.net/frank_hileman/archive/2005/04/26/404530.aspx

© чистоте наших рядов или миф о чистоте .NET

Автор статьи: С.Хансельман

Комментарии (курсивом) В.Чужи.

На самом деле пропаганда платформы .Net, которая, безусловно, является одной из красивейших и удобнейших платформ с точки зрения разработчика, достигла такого этапа, когда действительно, без разбора нужно это или не нужно, стараются переписать всё «под .Net». При этом часто не учитывается тот момент, что и старое приложение работает вполне неплохо. Приложение, написанное на старой, испытанной технологии, кажется «устаревшим». И это звучит как приговор.

Весна 2004. Microsoft .NET Framework включен в поставку Microsoft® Windows Server™ 2003, также мы знаем, что многие сервисы следующей версии Microsoft® Windows® "Longhorn" будут построены на управляемом APIs. Многие спрашивают "Является ли Longhorn управляемым?", намекая на то, что было бы здорово, если бы большие части операционной системы были написаны на управляемом коде. Когда же они слышат о том, что не весь код управляем – они выглядят несколько разочарованными.

Свою роль здесь, наверное, играет тяга человека ко всему новому, тем более что Longhorn изначально рекламировался как операционная система просто невероятно изобилующая всякими новинками. Многие из них «потерялись» по ходу событий, но когда оказывается, что не вся она будет написана на новой платформе – люди, конечно, выглядят разочарованными. Впрочем, не надо забывать, что реклама есть реклама. А Microsoft есть Microsoft :)

Don Box говорит прямо: "Не важно. Что важно, так это то, что… основной режим доступа… управляем". Должен ли я задумываться о том, что драйвер моего устройства написан на управляемом коде или на C? Нет, мне важно, чтобы он работал и работал хорошо.

С одной стороны, всё это так. Но мы ведь ещё не можем сказать твёрдо, каковы будут требования Microsoft к написанию драйверов устройств. Вдруг они всё-таки будут отдавать предпочтение именно управляемым драйверам? Хоть это и маловероятно, скажем честно.

В кругах разработчиков вокруг темы «чистоты .NET» разгорается всё больше дискуссий. Когда вы продаёте приложение, часто встаёт вопрос: "А является ли ваше приложение на 100% .NET"? Или - "Насколько ваше приложение является .NET приложением"? За этими вопросами кроется заранее сформированное суждение.

Я часто слышал от наших руководителей на многих технических брифингах высказывания типа: "Мы планируем портировать всю нашу систему на .NET." Зачем тратить 18 месяцев на конвертацию вашего приложения для того, чтобы очутиться в той же точке, в которой вы находитесь сейчас?

Тут можно согласиться, а можно и не согласиться. С одной стороны – да, зачем? А с другой ясно, что, во-первых, такое приложение будет лучше работать на следующих версиях Microsoft Windows (я думаю, что даже текущий .Net Framework 1.1 будет поддерживаться ещё долго), а во-вторых, персонал уже сейчас получит необходимый опыт разработки на этой платформе. Не говоря уже о том, что будет использован современный инструментарий и получен ценный опыт работы с ним. Конечно, если приложение «мертво», оно работает и не планируется его дальше развивать – смысла в этой переписке нет никакого. Кроме того, стоит заметить, что переписанное приложение уже не будет таким, как старое – используя новые технологии вы наткнётесь на какие-то новые подходы и решения, которые, наверняка, помогут изменить ваше приложение к лучшему и в большей степени соответствовать требованиям ваших клиентов.

Подразумевается, что приложение, полностью написанное на .NET, преимущественно без взаимодействия с COM или прямых вызовов Win32 API, превосходит приложение, являющееся комбинацией таких технологий. Трагичность иронии стремления "как можно меньше использовать интероперабельность" заключается в том, что сам .NET Framework, на основе которого всё построено, является фантастическим примером интероперабельности.

Как-то, начитавшись всякой рекламы о «чистоте» .Net я об этом также задумался. Первой мыслью была такая – неужели они ВСЁ переписали на ассемблере. Второй – а смысл? Если есть функции WinAPI и готовые COM-объекты, которые можно использовать. Ведь .Net не позиционировалась, в отличие Java, как многоплатформенный продукт. Она позиционировалась как многоязычная платформа. Хотя, если посмотреть, что каркас .Net работает на Win98, WinMe, Win2k, Win2003, то может быть она и многоплатформенная? :)

.NET предоставила разработчику возможность сделать фантастический прыжок в продуктивности работы, а также понятный, непротиворечивый и последовательный интерфейс к сервисам, предоставляемым платформой Windows. Многие годы набор интерфейсов, предлагаемый платформой Windows —известный всем как Windows SDK— был знаком разработчикам как экспортируемые функции в стиле "C", размещённые в DLL, а в последние годы в виде модели Component Object Model (COM). Всё усложнялось, появлялись новые абстракции и новые режимы доступа.

И это тоже доводы в пользу переписки/перевода развивающихся продуктов на платформу .Net – если говорить терминами Толкиена, это то кольцо, которое объединит всех.

Классы MFC были обёрткой над Win32, но были реализованы несколько другим образом, чем в Microsoft® Visual Basic® 6.0 или Windows Template Library (WTL). Классический Microsoft® ASP вообще не имел событийной модели, а был надстройкой над HTTP, в то время как Visual Basic привлекал народ возможностью, дважды щёлкнув, получить автоматически привязанный (например, к нажатию контрола) обработчик события. В случае с .NET Framework многие подходы были унифицированы в одной программной модели. Одно бесспорно – управляемые API .NET Framework «сидят» на Windows, собственно "Платформе".

Честно говоря, не думаю, что это какое-то «великое» открытие автора. Всё это было достаточно очевидно с самого начала, стоило чуть пристальней взглянуть на реализацию, отбросив рекламную мишуру.

Библиотека .NET Framework

Платформа Windows имеет десятки высокоуровневых сервисов, являющихся результатом существования буквально тысяч функций API. Эта огромная, невероятно функциональная библиотека, обладает разными уровнями мощи. Низкоуровневое API может открыть файл на диске, в то время как высокоуровневое может проиграть аудио файл. Дизайнеры .NET Framework хотели создать понятный, объектно-ориентированный интерфейс, использующий богатое наследство функциональности платформы. CLR и .NET Framework работают вместе над раскрытием возможностей платформы Windows, включая те из них, которые были малоизвестны благодаря трудностям работы с API.

В то время как CLR предлагает новую парадигму разработки приложений, она не закрывает дверь доступа к существующим библиотекам. CLR предлагает разработчику сервисы интероперабельности, но самым большим потребителем таких сервисов является сама библиотека классов .NET, которая предоставляет нам функциональность платформы Windows посредством .NET API.

Например, отсылая e-mail, вы используете класс System.Web.Mail.SmtpMail, а библиотека классов использует вспомогательный класс, задействующий CDO (Collaboration Data Objects) из библиотеки COM. Это только один пример, один из тысячи, в которых разработчик библиотеки .NET решил использовать уже испытанный способ, а не писать нечто совершенно новое. Этот и другие примеры из библиотеки классов не являются чем-то необычным, ведь и Common Language Runtime также иногда использует внутренние API платформы Windows. Как

говорится, не важно с чем вы работаете, всё равно вам придётся вызывать функцию LoadLibrary().
Если бы Microsoft действительно полностью виртуализировала машину, то они бы поставили под сомнение свои прежние инвестиции в платформу Windows.

А деньги считать там умеют. К тому же, будем смотреть правде в глаза, одоной из целей создания платформы .Net была ещё большая привязка разработчиков к платформе Windows – радует то, что на этот раз нас «привязывают» удобством разработки, совершенством инструментов и красотой самой платформы.

Обычно дизайнеры стараются делать обращения к существующим библиотекам как можно более безболезненными. В данном случае они сделали это через интероперабельность NET/COM как через обёртки времени выполнения, так и COM-обёртки, добавили возможность вызова стандартных API платформы Win32 посредством технологии, названной P/Invoke (сокращение для platform invoke). При написании кода, хостящегося в CLR, огромное количество ресурсов платформы находятся под разработчиком — саму среду выполнения скорее можно назвать прозрачной, чем виртуальной. Это подчёркивает фундаментальное различие в видении роли платформы относительно других реализаций виртуальных машин.
GDI, он же Graphics Device Interface или Интерфейс Графических Устройств, яркий пример неуправляемого сервиса нижнего уровня, который был практически бесшовно соединён с управляемым миром посредством P/Invoke. Полезно было бы посмотреть на текущие "три столпа Windows" для того, чтобы напомнить пользователям, где выполняется большая часть неуправляемой работы, не взирая на то, насколько ты чист.

DLL	Описание содержимого
Kernel32.dll	Содержит низкоуровневые функции операционной системы для неуправляемого управления памятью и ресурсами.
GDI32.dll	Функции Graphics Device Interface (GDI) для рисования, управления шрифтами, общим выводом устройств.
User32.dll	Функции управления Windows – обработка сообщений, меню, коммуникациями.

Если бы мне нужно было использовать внутреннюю неуправляемую функцию FindWindow из User32.dll, я бы добавил такое объявление в моё приложение на C#:

```
[DllImport("User32.dll")]
public static extern int
FindWindow(string strClassName,
string strWindowName);
```

После такого объявления (которое заметно отличается от типичной функции .NET только атрибутом DllImport и возможно static extern), я могу использовать этот метод, как и любой другой. Конечно, при маршаллинге между управляемым и неуправляемым кодом приходится применять некоторые хитрости, но в большинстве случаев всё это замечательно работает. Доказательством этого служит ваш собственный код, постоянно обращающийся к неуправляемым DLL, о которых вы вообще вряд ли вспоминаете.

Как, собственно говоря, и должно быть.

При создании приложения, использующего только .NET, можно получить бонусы в области распространения или маркетинга архитектуры ("маркетектуры"), эти преимущества сводятся на нет при переписывании не-.NET компонент на .NET. «Чистое» .NET решение может использовать только ту функциональность, которая доступна целиком во время выполнения из библиотеки времени выполнения, или те функции, которые есть в Base Class Library—которая сама использует COM Interop и P/Invoke.
Сама по себе библиотека .NET Framework не является "чистой .NET", поскольку использует каждую возможность для задействования примитивов платформы.

Что наверняка даёт выигрыш в

производительности. Примитивы платформы (функции API, функциональность, заложенная в COM-объектах) тем и хороши, что они (в идеале) уже сто раз проверены и отточены – как с точки зрения быстродействия, так и с точки зрения безопасности.

Концепция чистоты в .NET теперь представляется в несколько другом свете. .NET Framework – конечно лучший способ для создания бизнес-компонент на платформе Windows, но любое приложение, написанное на .NET, не сможет прыгнуть выше, чем используемые им сервисы операционной системы.

Тоже спорное утверждение. Целое может быть качественно функциональнее, чем его составные части. Иначе, фактически, ставится под сомнение вообще создание чего-либо нового – в принципе.

Зачем вообще писать управляемый код?

Зачем делать шаг вверх по лестнице языков и среды разработки и выполнения? Для того чтобы жить на другом уровне абстракций. Конечно, .NET CLR даст вам лучший объектно-ориентированный опыт по сравнению с Visual Basic 6.0 и даже, возможно, C++, но реальная ценность каркаса в его способности эффективно прятать системные API используемой операционной системы, а затем отображать их на новом уровне для множества языков. Конечно, вы не станете писать драйвер устройства реального времени на управляемом коде (пока), но я уверен, что к середине этой декады почти весь бизнес-ориентированный код, достойный хоть какого-нибудь внимания, будет написан на управляемой среде. (Скорее всего это уже так.)

Подытожить этот абзац можно ещё и так – трудно спорить с тем, кто оплачивает музыку.

CLR нацелена на освобождение разработчика от заботы о выделении и освобождении памяти, низкоуровневых операций ввода-вывода и даже работы с протоколами, во времена C++, мы много и хорошо говорили об объектах "более близких к людям бизнеса" и "лёгким в работе", но наши объектно-ориентированные иллюзии рассеялись, как только понадобилось сериализовать объекты, легко передавать их данные или сохранять их в БД. Сервисно-ориентированная архитектура (Service-oriented architecture, SOA) начала примирять обязанности логического сервиса и объекта (сообщения), с которым он имеет дело, но этот шаг вперёд был бы не возможен без расширяемых, полноценных и вездесущих метаданных и унифицированной системы типов. Мы слишком погрязли в деталях. Вот почему необходимость написания управляемого кода очевидна.

Common Language Runtime или Виртуальная машина?

Часто .NET Common Language Runtime, или CLR, прямо сравнивают с Java™ Virtual Machine. Изначально много явных параллелей: обе являются "управляемыми" средами, являющимися контейнерами компонент, обе используют "частично пережёванный" промежуточный язык, обе предлагают низкоуровневые сервисы – такие, как уборка мусора и удобства работы с нитями.

В то время как семантически эти параллели корректны на внешнем уровне, две эти реализации различаются фундаментально – в самой философии их построения. Сравнивать CLR и Виртуальную Машину можно до определённого момента – архитектурные цели совершенно различны.

Sun Microsystems™ проводила маркетинговую программу 100% Pure Java, которая уместна в случае желания достичь переносимости кода и наличия прозрачности операционной системы. Однако многие третьи поставщики серверов приложений Java создали конкуренцию, используя благоразумные прямые вызовы функций на "C" (через Java Native Interface или JNI), использующие ценные сервисы операционных систем, которые не были доступны на Java Application Platform (в библиотеке классов Java). Обращение к функциям ядра платформы – единственный способ использования базовой функциональности, доступной через родной интерфейс.

Java Virtual Machine – это настоящая "виртуальная машина", конечная цель которой – абстрагировать (виртуализировать) используемую операционную систему и предложить идеализированную (не обязательно идеальную, но идеализированную) среду разработки. Java Virtual Machine также тесно связана с API платформы приложений

Java (Java Application Platform), с сервисами, поддерживаемыми реализацией виртуальной машины. Вне зависимости от того, где вы будете запускать скомпилированный код Java, вы будете его запускать внутри контекста Виртуальной Машины, как бы имея связь с API, поддерживаемым платформой Java.

Тут можно заметить, что, раз возникает необходимость обращения непосредственно к функциям ОС через JNI, значит даже не идеальная, а идеализированная среда не всех устраивает.

.NET Common Language Runtime хорошо назван – поскольку он более является библиотекой времени выполнения для различных языков, чем виртуальной машиной. И хотя он успешно абстрагируется от аспектов используемого железа, посредством использования IL (intermediate language), как только CLR комбинируется с библиотекой API каркаса .NET, то сразу становится привязанной к одной платформе - Windows. CLR предлагает все удобства платформы Windows любому языку .NET.

Также стоит заметить, что весь код, исполняющийся в управляемой среде .NET, фактически работает откомпилированным в родные инструкции, что позволяет сбалансировать гибкость и производительность, абстрагированные от железа, с близостью к этому самому железу.

Вам наверное известно, что IL-код перед выполнением транслируется в так называемый «родной» код с помощью утилиты ngen. Код этот, к сожалению, действителен только для того компьютера, на котором он скомпилирован. Думаю, что сделано это искусственно – просто для того, чтобы .Net программы не распространяли без самой платформы .Net, хотя встречалась также информация об оптимизации под конкретный установленный процессор, что сомнений не вызывает, но ситуацию не оправдывает.

“Гибридные” решения и есть реальные решения

Многие большие существующие приложения написаны на Microsoft® Visual C++® и COM. Они написаны “близко к железу” – для использования преимущества родной многопоточности Windows и тонкого (неавтоматического) управления памятью. Однако новые бизнес-компоненты могут также быть написаны на языках .NET, таких как C# или Visual Basic .NET. Существующая система хостит .NET Common Language Runtime внутри себя. Интерфейс, использующий взаимодействие с COM, вовлекает в работу небольшое, от 10 до 40, инструкций процессора на одно обращение (число инструкций может варьироваться).

Компоненты .NET, размещённые в приложении, могут получить преимущества от использования сервисов этого приложения. Низкоуровневые возможности, такие как управление памятью, время жизни объекта и т.п. поддерживаются CLR, в то время как высокоуровневая, вертикально-специфичная бизнес-функциональность доступна через приложение.

Более полезно было бы представлять неуправляемые сервисы приложения как управляемые и, возможно, сервисно- или объектно-ориентированные API. А то, что внутренняя реализация сервиса неуправляема – всего лишь деталь этой самой реализации.
Эта “гибридная” модель могла бы предложить и предлагает лучшие решения на платформе Windows, эксплуатируя высокопроизводительные низкоуровневые API посредством C++, и высок окомпонентизированные особенности каркаса .NET Framework. Эти решения могут быть очень успешными, пока компании переходят полностью на платформу .NET.

Какие ещё выводы можно сделать по прочтению этой статьи? Первый, просто напрашивающийся вывод прост и всеобъемлющ – посмотрел рекламу? Потрогай руками! Что же касается «чистоты» .Net – конечно, «гибридные» модели на данный момент работают лучше. Но есть одно но – никто не обещает вам, что функции платформы Windows, которые вы используете напрямую в своей программе, будут поддерживаться и далее. Значит, их использование должно быть оправданным. Кроме того «морально» нужно быть готовым к изменению сигнатур таких функций или переписывания текущей функциональности без их участия. А именно «чистый» .Net код избавляет от такого рода проблем.

Журнал	“Алгоритм”
Учредитель	В.Ф.Чужа
Издатель	ЧП Чужа Т.А.
Веб-сайт	http://dotnetgrains.sql.ru/
Эл.почта	hdrummer@sql.ru
Телефон	+380 5366 23109
Адрес редакции	Украина, г.Кременчуг, ул.Бутырина, 71/14
Главный редактор	В.Ф. Чужа
Корректор	А.Дорофеева
Дизайн	О. Ситников
Подписано в печать	23.06.2005 г.
Тираж	500 экз.

Журнал зарегистрирован Государственным Комитетом информационной политики, телевидения и радиовещания Украины. Свидетельство о регистрации КВ9094 от 26.08.2004 г.