Федеральное агентство по образованию
Нижегородский государственный университет им. Н.И. Лобачевского

Национальный проект «Образование»
Инновационная образовательная программа ННГУ. Образовательно-научный центр
«Информационно-телекоммуникационные системы: физические основы и математическое обеспечение»

О.Г.Савихин

Основы разработки приложений в Microsoft Visual Studio .NET

Учебно-методические материалы по программе повышения квалификации «Информационные технологии и компьютерное моделирование в математике и механике»

Нижний Новгород
2007

Учебно-методические материалы подготовлены в рамках инновационной образовательной программы ННГУ: *Образовательно-*

научный центр «Информационно-телекоммуникационные системы: физические основы и математическое обеспечение»

Савихин О.Г Основы разработки приложений в Microsoft Visual Studio .NET

Учебно-методический материал по программе повышения квалификации «Информационные технологии и компьютерное моделирование в математике и механике». Нижний Новгород, 2007, 93с.

Учебно-методический материал содержит теоретические основы разработки Windows приложений, включая введение в событийно-ориентированное программирование и введение в объектно-ориентированное программирование на C++. В качестве среды разработки используется интегрированная среда Microsoft Visual Studio .NET Обсуждаются особенности платформы Microsoft .NET, основные элементы визуальной среды Visual Studio и ее возможности.

ОГЛАВЛЕНИЕ

СОДЕРЖАНИЕ

2. Введение в объектно-ориентированное программирование на C++
    1. Парадигма программирования
    2. Абстрактный тип данных (АТД)
    3. Классы
        1. Инкапсуляция. Функции-элементы
        2. Сокрытие данных. Открытые, закрытые, защищенные члены класса.
        3. Управление созданием, инициализацией и уничтожением объектов классов. Специальные функции-члены классов: конструктор и деструктор
    4. Наследование
        1. Производные и базовые классы
3. Платформа Microsoft .NET
    1. Общая среда исполнения (Common Language Runtime)
    2. Стандартная система типов (Common Type System)
    3. Структурные и ссылочные типы
        1. Самоописывающие ссылочные типы
            1. Классы
            2. Типы-интерфейсы
        2. Дополнительные элементы системы типов .NET
            1. Структуры и перечисления
            2. Указатели
        3. Упакованные типы-значения
    4. Встроенные типы данных CTS
    5. Преобразование типов с помощью класса System::Convert
    6. Библиотека базовых классов
        1. Класс System::Object
4. Расширения управляемого C++ (Managed C++)
    1. Управляемые типы данных
    2. Класс управляемого C++

3. Свойства в .NET
4. Делегаты
5. Реализация обработки событий в .NET
6. Управляемые массивы.
7. Класс System::Array
8. Управляемые двумерные массивы

5. Работа со строками в Windows
    1. Использование кодировки UNICODE
        1. Представление символов в UNICODE
        2. Разработка универсальных программ для работы со строками
        3. Перекодирование однобайтовых символов в Unicode и обратно с учетом кодовой страницы
        4. Пример перекодирования символов Unicode в однобайтовые
    2. Класс System::String
        1. Инициализация строк при помощи строковых констант
        2. Массивы строк
        3. Сравнения объектов класса String
        4. Методы класса String
    3. Строковое представление данных
    4. Форматирование строк
        1. Строки стандартных числовых форматов
        2. Составное форматирование
        3. Синтаксис элементов форматирования
        4. Разбор строк

6. Принципы разработки Windows приложений в .NET (Windows Forms Application)
    1. Визуальная разработка приложений.
        1. Формы
        2. Компонентная модель.

3. Конструирование приложения.
4. Автоматическая генерация кода.
5. Механизм двунаправленной разработки.

2. Создание проекта при помощи шаблона Windows Forms Application
3. Генерация кода
4. Режимы дизайна и кода

7. Окна инструментов среды разработки Visual Studio
    1. Окно Toolbox
    2. Форматирование элементов управления
    3. Окно свойств (Properties window)
        1. Интерфейс окна Properties
        2. Редакторы свойств
        3. Обработка событий
    4. Окно проводника классов (Class View)
    5. Окно Object Browser
    6. Окно Server Explorer
    7. Окно динамическая справка (Dynamic Help)
    8. Перемещение и изменение размеров окон инструментов
        1. Закрепление инструмента в Visual Studio
        2. Скрытие инструмента в Visual Studio

ГЛАВА 1. МНОГОФАЙЛОВАЯ ОРГАНИЗАЦИЯ ПРОГРАММЫ (МОДУЛЬ, ПРОЕКТ, РЕШЕНИЕ, ПРОСТРАНСТВА ИМЕН, СБОРКА)

Все нетривиальные программы сбираются из нескольких раздельно компилирующих единиц. Единицей компиляции является файл. Определения и декларации глобальных объектов в различных файлах программы должны быть согласованны.

Отличительной особенностью Си является отсутствие в языке современных средств раздельной зависимой компиляции.
Раздельно зависимая компиляция облегчает возможность использования объекта, определённого в другом модуле. Для этого достаточно указать имя модуля. Извлечение описания объекта и проверка правильности его использования обеспечивается компилятором. В Си независимо раздельная компиляция наделяется свойствами зависимой при помощи препроцессора, который является неотъемлемой частью языка. Согласованное использование глобальных имён и типов достигается за счёт наличия только единственной копии их описания. Описание глобальных объектов выделяется в отдельные файлы, которые называются заголовочными. Заголовочные файлы с помощью директива препроцессора include помещаются, как в файлы, где они определяются, так и в файлы, где они используются. В результате компилятор получает возможность находить несоответствие в описаниях одного и того же объекта. Для небольших программ удобно создавать один заголовочный файл. Он должен содержать описание всех объектов, используемых по крайней мере в двух файлах.

Один из способов повышения надёжности программ заключается в разбиении её на части, которые содержат только информацию необходимую для их работы. Набор взаимосвязанных процедур и тех данных, с которыми они оперируют, называются **модулем**, а подход построения программ - модульным стилем программирования. В языке Си модуль состоит из двух файлов: заголовочного (с расширением .h) и исполняемого (.cpp). Заголовочный файл представляет интерфейс модуля, а исполняемый файл задаёт реализацию функций, содержащихся в интерфейсе. Код пользователя, использующего только интерфейс модуля, не зависит от деталей его реализации.

Программа может состоять из нескольких модулей и файлов различных типов. Совокупность всех файлов и модулей программы образует **проект (project)**. Приложение в Visual Studio .NET может состоять из нескольких проектов, совокупность которых называется термином **решение (Solution).** В результате компиляции решения создается исполняемый файл в формате PE (PE-файл), который называется **сборкой (assembly)** . Программист работает с решением, CLR - со сборкой.

Решение содержит один или несколько проектов, ресурсы, необходимые этим проектам, возможно, дополнительные файлы, не входящие в проекты. Один из проектов решения должен быть выделен и назначен стартовым проектом. Выполнение решения начинается со стартового проекта. Проекты одного решения могут быть зависимыми или независимыми. В уже имеющееся решение можно добавлять как новые, так и существующие проекты. Один и тот же проект может входить в несколько решений. Проект - это основная единица, с которой работает программист. Он выбирает тип проекта, а Visual Studio создает скелет проекта в соответствии с выбранным типом.

Проект состоит из классов, собранных в одном или нескольких **пространствах имен (namespace)** . Пространства имен позволяют структурировать проекты, содержащие большое число классов, объединяя в одну группу близкие классы. Если над проектом работает несколько исполнителей, то, как правило, каждый из них создает свое пространство имен. Помимо структуризации, это дает возможность присваивать классам имена, не задумываясь об их уникальности. В разных пространствах имен могут существовать одноименные классы. Таким образом, пространство имен — это логическая структура для организации имен, используемых в приложении .NET. Основное назначение пространств имен — предупредить возможные конфликты между именами. Класс проекта погружен в пространство имен, имеющее по умолчанию то же имя, что и решение, и проект. Итак, при создании нового проекта автоматически создается достаточно сложная вложенная структура - решение, содержащее проект, содержащий пространство имен, содержащее класс, содержащий точку входа. Для простых решений такая структурированность представляется избыточной, но для сложных - она осмысленна и полезна.

Основным понятием при программировании в среде .NET является понятие сборки. Согласно терминологии Microsoft код, предназначенный для работы в среде выполнения .NET, — это *управляемый код* (managed code). Двоичный файл, который содержит управляемый код, называется ***сборкой*** **(assembly)**. Приложения .NET создаются путем объединения любого количества сборок. Сборка — это двоичный файл (DLL или EXE), который содержит в себе номер версии, метаданные, а также типы (классы, интерфейсы, структуры и т. п.) и дополнительные ресурсы (изображения, таблицы строковых данных и т.д.).

Сборки .NET содержат не платформенно-зависимые инструкции, а код на так называемом **промежуточном языке Microsoft (Microsoft Intermediate Language, MSIL или просто IL)**. Этот язык не зависит ни от платформы, ни от типа центрального процессора. Код IL компилируется в платформенно-зависимые инструкции только во время выполнения.

Помимо собственно инструкций на языке IL, каждая сборка .NET содержит в себе информацию о каждом типе сборки и каждом члене каждого типа. Эта информация генерируется полностью автоматически. Любая сборка .NET содержит **манифест — набор метаданных о самой сборке**. Манифест содержит информацию обо всех двоичных файлах, которые входят в состав данной сборки, номере версии сборки, а также, что очень важно, — сведения обо всех внешних сборках, на которые ссылается данная сборка.

Причиной появления понятия сборки можно считать трудности установки Windows-приложений. Обычное Windows-приложение состоит из множества файлов - запускаемые модули, библиотеки, дополнительные файлы и т.п. Помимо этого, при установке некоторых приложений (особенно COM-компонент) необходимо записывать в реестр Windows сведения о нахождении и способе вызова. Наконец, многие приложения использовали разделяемые DLL, что зачастую приводило к проблемам при установке более новых версий этой DLL.

Понятие сборки было введено для того, чтобы решить эти проблемы. Сборка представляет собой набор файлов, модулей и дополнительной информации, которые должны обеспечить простую установку приложения и последующую работу. Таким образом, можно говорить и о том, что повторное использование приложений может быть реализовано с помощью интеграции различных сборок.

Пользователю сборки гораздо важнее ее логическое представление, в котором сборка — это набор открытых типов, используемых в приложении («внутренние» типы — это, как правило, служебные типы, используемые другими типами той же самой сборки). На физическом уровне сборка — это единственный исполняемый файл , а в тоже время на логическом уровне — это иерархия взаимосвязанных типов

ГЛАВА 2. ИНТЕГРИРОВАННАЯ СРЕДА РАЗРАБОТКИ MICROSOFT VISUAL STUDIO .NET

1. Создание нового проекта
2. Виды проектов
3. Основные части визуальной среды разработки Visual Studio .NET
4. Окно проводника решения (Solution Explorer)
5. Файлы проекта
6. Свойства проекта
7. Конфигурация проектов
8. Редактор кода

Microsoft Visual Studio .NET - это интегрированная среда разработки (IDE) для создания, документирования, запуска и отладки программ, написанных на языках .NET.

Интегрированная среда разработки IDE (Integrated Development Envirionment) Visual Studio является многооконной, настраиваемой, обладает большим набором возможностей.

Главное окно Visual Studio.NET, подобно другим приложениям Windows, содержит строку меню,

включающую в себя следующие категории

- File — открытие, создание, добавление, закрывание, печать и проч.
- Edit — стандартные команды правки: копирование, вставка, вырезание и проч.
- View — команды для скрытия и отображения всех окон и панелей инструментов.
- Project — команды для работы с проектом: добавление элементов, форм, ссылок и проч.
- Build — команды компиляции программы.
- Debug — команды для отладки программы.
- Data — команды для работы с данными.
- Format — команды форматирования располагаемых элементов (выравнивание, интервал и проч.).
- Tools — команды дополнительных инструментов и настройки Visual Studio

.NET.

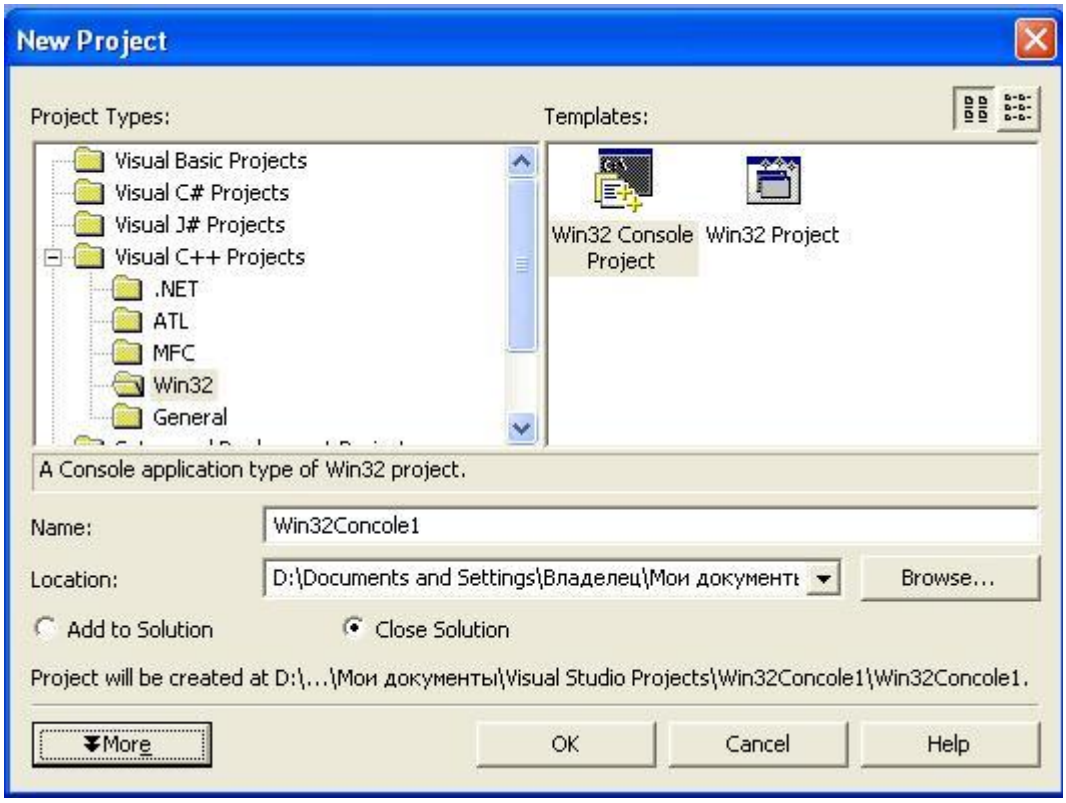
- Window — управление расположением окон.
- Help — справка.

Под строкой меню расположена *панель инструментов*, содержащая вложенные панели кнопок, запускающих те или иные команды из определенной группы или управляющих средой разработки Visual Studio.

Поместить группу кнопок на панель инструментов можно при помощи пункта меню View / Toolbars

Создание нового проекта

Если в меню среды разработки выбрать пункт File | New | Project, на экране появится диалоговая панель New Project .



Диалоговая панель New Project

При создании нового проекта в поле Location необходимо указать имя каталога, в котором следует сохранить его файлы. При этом в данном каталоге автоматически будет создан другой каталог, имя которого совпадает с именем проекта. По умолчанию проекты сохраняются в файле C:\Documents and Settings\Владелец\Мои документы\Visual Studio Projects\Имя проекта.

Виды проектов

Visual Studio .Net для языков C#, Visual Basic и J# предлагает 12 возможных видов

проектов. Среди них есть пустой проект, в котором изначально не содержится никакой функциональности; есть также проект, ориентированный на создание Web-служб.

В левой части этой диалоговой панели можно выбрать тип проекта. В общем случае можно выбрать проекты, созданные на языках программирования Visual Basic .NET, C#, C++, а также на ряде других. Этот список зависит от того, какие языки были выбраны при установке Visual Studio, а также от того, были ли приобретены и установлены дополнительные языки программирования сторонних производителей.
В правой части экрана можно выбрать один из предложенных шаблонов для данного типа проектов:

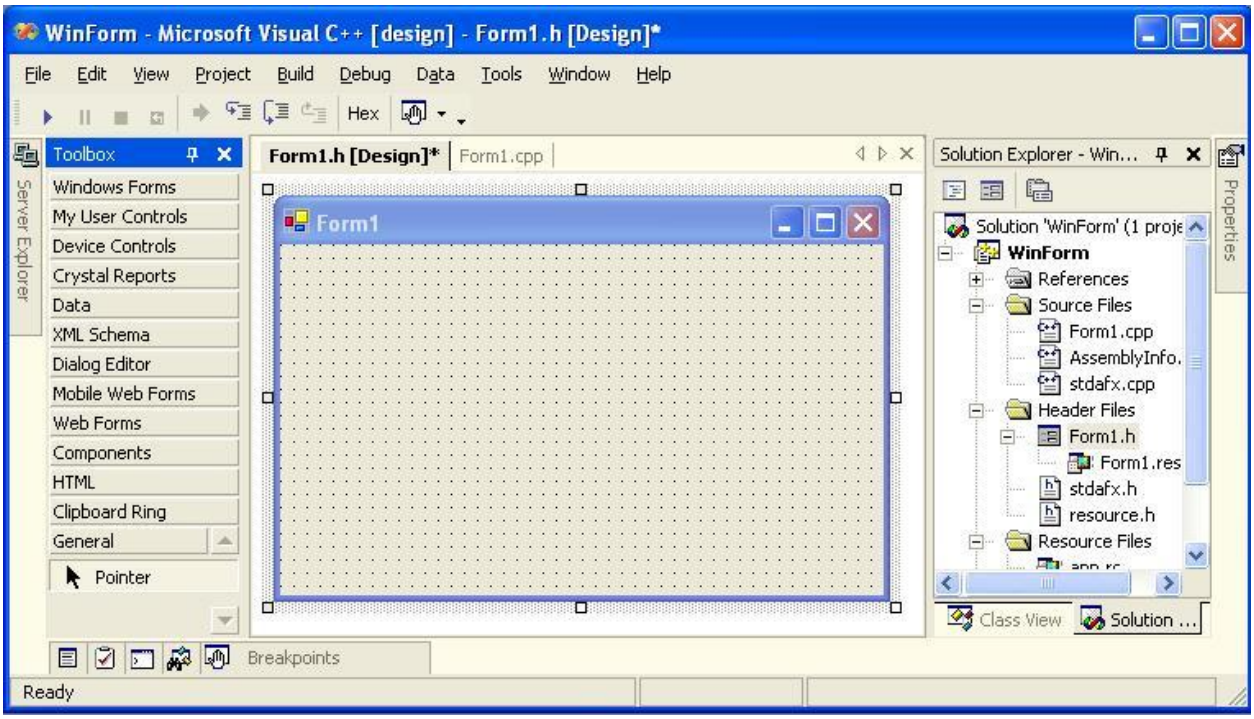
- Windows Application — шаблон для Windows-приложения;
- Class Library — шаблон для создания библиотеки классов, которая будет использоваться другими приложениями;
- Control Library — шаблон для создания элементов управления, которые будут использоваться в приложениях с графическим пользовательским интерфейсом для платформы Windows (называемых также приложениями Windows Forms);
- ASP.NET Web Application — шаблон для создания Web-приложений ASP.NET;
- ASP.NET Web Service — шаблон для создания Web-сервисов;
- Web Control Library — шаблон для создания элементов управления, которые будут использоваться в Web-приложениях;
- Console Application — шаблон для создания консольных приложений;
- Windows Service — шаблон для создания сервисов операционой системы;
- Empty Project/Empty Web Project — проект, который создается без использования шаблонов;
- New Project In Existing Folder — добавить новый проект в уже существующую папку.

Хотя при создании нового проекта в среде Visual Studio .NET предлагается довольно большой список типов проектов , но на самом деле есть всего три основные разновидности приложений - Windows Application, Console Application и Class Library . Все остальное - это их различные вариации за счет использования тех или иных шаблонов или мастеров (именно поэтому правое окно и называется Templates), обеспечивающих автоматическое выполнение каких-то начальных действий, которые при желании можно выполнить и "руками" (в том числе изменив и базовый тип приложения). Пользователь может подключить и свои собственные варианты шаблонов.

Основные части визуальной среды разработки Visual Studio

Существует три основные части визуальной среды при разработке проекта. В центре находится главное окно для создания визуальных форм и написания кода. Справа размещается окошко Solution Explorer (проводник решения), а ниже его окно инспектора

свойств Properties Explorer. Окно Solution Explorer позволяет увидеть, из каких проектов состоит решение и какие файлы входят в состав этих проектов. Окно свойств (Properties) содержит список атрибутов объекта, выделенного в данный момент. В левой части среды разработки присутствует элемент управления со значком окна Server Explorer; это окно появится, если указатель мыши окажется над данным значком. Там же имеется и значок окна Toolbox — оно появится, если поместить указатель мыши над этим значком.

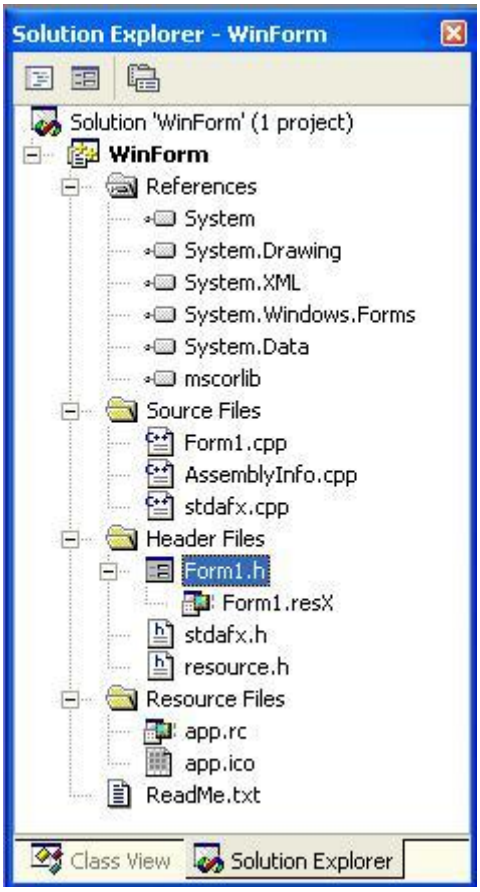


Среда разработки Visual Studio .NET содержит два типа окон — окна инструментов и окна документов. Окна инструментов (часть из которых была описана выше) доступны с помощью команд меню View и некоторых других, и их доступность зависит от типа приложения и от того, какие модули расширения (дополнительные утилиты и инструменты, в том числе произведенные сторонними разработчиками) добавлены к среде разработки. В окнах же документов можно редактировать компоненты проектов.
С окнами инструментов можно производить различные манипуляции. В частности, можно заставить их автоматически появляться и исчезать, группировать их в виде многостраничного блокнота, варьировать их расположение в среде разработки, делать их «плавающими» и даже отображать на дополнительном мониторе, если использование такового поддерживается операционной системой.
Некоторые окна инструментов, например окно Web Browser, можно создавать в виде нескольких экземпляров (это можно сделать, выбрав пункт меню Windows | New Window). Можно также заставить окна инструментов автоматически исчезать, если они в данный момент не являются активными, — в этом случае на экране отображаются название и пиктограмма окна, над которой можно поместить указатель мыши, если окно нужно отобразить целиком. Если необходимо предотвратить исчезновение окна с экрана, следует щелкнуть мышью по изображению канцелярской кнопки на заголовке окна.

Окно проводника решения (Solution Explorer)

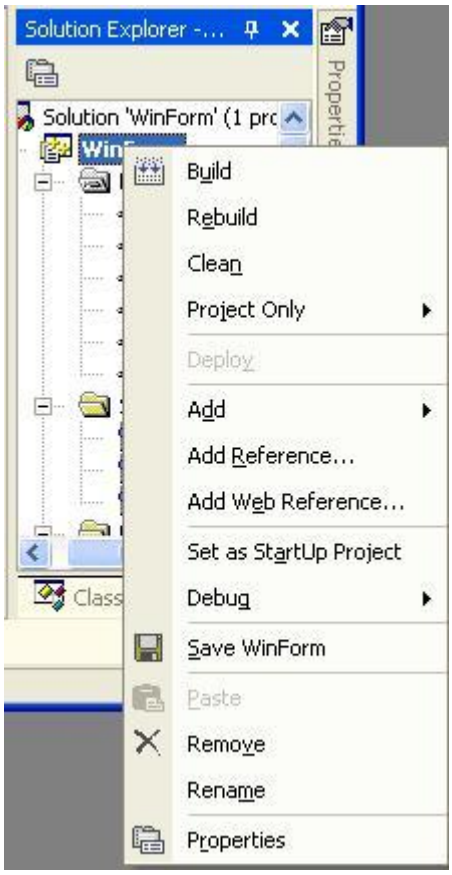
Окно Solution Explorer можно отобразить на экране с помощью команды меню View |

Solution Explorer.Окно Solution Explorer содержит древовидное представление элементов проекта, которые можно открывать по отдельности для модификации или выполне ния задач по управлению. В дереве отображаются логические отношения решения и проектов, а также элементов решения. Решение — это набор проектов, из которых состоит приложение. Компонентами проектов могут быть модули, а также другие файлы, которые требуются для создания приложения. Если нужно отредактировать компонент проекта, следует дважды щелкнуть по его имени в окне Solution Explorer.



Окно Solution Explorer

Пункты контекстное меню этого окна (вызывающегося нажатием правой кнопкой мыши) позволяют изменять содержимое проекта, а также добавлять новые компоненты. Помимо обычных программных модулей, мы можем с помощью команды File | Add Item подключать к проекту самые разные компоненты, например, HTML-страницу, которую затем можно наполнить с помощью встроенного HTML-редактора. Кроме того, в среде разработчика имеется новый дизайнер XML-документов и XSD-схем, набор графических редакторов и целый ряд других инструментов. Чтобы связать файлы с решением, но не с одним из его проектов, достаточно присоединить его прямо к решению.



С помощью кнопок, расположенных в верхней части окна Solution Explorer, можно указать, что именно должно отражаться в среде разработки:

- View Code — код, связанный с файлом, выделенным в окне Solution Explorer;
- View Designer — дизайнер (визуальный редактор) файла, выделенного в окне Solution Explorer;
- Refresh — обновить содержимое окна Solution Explorer;
- Show All Files — все файлы, включая код, связанный с формами;
- Properties — свойства выбранного файла.

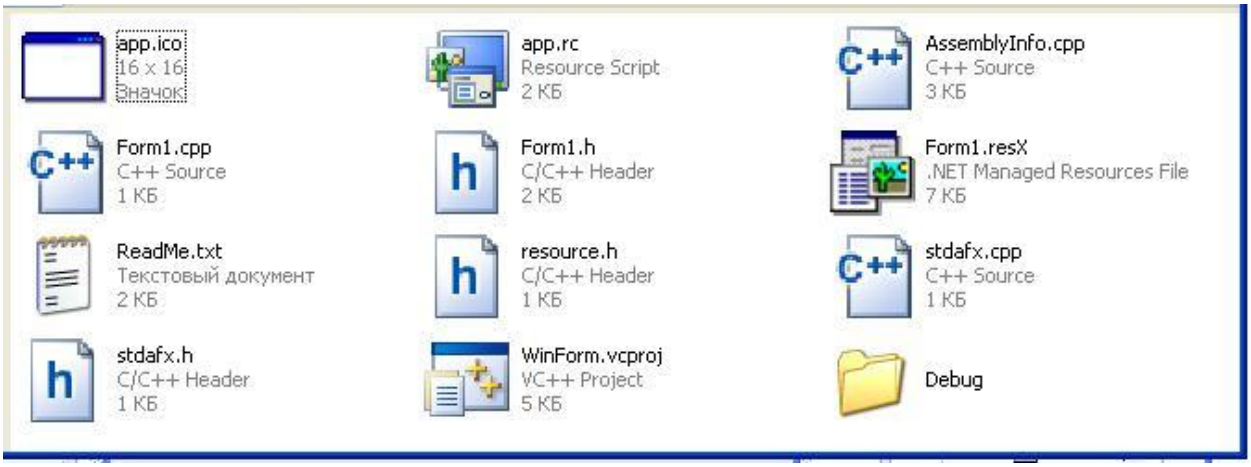
При создании нового проекта Solution Explorer содержит компоненты, созданные шаблоном

Папка References содержит ссылки на классы, используемые в проекте по умолчанию. Двойной щелчок мыши на подпапках References запускает окно Object Browser (проводник объектов, View —> Object Browser, или сочетание клавиш Ctrl+Alt+J). Окно Object Browser, в свою очередь, является исчерпывающим средством получения информации о свойствах объектов. Можно получать краткое описание любого метода, класса или свойства, просто щелкнув на нем, — на информационной панели немедленно отобразится краткая справка.

Файлы проекта

В проекте Visual C++ .NET взаимозависимости между отдельными частями описаны в

текстовом файле проекта с расширением VCPROJ. А специальный тексто вый файл решения с расширением SLN содержит список всех проектов данного решения.Чтобы начать работу с существующим проектом, достаточно открыть в Visual C++ .NET соответствующий SLN-файл.Visual C++ .NET также создает промежуточные файлы нескольких типов



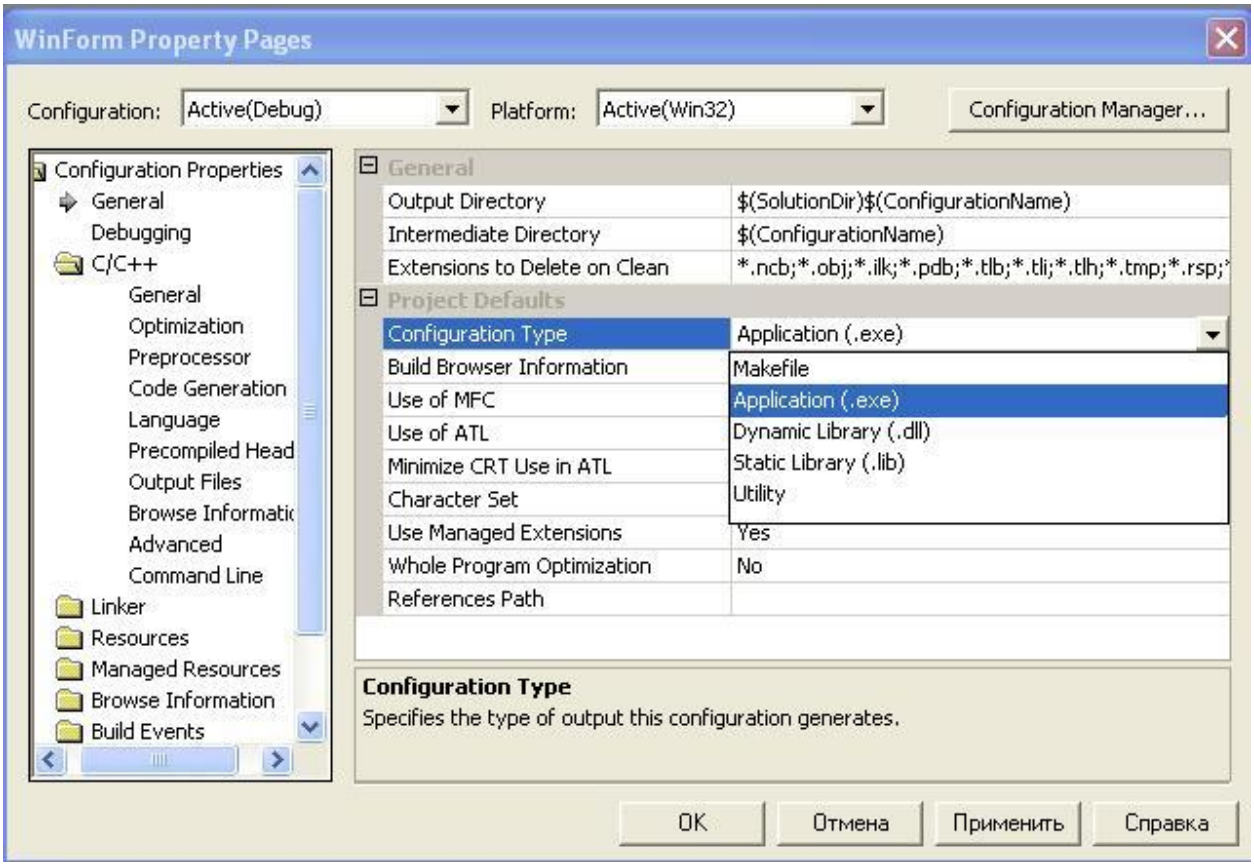
Содержимое папки Debug

RC, resX -Поддержка просмотра ресурсов
RES, RESURCES откомпилированные файлы ресурсов
NCB -Поддержка просмотра классов. Этот файл создается и затем обновляется при каждом запуске программы. Он имеет самый большой объем среди всех файлов проекта. С целью экономия места на диске ***файл с расширением NCB***, а также папку ***Debug***, которая образуется после компиляции программы, ***необходимо удалять***.
PDB файл, используемый компоновщиком для записи отладочной информации о пользовательской программе с целью ускорения редактирования связей в режиме отладки. Этот файл содержит отладочную информацию, а также информацию о состоянии проекта.
SLN Файл решения.
SUO Поддержка параметров и конфигурации решения
VCPROJ Файл проекта.
ICO Файл содержит изображение иконки, которое на форме расположено в верхнем левом углу.
Файл AssemblyInfo содержит информацию о приложении. При создании дистрибутива (установочного пакета) в этот файл помещается информация программы, используемая в технических целях, а также цифровой ключ.

Вся информация о том, из чего состоит наше приложение, находится в файле с расширением .sln (Microsoft Visual Studio Solution File). В этом файле, в частности, написано, как называется наше приложение, и какой файл проекта относится к нему. Файл проекта, имеющий расширение, отражающее выбранный нами язык программирования (WinApp.vcproj в нашем примере), — это XML-файл, содержащий все необходимые характеристики проекта. В частности, здесь есть информация о платформе, для которой создается результирующий файл (OutputType = “WinExe” в нашем примере), о начальном объекте (StartupObject = “WinApp.Form1” в нашем примере), имя корневого пространства имен (RootNamespace = “WinApp” в нашем примере). Отдельный интерес представляет список ссылок на пространства имен, доступных по умолчанию (остальные надо указывать с помощью ключевого слова using), а также список импортируемых пространств имен Список файлов, из которых состоит наше приложение, располагается в секции Files.

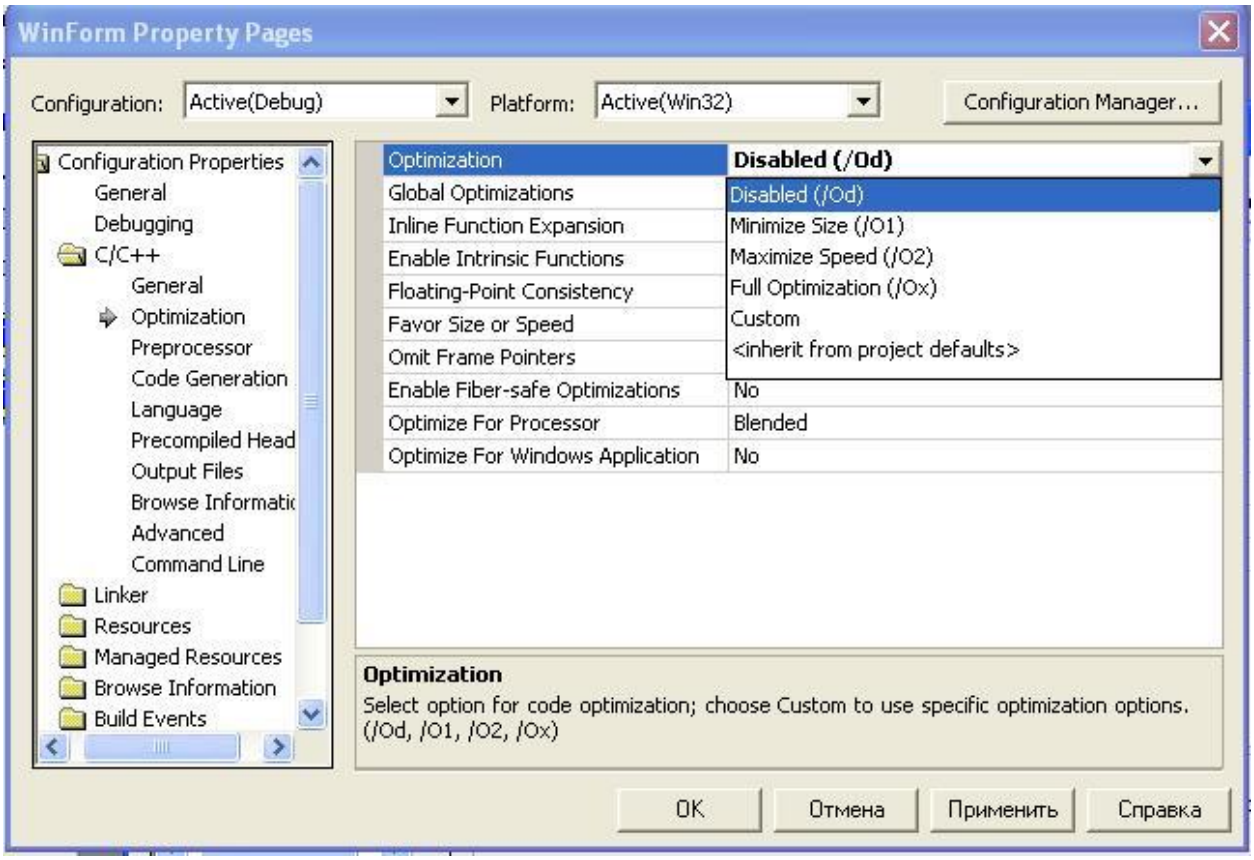
Свойства проекта

В окне Solution Explorer выделяем название проекта, щелкаем правой кнопкой мыши и отбражаемм контексное меню. В контексном меню выбираем пункт Properties.
В появившемся окне содержатся все свойства текущего проекта

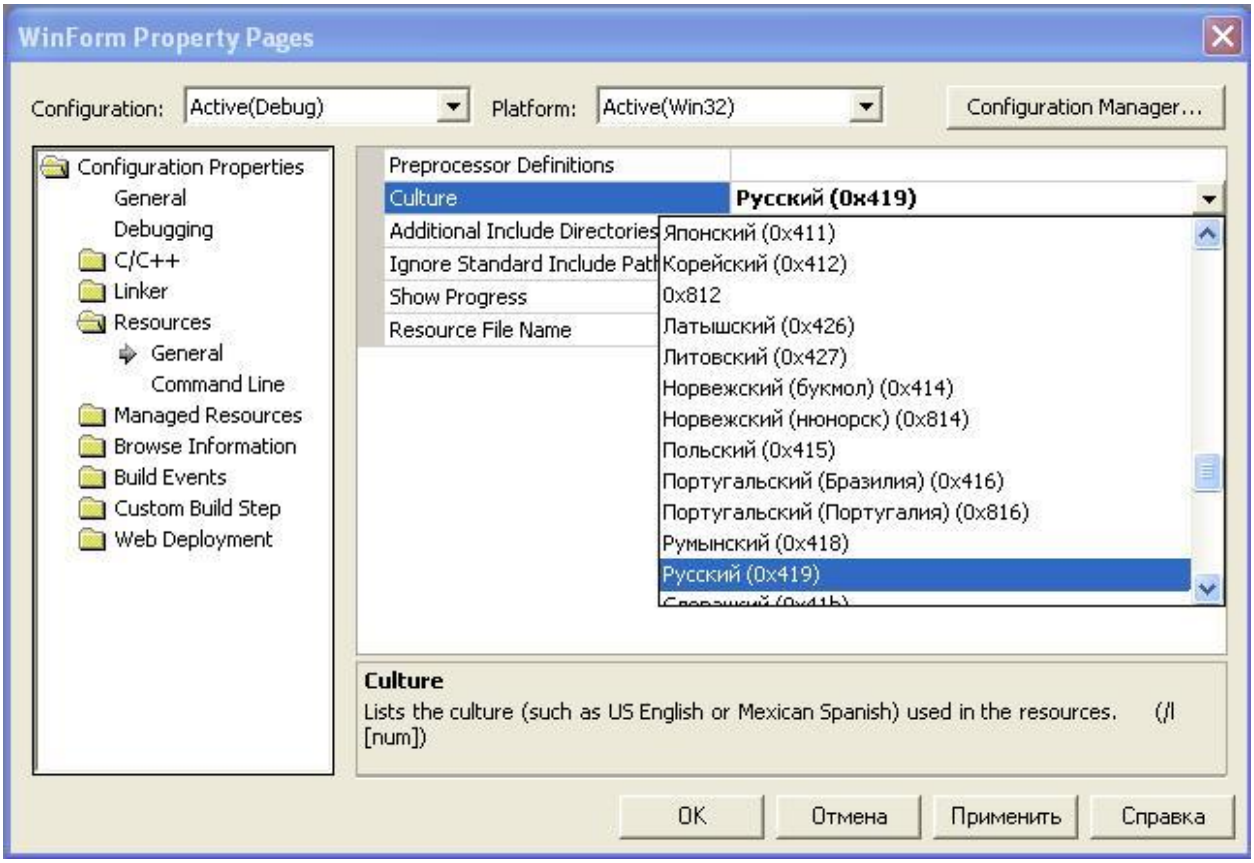


Configuration Type - тип программы, которая получается в резельтате компиляции. Только на языке C++ в Studio Net можно создать статически подключаемую библиотеку.

Optimize Code — оптимизация программы, значение этого свойства может значительно увеличить производительность приложения.

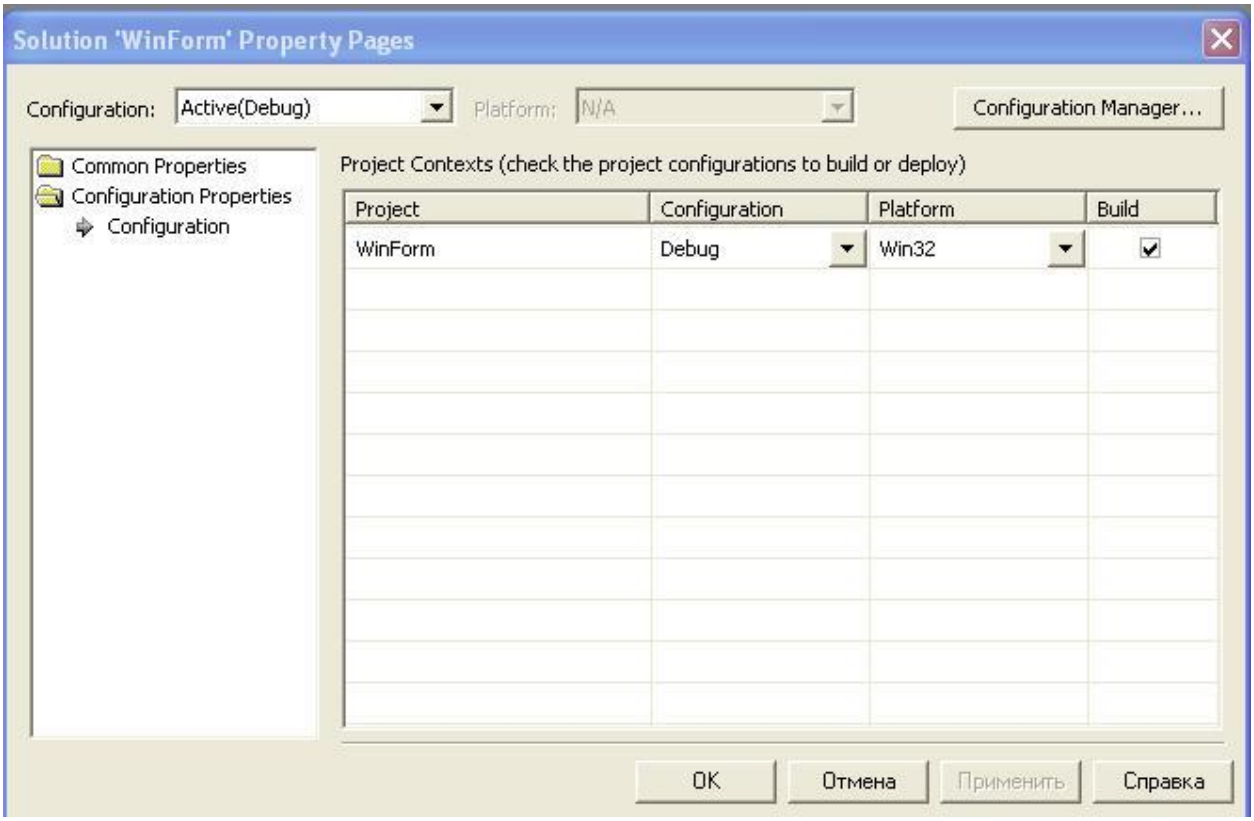


Диалоговое окно позволяет задать параметры культуры для приложения



Конфигурация проектов

В окне Solution Explorer выделяем название ***решения***, щелкаем правой кнопкой мыши и отбражаемм контексное меню. В контексном меню выбираем пункт Properties.
В появившемся окне содержатся общие свойства ***решения***



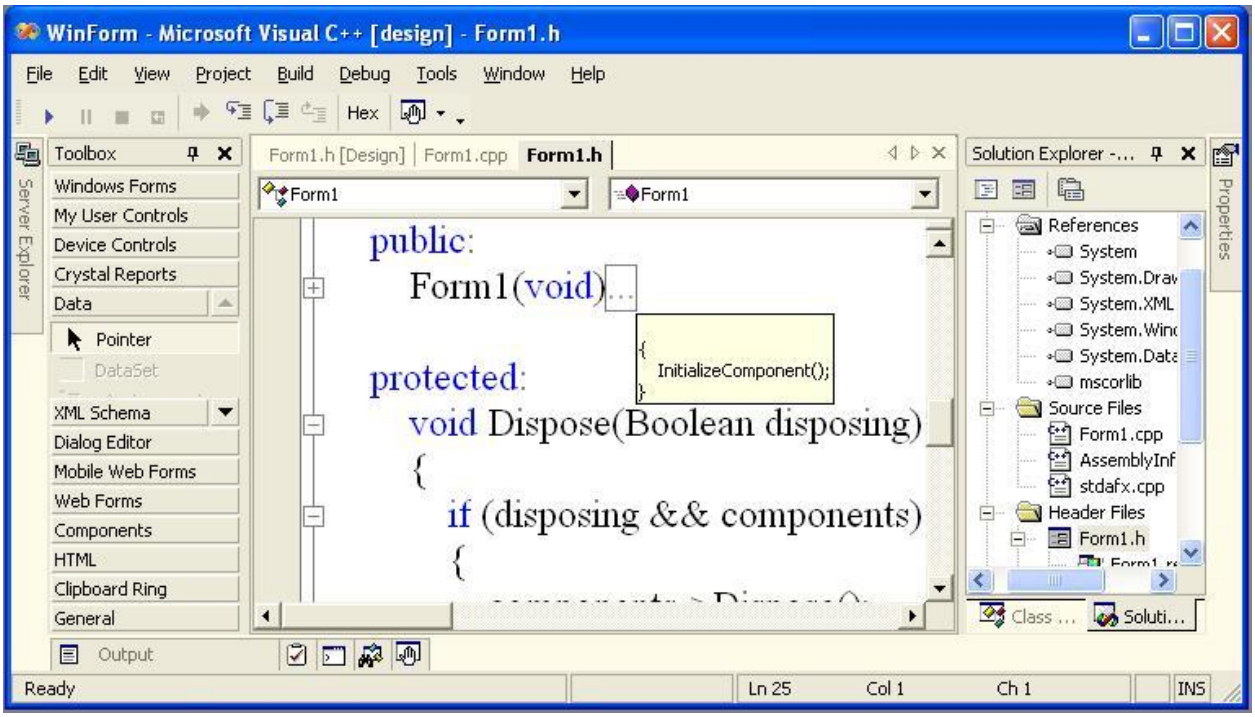
Конфигурация проекта определяет параметры компоновки приложения. Одновремен но может быть определено несколько различных конфигураций, причем приложение для каждой из них будет создаваться в отдельной папке, так что у вас есть возможность срав нить эти конфигурации. Изначально каждый проект в решении Visual Studio имеет две конфигурации — Debug (Отладка) и Release (Выпуск). При использовании конфигурации Debug (Отладка) будет создаваться отладочная версия проекта, с помощью которой можно осуществлять отладку на уровне исходного кода посредством установки точек ос танова и т.д. В папке Debug (Отладка) при этом будет находиться *файл,* используемый компоновщиком для записи отладочной информации о пользовательской программе с целью ускорения редактирования связей в режиме отладки. Этот файл имеет расширение *.pdb* и
содержит отладочную информацию, а также информацию о состоянии проекта. Необходимую конфигурацию можно выбрать с помощью элемента списка Debug (Отладка) на главной панели инструментов. То же самое можно сделать, выбрав пункт меню Builds Configuration Manager (Компоновка Диспетчер конфигурации...), что приведет к запуску диалога Configuration Manager (Диспетчер конфигурации). Из выпадающего списка Active Solution Configuration (Текущая конфигурация решения) выберите пункт Release (Выпуск). Скомпонуйте проект еще раз. Теперь создана вторая версия программы , причем на этот раз она помешается в папку Release (Выпуск).

Редактор кода

Окна документов предназначены для редактирования компонентов проектов. Их взаимное расположение зависит от выбранного режима отображения окон в среде разработки.

Иерархическая структура программного кода

программные модули реализованы в виде иерархической структуры



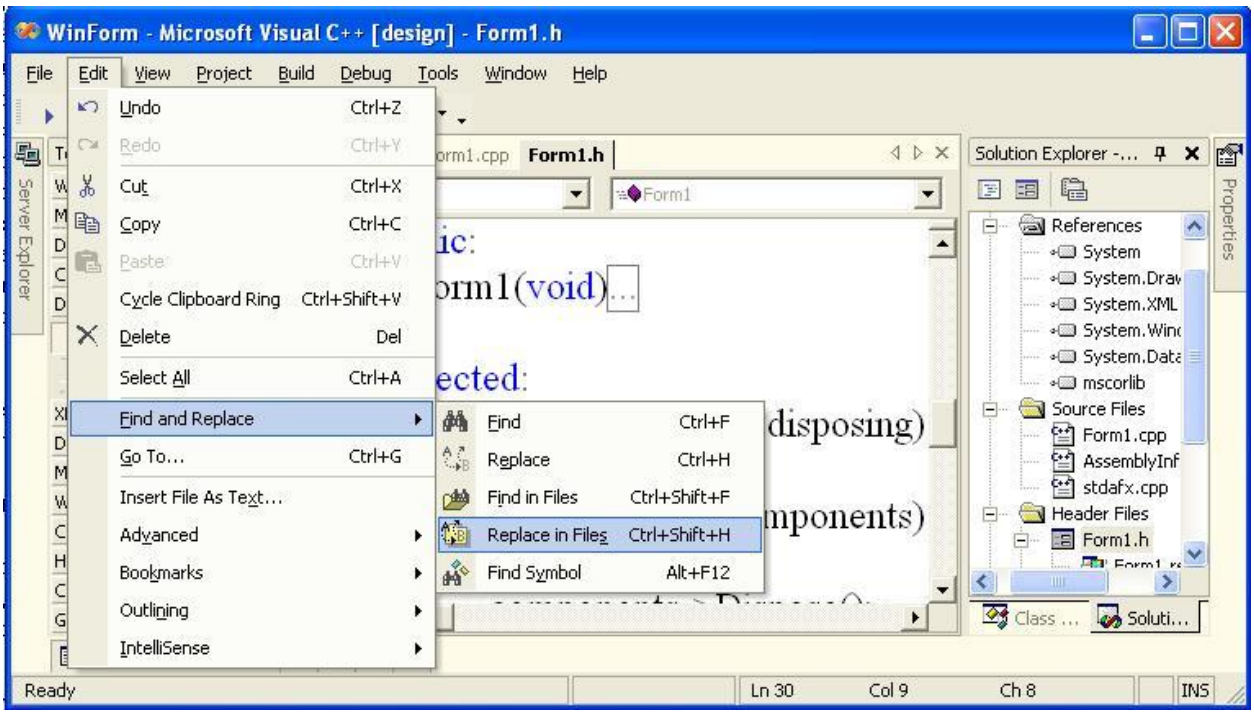
Верхний уровень иерархии определяется операторными скобками. Каждый узел на этой линейке соответствует отдельной процедуре (или аналогичной конструкции). С помощью узлов можно делать видимыми только нужные для работы фрагменты кода. Если мы закроем узел, то в заглавной строке этого блока появится небольшое окошко с многоточием. Подведя к нему мышь, мы сможем увидеть содержимое данной конструкции
очень полезное новшество — использование операторных скобок #Region, которые позволяют группировать в блоки отдельные процедуры

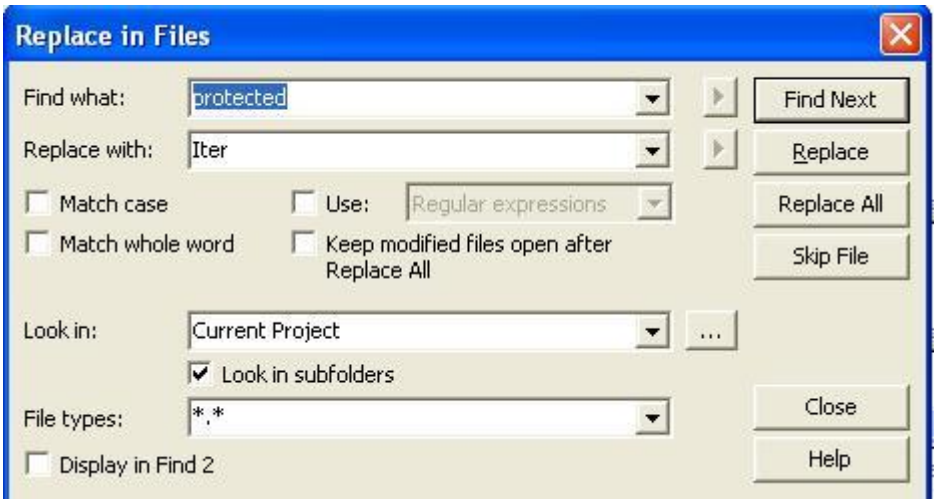
Контекстный поиск и замена

Окно редактирования можно разбить на несколько частей, в которых будут отображаться разные фрагменты кода. Допустимо также отобразить второе окно редактирования с помощью пункта меню Window | New Window.
В редакторе кода можно осуществлять контекстный поиск и замену текста в текущей процедуре, текущем модуле или в выделенном фрагменте кода с помощью стандартной диалоговой панели Windows Find and Replace.

В строке для поиска могут содержаться символы «*» и «?», означающие любую последовательность символов и любой символ соответственно.
Возможен также поиск и замена фрагментов текста во всех файлах проекта. В этом случае

следует использовать диалоговые панели Find in Files и Replace in Files.





Помимо фрагментов кода можно искать также названия классов и структур — для этой цели используется диалоговая панель Find Symbols. Результаты поиска отображаются в окне Find Symbol Results.

В редакторе кода можно установить закладку на какую-либо строку кода и вернуться к ней позже. Закладки не исчезают и при сохранении проекта.

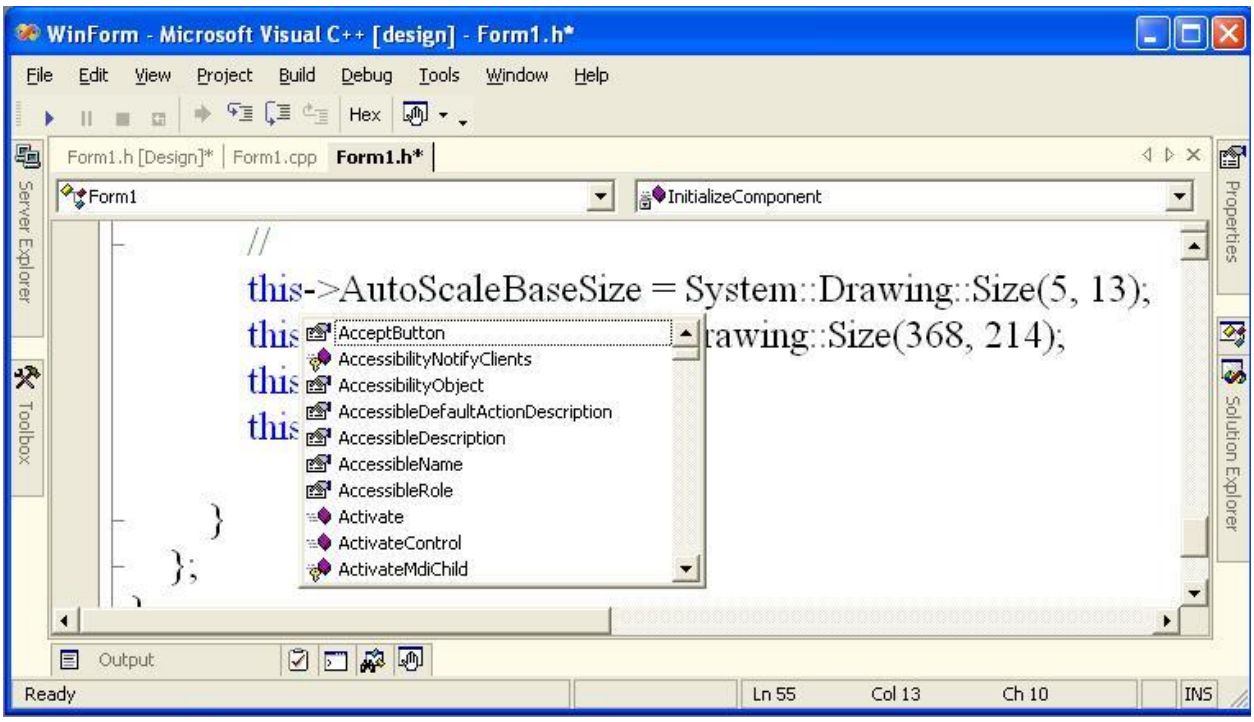
Можно также создать комментарий, связанный с выделенным фрагментом текста, с помощью команды меню Edit | Advanced | Comment Selection.

Возможно перемещение фрагментов текста посредством мыши в другое место, копирование фрагментов, а также перемещение фрагментов текста из редактора кода в окно Toolbox

IntelliSense (выпадающий список-подсказка)

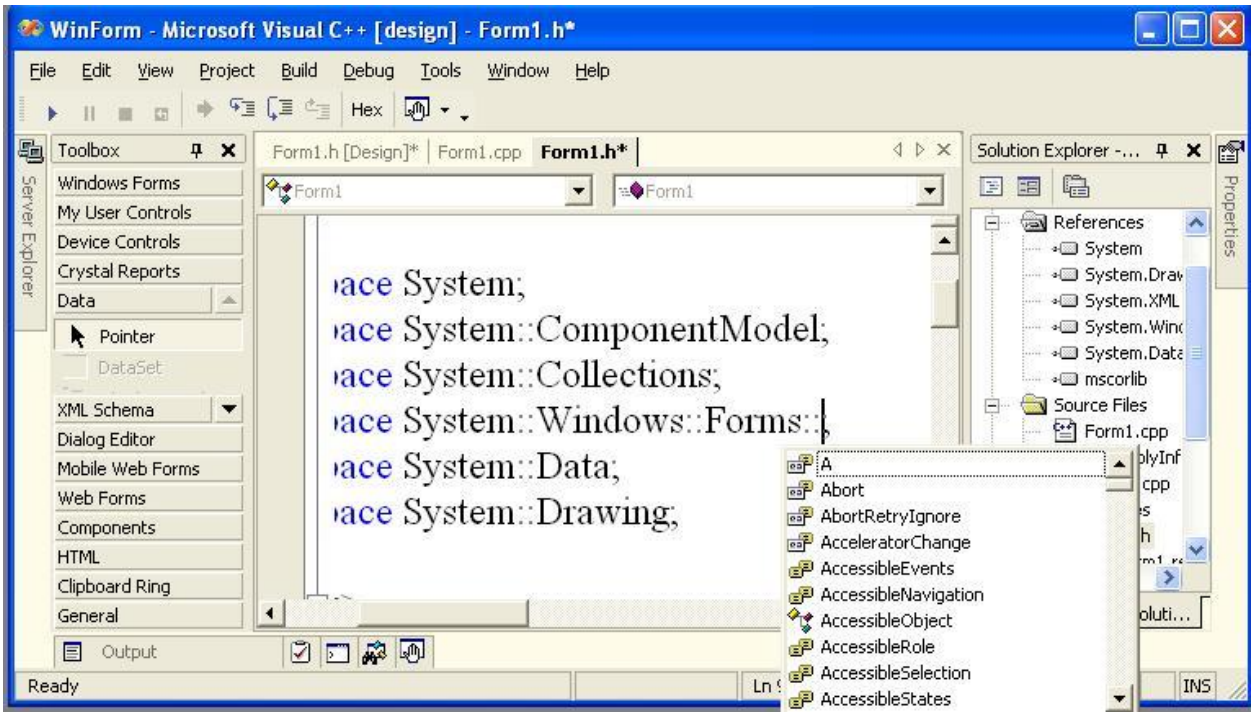
В Visual Studio после набора имени объекта и ввода точки, либо набора имени указателя на объект и стрелки (->) на экране появляется список свойств и методов данного

объекта.



При вводе имени метода (или функции) и круглой открывающейся скобки можно увидеть на экране описание метода и его параметров.

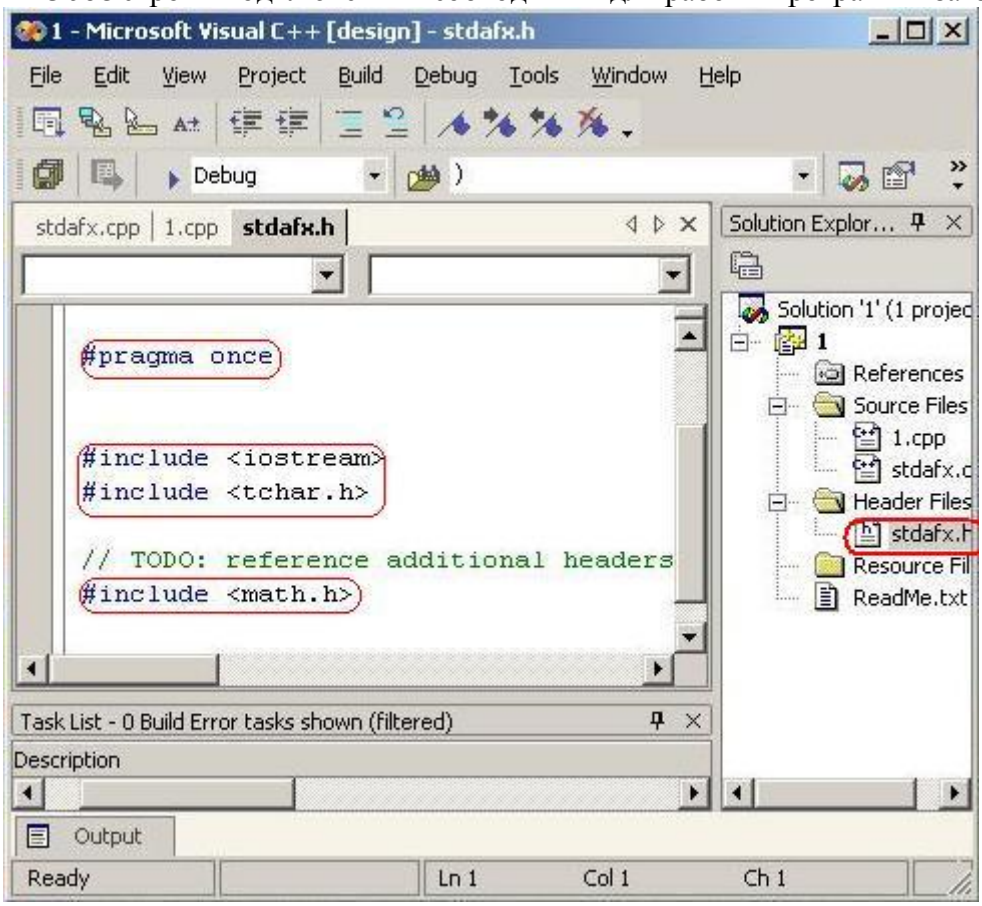
После набора имени пространства имен и :: на экране отображается его содержимое.



ГЛАВА 3. РАЗРАБОТКА КОНСОЛЬНОГО ПРИЛОЖЕНИЯ

Механизм предварительной компиляции заголовочных файлов

Механизм предварительной компиляции заголовочных файлов реализован при помощи пары файлов Stdafx.h и Stdafx.cpp. В файл Stdafx.h добавляются при помощи директивы #include строки подключения необходимых для работы программы заголовочных файлов.



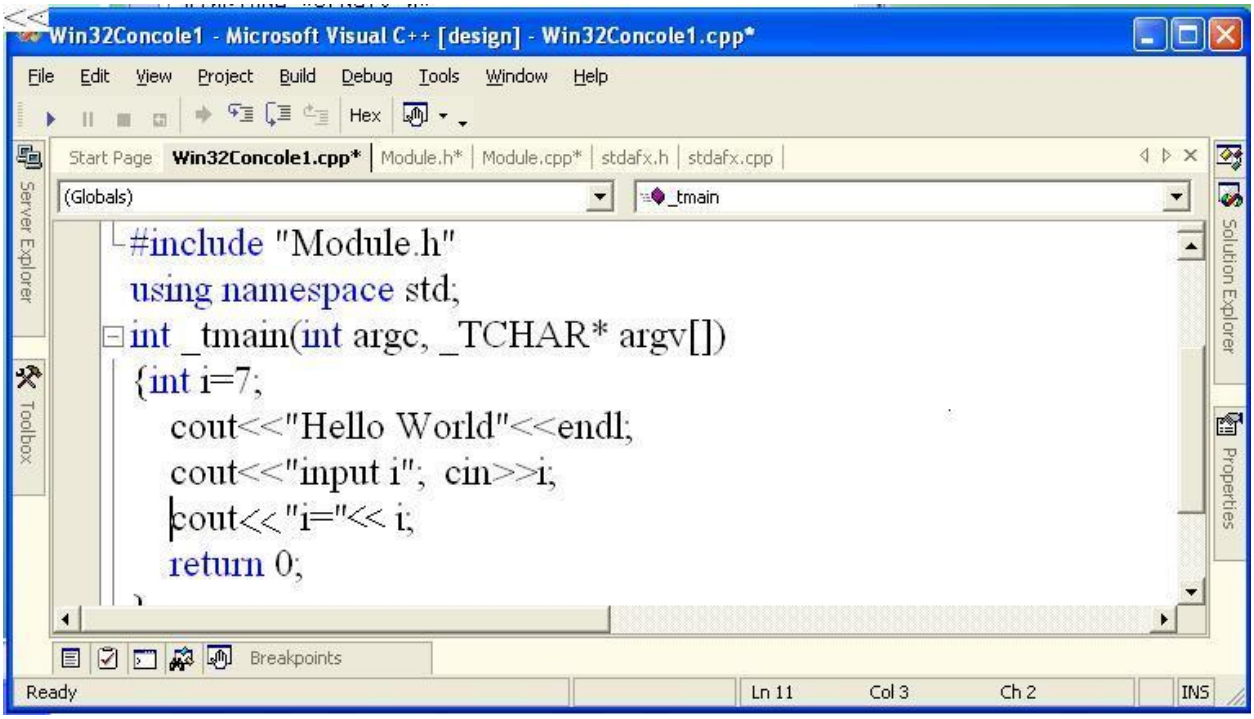
В данном примере подключается заголовочный файл <math.h> для работы с математическими функциями. Строки подключения библиотек <iostream> , <tchar.h > генерируются автоматически при создании шаблона консольного приложения. Директива препроцессора #pragma once обеспечивает включение только единственной копии заголовочных файлов

Сам файл Stdafx.h включается во все файлы реализации программы (*.CPP). Все другие директивы препроцессора, например , #define (определение именованных констант) и #include (подключения заголовочных файлов, не требующих предварительной компиляции) располагаются ниже строки #include <Stdafx.h>. При нарушении этих правил возникает ошибка на стадии компиляции – не найден конец файла.

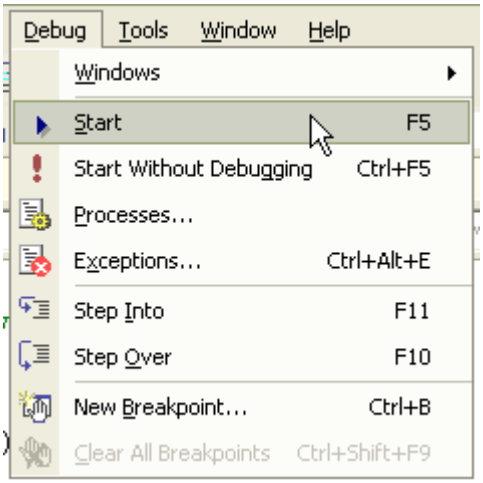
Ввод и вывод данных

В консольном приложении Win32 для ввода и вывода данных можно использовать все функции стандартной библиотеки языка Си, а также средства потокового ввода-вывода языка C++. Причем подключение заголовочных файлов stdio.h и iostream.h не требуется, поскольку сгенерированный шаблон приложения содержит строку #include <iostream>.

Средства потокового ввода-вывода языка C++ принадлежат пространству имен std. Поэтому для доступа к объектам можно явно указывать принадлежность к пространству имен, например, std::cout, либо, что более удобно, сделать доступным все пространство имен при помощи строки
using namespace std;
Данная строка помещается до основной программы.



Компиляция программы



Пункт главного меню Debug

При запуске консольных приложений при помощи команды Start Without Debugging окно с результатами работы программы исчезает только после нажатия клавиши Enter.

ГЛАВА 4. ОТЛАДКА ПРОГРАММЫ

Использование режима останова

Одним из способов обнаружить логическую ошибку является выполнение кода программы по одной строке и изучение изменения содержимого одной или нескольких переменных. Для этого можно при работе программы войти в режим останова, а затем просмотреть код в Редакторе кода. Режим останова дает возможность просмотреть программу во время ее исполнения. Точка останова показывает то место программы, где выполнение будет остановлено, и вы сможете использовать инструменты разработки Visual Studio. Простейший способ установить точку останова— щелкнуть на сером поле слева от окна с исходным кодом программы. Можно также переместить курсор на нужную строку и щелкнуть на кнопке "Точка останова" на панели инструментов. Щелчок на этой кнопке установит точку останова, если ее там не было, и, наоборот, уберет ее, если она уже
была установлена на этой строке. Теперь, если запустить программу в режиме отладки, ее выполнение остановится при достижении точки останова. Желтая стрелка на красном кружке, обозначающем точку останова, указывает, на какой именно точке останова прервано выполнение программы

установить точку останова

Переместите указатель мыши к полосе Margin Indicator (указатель поля - серая полоса, расположенная сразу за левым полем окна Редактора кода) рядом с оператором , а затем щелкните на этой полосе, чтобы установить точку останова. Немедленно появится красная точка останова. .

оператор, который вы выбрали для создания точки останова, выделен желтым цветом, а на полосе Margin Indicator (Указатель поля) появилась стрелка. Теперь Visual Studio находится в режиме останова, и вы можете узнать этот режим по слову "[break]", появившемуся в строке заголовка Visual Studio.

Поместите указатель мыши в Редакторе кода над переменной . Visual Studio выведет сообщение, в котором будет указано значение этой переменной.

Щелкните на кнопке Stop Debugging (Остановить отладку) панели инструментов Debug (Отладка).

Удаление точки останова

1. Щелкните в Редакторе кода на красном кружке, расположенном на полосе Margin Indicator (Указатель поля) и ассоциированном с точкой останова. Точка останова исчезнет. Это все, что касается этой задачи. Заметьте, что если у вас в программе более одной точки останова, вы можете удалить их все, щелкнув на команде Clear All Breakpoints (Снять все точки останова) из меню Debug (Отладка). Visual Studio сохраняет точки останова в вашем проекте, так что важно знать, как удалять их; в противном случае они останутся в вашей программе, даже после ее перезапуска!

2. Щелкните на кнопке Stop Debugging (Остановить отладку) панели инструментов Debug (Отладка). Выполнение программы завершится.

3. В меню View (Вид) укажите на Toolbars (Панели инструментов), а затем щелкните на Debug (Отладка). Панель инструментов Debug (Отладка) закроется.

панель инструментов Debug

При отладке вашего приложения используется панель инструментов Debug (Отладка) - специальная панель инструментов, предназначенная для поиска ошибок. . Ее можно открыть, выбрав команду Toolbars (Панели инструментов) в меню View (Вид), а затем щелкнув на Debug (Отладка).

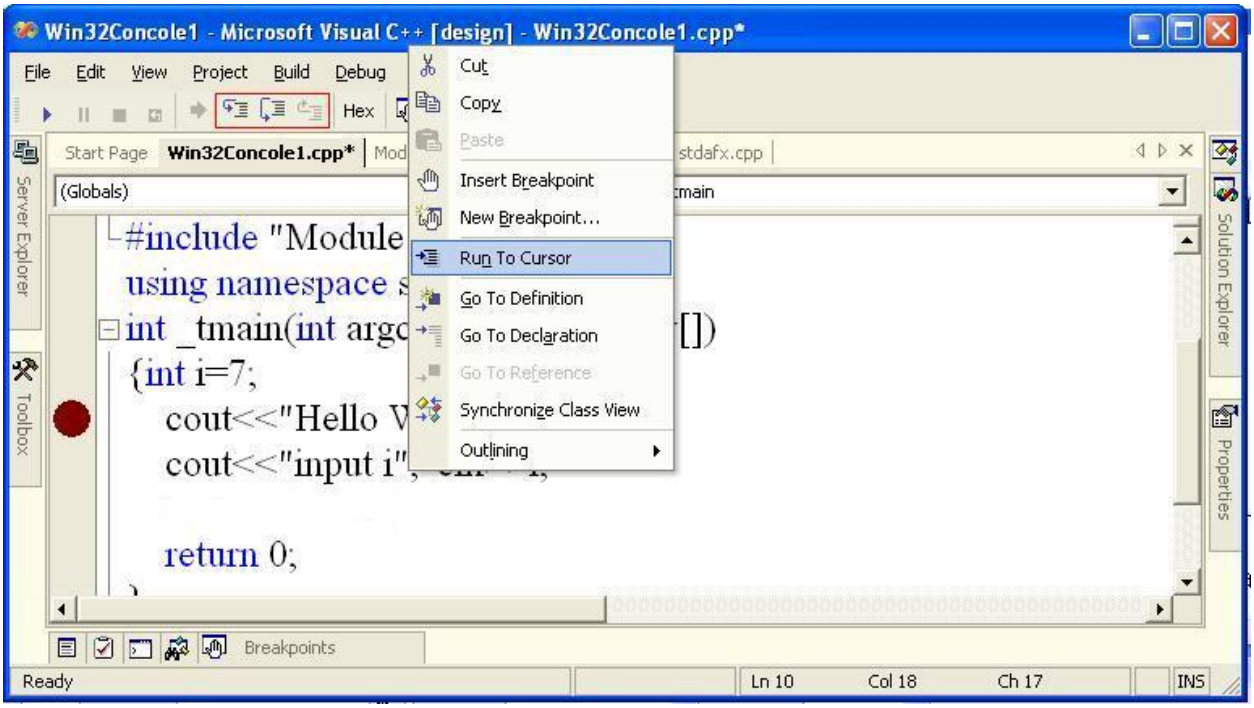
Выполнение в пошаговом режиме

После остановки программы отладчиком можно продолжить ее выполнение в пошаговом режиме. . Есть несколько кнопок, предназначенных для выполнения в пошаговом режиме. Наиболее часто используются следующие из них (в порядке расположения на панели инструментов):

• Step Into (Шаге заходом);
• Step Over (Шаг с обходом);
• Step Out (Шаг с выходом).

Есть еще одна команда контекстного меню— Run to Cursor (Выполнить до текущей позиции).

Если курсор находится на вызове какой-либо функции, то при щелчке на кнопке Step Into (Шаг с заходом) он перейдет на первую строку этой функции. Если же щелкнуть на кнопке Step Over (Шаг с обходом), произойдет вызов функции и курсор переместится на следующую строку



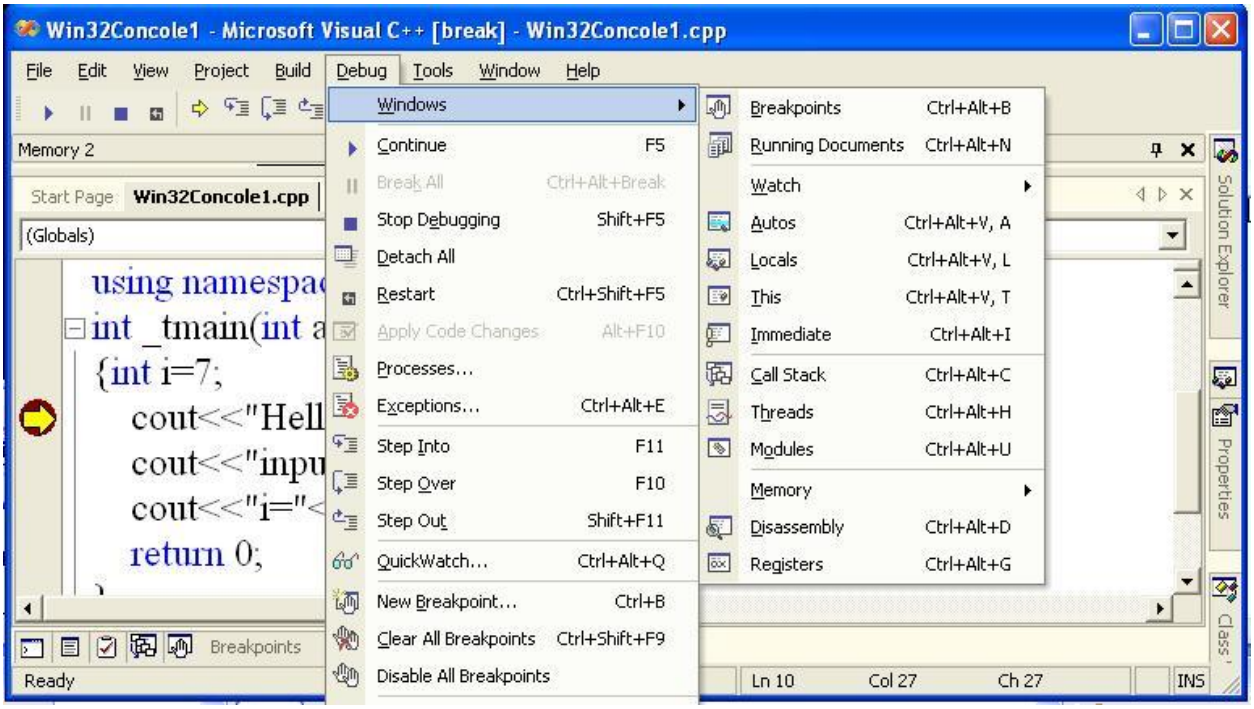
Контрольные значения

В режиме останова можно посмотреть, как обрабатывается логика вашей программы, в часности просмотреть значения переменных. . Простейший способ это сделать навести указатель мыши на переменную, и во всплывающей подсказке (на желтом фоне) около

курсора будет выведено значение переменной. . В режиме останова можно изучить значения свойств и переменных, но нельзя изменить код программы в Редакторе кода - при выполнении программы он заблокирован и защищен. Для просмотра переменных изменения их значений и вычисления выражений используются окна отладки.

Окна отладки

Открываюшийся список, который расположен последним на панели инструментов Debug , дает доступ к полному набору окон отладки, имеющихся в Visual Studio. В интерфейс пользователя Visual Studio .NET добавлено несколько новых окон отладки, включая Autos (Видимые), Command (Окно команд), Call Stack (Список задач), Threads (Потоки), Memory (Память), Disassembly (Дизассемблированный код) и Registers (Регистры).



Окно Quick Walch (Контрольное значение)

Для отображения окна необходимо щелкнуть правой кнопкой мыши на переменной и выбрать пункт Quick Walch (Контрольное значение) во всплывшем меню (или воспользоваться командой Quick Walch в пункте меню Debug). В этом окне можно изменить значение переменной

Окно Autos (Видимые)

Открытие окна Autos

В меню Debug (Отладка) укажите на Windows (Окна), а затем щелкните на Autos (Видимые).

Окно Autos (Видимые) - это автоматическое окно, показывающее состояние переменных и свойств, которые используются в текущий момент. Окно Autos (Видимые) полезно для изучения состояния отдельных переменных и свойств. Но элементы в окне Autos (Видимые) сохраняют свои значения только для текущего (выделенного в

отладчике) и предыдущего оператора (того, который только что был выполнен). Когда программа доходит до выполнения кода, не использующего эти переменные, они исчезают из окна Autos (Видимые).

Окна Watch

Открытие окна Watch

Чтобы видеть содержимое переменных и свойств на протяжении всего выполнения программы, используйте окно Watch (Контрольное значение) - специальный инструмент Visual Studio, который отслеживает нужные значения, при работе в режиме останова. В Visual Studio .NET вы можете открыть до четырех окон Watch (Контрольное значение). Эти окна пронумерованы как Watch 1 (Контрольное значение 1), Watch 2 (Контрольное значение 2), Watch 3 (Контрольное значение 3) и Watch 4 (Контрольное значение 4) и находятся в подменю Watch (Контрольное значение), которое вы можете открыть, выбрав команду Windows (Окна) в меню Debug (Отладка). Также можно добавлять в окно Watch (Контрольное значение) выражения.

Окно команд

Окно Command (Окно команд) - инструмент среды разработки Visual Studio двойного назначения. Когда окно команд находится в режиме Immediate (Интерпретация), вы можете использовать его для взаимодействия с кодом отлаживаемой программы. Когда окно команд находится в режиме Command (командном), вы можете использовать его для исполнения команд Visual Studio, таких, как Save All (Сохранить все) или Print (Печать). Если вы исполняете более одной команды, то можете использовать клавиши со стрелками для просмотра предыдущих команд и их результатов.

Открытие окна команд в режиме Immediate (Интерпретация)

В меню Debug (Отладка) укажите на Windows (Окна), а затем щелкните на Immediate (Интерпретация). Visual Studio откроет окно Command (Окно команд) в режиме Immediate (Интерпретация) - специальном состоянии, которое позволяет вам взаимодействовать с программой в режиме останова. В режиме Immediate (Интерпретация) строка заголовка окна содержит текст "Command Window - Immediate" ("Окно команд: интерпретация"). Если окно команд находится в режиме Command (Командный), вы можете переключить его в режим Immediate (Интерпретация), введя команду **immed**. У режима Immediate (Интерпретация) окна команд много способов применения: он представляет собой великолепное дополнение к окну Watch (Контрольное значение) и поможет экспериментировать с различными тестовыми ситуациями, которые было бы сложно ввести в программу другим способом.

Переключение окна команд в командный режим

Если окно команд находится в режиме Immediate (Интерпретация), вы можете переключить его в командный режим, введя команду **>cmd**. (Символ > обязателен.) Окно команд также может быть использовано для запуска команд интерфейса среды Visual Studio. Например, команда File.SaveAll сохранит все файлы текущего проекта.

Запуск команды File.SaveAll

1. Для переключения в командный режим, введите в окне команд команду >cmd, а затем нажмите на (Enter). Строка заголовка окна команд изменится на

"Command Window" (Окно команд) и в окне появится символ запроса команд ">" (визуальная подсказка о том, что окно находится в командном режиме).

2. Введите в окне File.SaveAll, а затем нажмите на (Enter). Visual Studio сохранит текущий проект, а затем снова появится запрос команды.

Окно команд использует функцию автозавершения написания команд и показывает вам все команды, начинающиеся с символов, которые вы уже ввели. Это мощная функция, с помощью которой можно найти большинство команд, выполняемых с помощью окна команд.

3. Щелкните на кнопке Закрыть окна команд.

ГЛАВА 5. ТЕОРЕТИЧЕСКИЕ ОСНОВЫ РАЗРАБОТКИ WINDOWS ПРИЛОЖЕНИЙ

1. Введение в событийно-ориентированное программирование
    1. События
    2. Сообщения
    3. Обработка событий
2. Введение в объектно-ориентированное программирование на C++
    1. Парадигма программирования
    2. Абстрактный тип данных (АТД)
    3. Классы
        1. Инкапсуляция. Функции-элементы
        2. Сокрытие данных. Открытые, закрытые, защищенные члены класса.
        3. Управление созданием, инициализацией и уничтожением объектов классов. Специальные функции-члены классов: конструктор и деструктор
    4. Наследование
        1. Производные и базовые классы
3. Платформа Microsoft .NET
    1. Общая среда исполнения (Common Language Runtime)
    2. Стандартная система типов (Common Type System)
    3. Структурные и ссылочные типы
        1. Самоописывающие ссылочные типы
            1. Классы
            2. Типы-интерфейсы
        2. Дополнительные элементы системы типов .NET
            1. Структуры и перечисления
            2. Указатели
        3. Упакованные типы-значения
    4. Встроенные типы данных CTS

5. Преобразование типов с помощью класса System::Convert
6. Библиотека базовых классов
    1. Класс System**::**Object

4. Расширения управляемого C++ (Managed C++)
    1. Управляемые типы данных
    2. Класс управляемого C++
    3. Свойства в .NET
    4. Делегаты
    5. Реализация обработки событий в .NET
    6. Управляемые массивы.
    7. Класс System::Array
    8. Управляемые двумерные массивы
5. Работа со строками в Windows
    1. Использование кодировки UNICODE
        1. Представление символов в UNICODE
        2. Разработка универсальных программ для работы со строками
        3. Перекодирование однобайтовых символов в Unicode и обратно с учетом кодовой страницы
        4. Пример перекодирования символов Unicode в однобайтовые
    2. Класс System::String
        1. Инициализация строк при помощи строковых констант
        2. Массивы строк
        3. Сравнения объектов класса String
        4. Методы класса String
    3. **С**троковое представление данных
    4. Форматирование строк
        1. Строки стандартных числовых форматов
        2. Составное форматирование
        3. Синтаксис элементов форматирования

4. Разбор строк

Введение в событийно-ориентированное программирование

События

Консольное приложение выполняется линейно, ожидая от пользователя ввода данных или команды и блокируя все другие действия. Логика работы приложения Windows называется *логикой, управляемой событиями*. С помощью событий приложения Windows получают уведомления о том, что что-то произошло. Например, при нажатии на кнопку мыши приложение, в окне которого вы произвели это действие, будет уведомлено об этом событии. То же самое происходит, когда вы делаете что-то с помощью клавиатуры. Событийная модель стала основой современного программирования.

Сообщения

Главной *особенностью логики, управляемой событиями,* является отделение чтения входных данных от функций, в которых производится их обработка. Приложения Windows имеют централизованный механизм чтения входных данных. Входные данные упаковываются в записи, которые называются ***сообщениями***, и после этого распределяются по соответствующим приложениям и отображаемым объектам. Передача сообщение Windows – это механизм, на основе которого организован обмен информацией между приложениями, либо модулями одного и того же приложения. Windows содержит в себе системную очередь сообщений. Туда поступают сообщения от драйверов устройств ввода/вывода и приложений. Приложения также имеют и свою очередь сообщений. Windows направляет сообщение от используемого органа управления в очередь ого приложения, которому он принадлежит. Т.о. любое приложение занимается обработкой своей очереди сообщений, а Windows направляет сообщения в нужную очередь.

В Windows используется многоуровневая система сообщений. Сообщения низкого уровня вырабатываются, когда происходит перемещение мыши или нажатие клавиши. Эти сообщения передаются Windows, в котором формируются сообщения более высокого уровня. В результате приложения получают сообщения о том, какой орган управления был выбран и каков его идентификатор. Т.о. Windows осуществляет работу по привязке мыши и клавиатуры к органам управления.

Обработка событий

Приложение **Windows** может связать с событием функцию, которая будет вызываться в случае, если данное событие произойдет. Такая функция называется обработчиком события. Обработчик события вызывает сама операционная система. При вызове обработчик события получает всю информацию о произошедшем событии в виде сообщения.

Введение в объектно-ориентированное программирование на C++

1. Парадигма программирования
2. Абстрактный тип данных (АТД)
3. Классы
    1. Инкапсуляция. Функции-элементы

2. Сокрытие данных. Открытые, закрытые, защищенные члены класса.

3. Управление созданием, инициализацией и уничтожением объектов классов. Специальные функции-члены классов: конструктор и деструктор

4. Наследование

1. Производные и базовые классы

Парадигма программирования

Любой язык программирования поддерживает определенный стиль или парадигму программирования. - это модель разработки и реализации программ. Согласно процедурному стилю программирования программа представляет собой набор взаимосвязанных процедур. Усложнение приложений выявило недостатки традиционных процедурно-ориентированных языков, связанные прежде всего с их низкой надежностью и выразительной способностью. Поэтому в области разработки программ акценты сместились от разработки процедур к организации данных.

Абстрактный тип данных (АТД)

В связи с необходимостью разрабатывать структуру алгоритма и данных одновременно возникает понятие абстрактного типа данных (АТД). Свойства АТД не зависят от его внутреннего устройства, а определяются набором операций над ним. Это позволяет отделить описание структуры типа от разработки его конкретного представления. Механизм реализации АТД базируется на идеях инкапсуляции и сокрытии данных. Инкапсуляция - это соединение данных и обрабатывающих их процедур в единое целое. Сокрытие данных выражается в возможности доступа к данным только функциям, определенным в интерфейсе типа. Сокрытие данных служит для достижения независимости кода пользователя при изменении скрытого представления объектов типа. Таким образом, АТД - это инкапсулированный тип данных, скрывающий детали реализации. Метод разработки программ с представлением в ней прикладных понятий из приложений как АТД называется абстракцией данных.
Одним из первых средств поддержки абстракции данных становится модуль. Модуль - это набор данных и соответствующих им функций, размещенный в отдельном файле. Модуль имеет свое собственное пространство имен. Построение программы в виде множества модулей базируется на механизме раздельной компиляции. Концепция модуля, обеспечивая сокрытие данных, в тоже время не поддерживает механизм абстракции данных в полном объеме. Ограниченность модуля для поддержки абстракции данных заключается в том, что модуль не является элементом языка. Типы, создаваемые как модули, в отличие от встроенных, не обрабатываются компилятором.

Классы

В C++ абстрактные типы данных реализованы с помощью классов. Классы - это определяемые пользователем типы данных, которые расширяют и дополняют стандартные типы базового языка. Механизм реализации АТД основан на объединении в классе данных и обрабатывающих их процедур и на возможности явного управления доступа к данным. Классы, как АТД, отражают понятия из прикладной области. Экземпляры классов называются объектами. В отличие от процедурного стиля программирования в ООП программа рассматривается как реализация классов, а не реализация алгоритмов. Поэтому первый шаг применения ООП состоит в нахождении

наиболее подходящих классов.

Для определения класса используется ключевое слово class. Построение класса как элемента языка основано на обобщении понятия структуры. Обобщение можно условно разделить на несколько этапов или шагов.

Инкапсуляция. Функции-элементы

Первым шагом на пути обобщения понятия структуры является использование функций как элементов структур. С помощью определения в одной структуре функций, обрабатывающих данные, вместе с представлением самих данных в С++ реализуется механизм инкапсуляции АТД.

Сокрытие данных. Открытые, закрытые, защищенные члены класса.

Конструкция структуры, усиленная по сравнению с Си, обеспечила связь между данными и функциями. Но эта связь не модульная и не прочная, как могла бы быть. Любая другая функция может читать и изменять данные.
В С++ для предотвращения неконтролируемого доступа к данным вводится новый механизм управления. Сокрытие данных обеспечивается с помощью закрытых, открытых и защищенных разделов описания. В качестве меток, отделяющих разделы, применяются специальные ключевые слова: **private(закрытый), protected (защищенный), public (открытый).**
Таким образом, внутреннее представление класса невидимо (скрыто) и не может быть изменено. Доступ к данным ограничен явно объявленным списком функций, которые представляют собой интерфейс класса.

Управление созданием, инициализацией и уничтожением объектов классов. Специальные функции-члены классов: конструктор и деструктор

Создание объекта связано с выделением памяти и заданием начального значения. Объявление объекта класса автоматически приводит к выделению необходимого объема памяти. В результате ограничения доступа инициализация закрытых данных классов возможна только с помощью функций-членов классов. Поэтому класс должен содержать функцию, специально предназначенную для инициализации.

В С++ для создания объектов классов вводится специальная функция-член класса, которая называется **конструктором**. Имя конструктора совпадает с именем класса. Конструктор не имеет возвращаемого значения. Вызов конструктора осуществляется автоматически компилятором при создании объектов класса. Класс может содержать несколько конструкторов. Конструкторы подчиняются тем же правилам, что и перегружаемые функции.

Конструктор может использоваться для управления распределением памяти под элементы объектов классов. Для этого в конструкторе производится обращение к свободной памяти с помощью оператора new. Если в конструкторе было обращение к свободной памяти, то при уничтожении объектов класса эта память должна быть освобождена явным вызовом оператора delete. Для освобождения памяти, выделенной под элементы объекта класса, до разрушения самого объекта в С++ вводится специальная функция-член класса, которая называется **деструктор**. Вызов деструктора выполняется

компилятором автоматически при выходе переменной из области действия. Имя деструктора совпадает с именем класса с предшествующим символом ~ (тильда). Деструктор не имеет параметров и возвращаемого значения.

Наследование

Компонентами ОПП помимо абстракции данных являются наследование и полиморфизм. АТД - это в некотором роде черный ящик. Используя только это понятие, невозможно отделить общие свойства от конкретных свойств множества родственных типов. ООП содержит механизмы выражения взаимосвязи и отношений между классами, которые поддерживаются компилятором на уровне языка. Основным из них является наследование.

Производные и базовые классы

Общность между классами в C++ выражают понятия базового и производного классов. Производный класс добавляет к уже имеющемуся базовому классу новые данные и функции без перепрограммирования и перекомпиляции базового. Отношение между базовым и производным классами называется отношением наследования. Отношение наследования выражает тот факт, что производный класс является разновидностью базового. При программировании это проявляется в возможности использовать указатель на производный класс там же, где и указатель на базовый класс. Вхождение базового класса в производный также, как и вхождение любого другого элемента класса , может быть объявлено с модификатором доступа. Доступ к элементам базового класса возможен по их именам. Параметры конструктора базового класса указываются в определении конструктора производного класса.

Производный класс сам может использоваться в качестве базового и так далее. Такое множество родственных классов называется иерархией классов. Наследование позволяет создавать иерархии связанных типов, совместно использующих код и интерфейс. Следовательно, ООП является расширением абстракции данных, которое заключается в представлении программы в виде иерархически организованных классов.

Платформа Microsoft .NET

1. Общая среда исполнения (Common Language Runtime)
2. Стандартная система типов (Common Type System)
3. Структурные и ссылочные типы
    1. Самоописывающие ссылочные типы
        1. Классы
        2. Типы-интерфейсы
    2. Дополнительные элементы системы типов .NET
        1. Структуры и перечисления
        2. Указатели
    3. Упакованные типы-значения

4. Встроенные типы данных CTS

5. Преобразование типов с помощью класса System::Convert

6. Библиотека базовых классов

    1. Класс System**::**Object

Платформа Microsoft .NET представляет собой единую среду исполнения программ и поддержки их разработки.
Ключевыми задачами при построении платформы .NET являлись:

1. Поддержка разработки распределенных корпоративных приложений, включая серверные и мобильные компоненты, на базе Web-сервисов и XML.

2. Унификации библиотек функций для всех приложений, независимо от используемого языка программирования.

Платформа .NET основана на единой объектно-ориентированной модели; все сервисы, предоставляемые программисту платформой, оформлены в виде единой иерархии классов. Модель платформы .NET существенно упрощает разработку приложений по сравнению с программированием для Windows-платформ, где практически вся функциональность предоставлялась разработчику как неструктурированный набор функций в Windows API.
Проблема создания объектно-ориентированной надстройки над функциями Windows API решалась независимо в различных языках программирования. С появлением платформы .NET впервые в истории программирования применяется единая модель, позволяющая на равных пользоваться различными языками для создания приложений. Поскольку базовые классы .NET стали общие для всех систем программирования, то это означает, что изменилось соотношение язык - базовые функции. Если раньше каждый язык содержал свою библиотеку классов, представляющих надстройку над функциями Windows API, например MFC в VC++, то теперь язык программирования адаптируется к платформе. Библиотека классов MFC - это набор статических объектных модулей . Они подключаются к приложению на этапе компоновки исполняемого модуля программы и становятся при этом его неотъемлемой частью. В то же время .NET Class Library - это динамические библиотеки классов, которые являются составной частью операционной среды (специальным видом исполняемых модулей) и используются приложением только в момент его выполнения. Разработчики .NET лишены возможности выбора между статическими объектными библиотеками (LIB) и библиотеками классов (DLL) (за исключением тех, кто пишет на C/C++, которые занимают особое положение в средствах разработки .NET).

3. Повышения управляемости приложений с точки зрения эффективного использования ресурсов и безопасности.

Управление ресурсами в .NET.

Платформа .NET предоставляет автоматическое управление ресурсами. Общая для всех языков среда исполнения управляет данными приложения, т. е. их структурой, и ссылками на объекты в рамках приложения, освобождая ссылки, когда потребность в них пропадает.

Аавтоматическое управление памятью в реализовано в .NET при помощи сборщика мусора (garbage collection). . Это решает многие распространенные проблемы, такие как утечки памяти, повторное освобождение ресурса и т.п

Безопасность в .NET

Код, сгенерированный для .NET, может быть проверен на безопасность. Это гарантирует, что приложение не может навредить пользователю или нарушить функционирование операционной системы . Безопасность является краеугольным камнем .NET. На всех этапах создания и выполнения программ происходят самые различные проверки - от проверки прав на доступ к коду до разрешений на ресурсы. Вот некоторые из типов проверок безопасности:

Безопасность типов. Программы, гарантирующие безопасность данных, обращаются только к тем участкам памяти, которые были выделены для них. Доступ к объектам осуществляется только через специальные интерфейсы, в которые встроены проверка безопасности. В целом, безопасность типов может быть проверена не всегда; однако ее наличие гарантирует невозможность одной из самых распространенных атак (чтение указателя большего размера, чем выделенная память).

Подлинность кода. Загрузчик классов сохраняет информацию об исходных текстах всех классов, которые были загружены. Таким образом, можно восстановить некоторые атрибуты кода (откуда загружен код, кто является автором и т.п.). Эту информацию можно использовать для дачи прав на запуск.

Разрешения на доступ к ресурсам. Ресурсы обычно ассоциированы с системой. В качестве ресурсов могут выступать файлы, сетевые соединения, право вызова неуправляемых API (unmanaged APIs). Отметим, что права доступа проверяются не только для вызвавшей сборки, но и для всех прочих, находящихся в данный момент в стеке вызовов. Это позволяет предотвратить классическую атаку, в которой неавторизованный компонент получает доступ к ресурсу путем обращения к нему через вызов компоненты с другими правами доступа.

Декларативная безопасность. Данный механизм предоставляет возможность встраивать проверки безопасности прямо в код путем аннотации классов, полей или методов. Проверка может производиться однократно при загрузке или постоянно (скажем, при каждом запуске метода).

Императивная безопасность. Обычный код внутри разрабатываемого метода, который проверяет права на данную операцию во время запуска. Такие проверки важны для доступа к файлам, пользовательскому интерфейсу и т.п.

Безопасность в .NET

- Другие модели обеспечения безопасности:
  - Модель политик доступа
  - Модель ролей
- Повышенная важность безопасности при удаленном выполнении
- Криптографические методы защиты, готовые для встраивания в

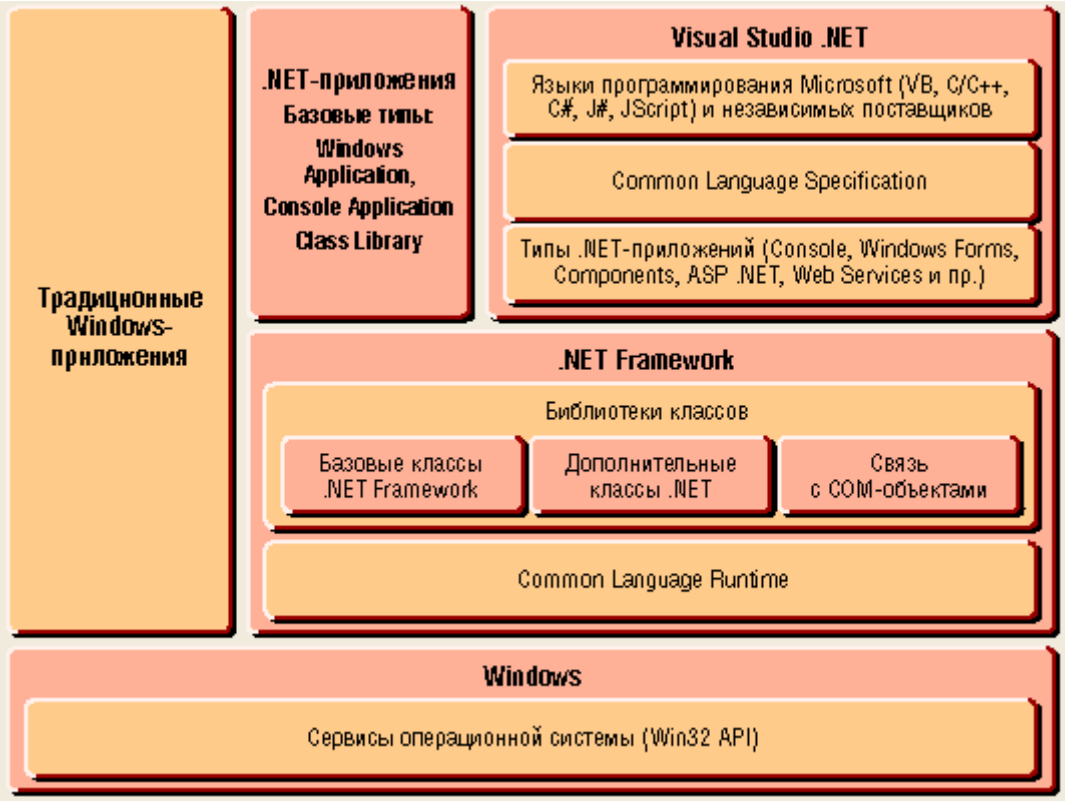
пользовательские приложения

Упомянем еще два способа обеспечения безопасности и защиты приложений в .NET:

Модель политик доступа. Политики позволяют автоматически присваивать коду определенные привилегии, основываясь на том, откуда были получены исходные тексты данного приложения (понятно, что локальному коду пользователи доверяют больше, чем программам из Интернета). Пользователи могут изменять эти политики.

Модель ролей. Эта модель безопасности проверяет, в какой роли выступает пользователь и разрешает/отказывает в доступе в зависимости от этого. Например, в финансовых приложениях максимальный лимит транзакции может зависеть от служебного положения банковского служащего.

Структура Платформы Microsoft.NET



Главным инструментом создания приложений является Visual Studio .NET, в котором каждый из языков программирования взаимодействует с .NET Framework через общий интерфейс. . В состав VS.NET входит несколько языков Microsoft, среди которых важнейшая роль отводится C/C++, C# и VB.NET. Создание универсальной среды разработки и общих базовых функций предопределило то, что отныне все языки программирования Microsoft поставляются в виде единого пакета. Среда разработки для них становится стандартом.

Какой язык из представленного в этой системе набора выбрать для решения своей задачи определяет программист. Несмотря на то ,что эти языки используют одну и ту же библиотеки базовых классов , они обладают различными возможностями. C/C++ позиционируется как достаточно автономный инструмент создания системных средств, в том чичле и вне среды .NET. Он может использоваться для разработки эффективных

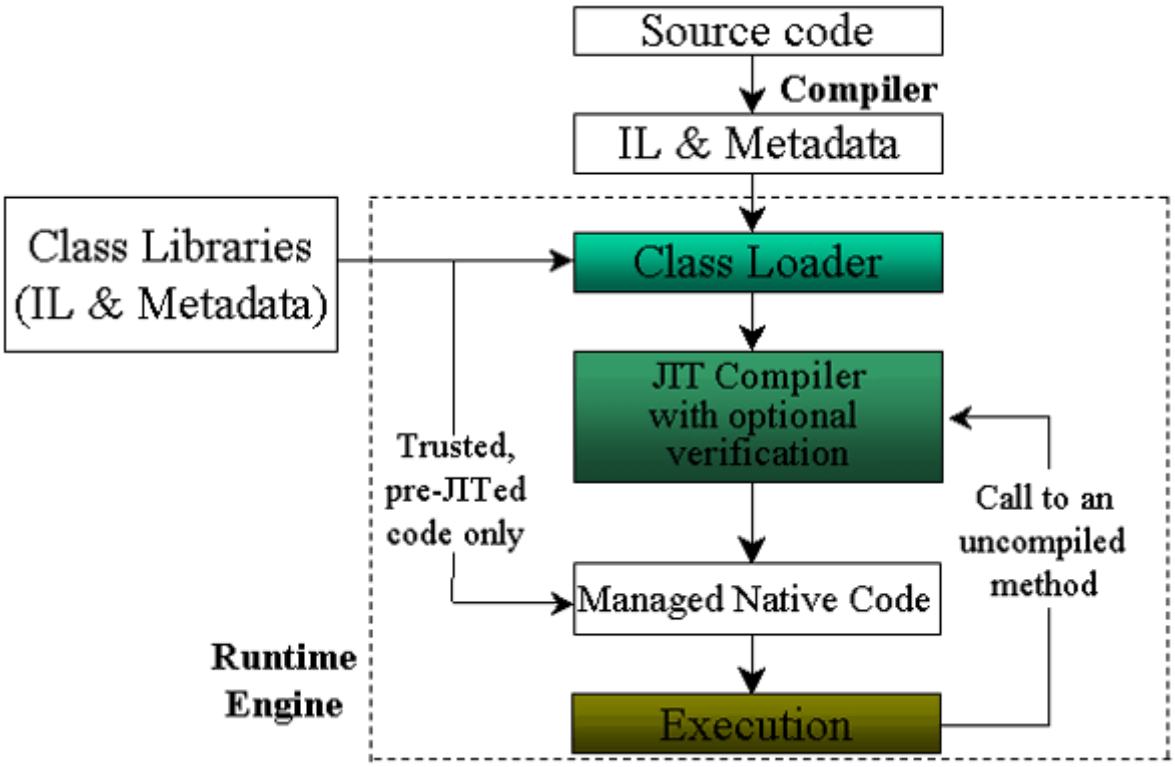
серверных приложений. C# и VB.NET используются исключительно для разработки .NET-приложений. Причем в наибольшей степени возможности среды разработки доступны программистам на C#. Применением C# и VB.NET может служить разработка клиентских и офисных приложений. Используемые ранее для оофисных приложений разнородные средства VB 6 и VBA больше не поддерживаются. Другим направлением применения C# и VB.NET является разработка Web-сервисов.

Один из основных элементов архитектуры Microsoft .NET является NET Framework. NET Framework эта среда, которая представляет собой дополнительный операционный слой, разделяющий приложения пользователя и базовые сервисы Windows. Среда .NET Framework, задающая единый каркас многоязыковой среды разработки приложений, во многом ориентирована на компонентное программирование.
NET Framework состоит из двух главных составляющих: Base Class Library (библиотеки базовых классов) и CLR (Common Language Runtime — общая для языков среда исполнения NET-приложений)

Общая среда исполнения (Common Language Runtime)

Библиотека классов - это статическая составляющая каркаса. Динамическая составляющая задает общую среду выполнения CLR. Роль этой среды весьма велика - в ее функции входит управление памятью, потоками, безопасностью, проверка кода на соответствие CLS, компиляция из промежуточного байт-кода в машинный код и многое другое. Важный элемент CLR - мощный механизм сборки мусора (garbage collector), управляющий работой с памятью типа "куча. На основе CLR решаются основные задачи повышения надежности и безопасности программ, а также платформенной независимости. Все исполняемые модули .NET-приложений (будем называть их CLR-модулями) состоят из исполняемого кода и метаданных. Метаданные (например, различные декларации полей, методов, свойств и событий) широко применяются и в COM-технологии, что и составляет ее основное отличие от обычных двоичных DLL. В CLR состав метаданных значительно расширен, что позволяет более эффективно контролировать версии, проверять надежность источников поступления программ и т. п.

CLR-модули реализуются не в виде машинного кода (native, "родного" для данного компьютера), а с помощью так называемого байт-кода по спецификациям промежуточного языка Microsoft Intermediate Language (MSIL). Иными словами, каждый совместимый с .NET-компилятор должен преобразовать исходный код на языке высокого уровня в двоичный MSIL-код, который уже затем будет исполняться в среде CLR.



В отличие от Java, CLR будет выполнять код не в режиме классического интерпретатора, а путем предварительной компиляции в машинный код отдельных фрагментов программы или целого приложения. Первый вариант - основной, при этом применяется так называемый Just-In-Time компилятор, который выполняет преобразование MSIL в машинный код по мере обращения к соответствующим процедурам (т. е. неиспользуемые фрагменты программы вовсе не компилируются).

MSIL можно рассматривать как ассемблер некоторой виртуальной машины. Это нетипичный ассемблер, так как он обладает многими конструкциями, характерными для языков более высокого уровня: например, в нем есть инструкции для описания пространств имен, классов, вызовов методов, свойств, событий и исключительных ситуаций. Кроме того, MSIL является стековой машиной со статической проверкой типов; это позволяет отслеживать некоторые типичные ошибки.

MSIL представляет собой дополнительный уровень абстракции, позволяющий легко справляться с переносом кода с одной платформы на другую, в том числе, и с изменением разрядности платформы: в отличие от Java bytecode MSIL не завязан на 32 бита или какую-либо другую фиксированную разрядность. В данный момент существуют версии MSIL для мобильных 16-разрядных устройств (.NET Compact Framework), стандартная 32-разрядная версия и специальная версия для работы с получающими все более широкое распространение 64-разрядными устройствами. Благодаря тому, что промежуточное представление .NET не привязано к какой-либо платформе, приложения, созданные в архитектуре .NET, являются многоплатформенными.

Отметим, что MSIL сохраняет достаточно много информации об именах, использованных в исходной программе: имена классов, методов и исключительных ситуаций сохраняются и могут быть извлечены при обратном ассемблировании. Однако

извлечение из MSIL исходных текстов путем дизассемблирования вряд ли имеет смысл, так как имена локальных переменных, констант и параметров сохраняются только в отладочной версии.

Стандартная система типов (Common Type System)

Еще один строительный блок платформы .NET — это Common Type System (CTS, стандартная система типов). Одна из основных целей .NET -обеспечить высокую степень совместимости различных компонентов независимо от языков, на которых они созданы И делается это по двум направлениям: путем предоставления всем компонентам общей среды исполнения и путем создания общей системы типов. Стандартная система типов полностью определяет, как одни типы данных могут взаимодействовать с другими и как они будут представлены в формате метаданных .NET.
Система типов в языке программирования разрабатывается для того чтобы можно было осуществлять статическую проверку программы. Она представляет собой набор правил, определяющих условия, при которых конструкции языка не вызывают запрещенных ошибок. Так как платформа .NET спроектирована с учетом поддержки разных языков программирования, то ее общая система типов (Common Type System - CTS) является объединением систем типов основных распространенных в настоящее время языков. Из этого следует, что все языки платформы .NET (объектно-ориентированные, процедурные, функциональные) совместно используют единую систему типов, и это обеспечивает взаимодействие программных компонентов, написанных на разных языках.

Наличие в .NET общей системы типов позволяет осуществлять статическую проверку программы не только на уровне компилятора, но и на уровне системы выполнения. Другими словами, система может проводить верификацию двоичных исполняемых файлов непосредственно перед их запуском. Это гарантирует безопасность кода, выполняемого в среде .NET и тем самым обеспечивает возможность автоматического управления памятью (сборку мусора).

При создании типов в библиотеках программ, которые можно использовать на всех языках программирования .NET, необходимо следовать синтаксическим конструкциям Common Language Specification . Common Language Specification (CLS) — это набор правил, определяющих подмножество общих типов данных. Если в библиотеке классов есть общие для всех языков типы, то при создании компонента, предназначенного для общего использования, всякая попытка использовать тип, не транслируемый в общий для всех языков тип, приведет к выдаче соответствующего уведомления. Возможность проверки типов - это лишь одно из достоинств CLS - системы общих спецификаций, которым должен удовлетворять компонент. Так, класс, созданный в C++ в соответствии с требованиями CLS, можно с успехом использовать в качестве базового класса в других языках. Так что единый каркас в многоязыковой среде - это революционное изобретение, уже по самой своей природе способствующее созданию действительно универсальных компонентов.

Структурные и ссылочные типы

Все типы в .NET разделяются на две основные разновидности: структурные типы (value-based или тип-значение) и ссылочные типы (reference-based). Использование типов-значений всегда связано с копированием их значений, а работа со ссылочными типами

всегда осуществляется через адреса их значений
Структурные типы содержат само значение, а не ссылку на него (т.е. улучшает производительность). К структурным типам относятся все числовые типы данных (int, float и пр.), а также перечисления и структуры. Следовательно, структурные типы – это не обязательно примитивные типы. Тип-значение не может наследовать от другого типа и быть наследуемым. Память для структурных типов выделяется из стека. При присвоении одного структурного типа другому присваивается не сам тип (как область в памяти), а его побитовая копия.
Ссылочные типы описывают так называемые объектные ссылки (object references), которые представляют собой адреса объектов. Память для них всегда выделяется в специальной области памяти, называемой кучей (heap) , они инициализируется значением null, при присваивании копируется только адрес, не само значение. Ссылочные типы являются объектом сборки мусора (имеют метод Finalize)

Самоописывающие ссылочные типы

В C++объекты могут храниться как в куче (в динамической памяти), так и в переменных: глобальных (в статической памяти) и локальных (на стеке). Поэтому системы типов в таких языках содержат отдельные типы для самого объекта и для объектной ссылки (указателя на объект).

В среде .NET объекты и объектные ссылки хранятся раздельно, а именно: объекты хранятся в куче, а ссылки - в ячейках. Поэтому общая система типов спроектирована таким образом, что один и тот же ссылочный тип может являться как типом объекта, так и типом объектной ссылки.

Каждый объект в куче содержит информацию о своем типе. Поэтому ссылочные типы, представляющие объекты, называются самоописывающими (self-describing).

Два самоописывающих типа являются встроенными - это System.Object (или просто object в текстовом представлении CIL) и System.String (или string). Тип System.Object является общим базовым классом, от которого непосредственно или транзитивно наследует любой другой класс. Тип System.String используется для представления строковых данных в формате Unicode.

Классы

Основу самоописывающих типов составляютклассы . Классы могут агрегировать значения других типов, а также наследоваться друг от друга (в .NET поддерживается только одиночное наследование). Классы могут содержать следующие элементы:

- Поля (fields).

Поля являются ячейками, в которых хранятся значения других типов.

- Методы (methods).

Методы представляют собой функции классов. Они бывают статическими (static method) и объектными (instance method). Вызываемый объектный метод всегда получает ссылку на объект, для которого он вызывается. Объектные методы делятся на виртуальные и невиртуальные.

- Свойства (properties).

Свойство представляет собой пару методов, один из которых возвращает некоторое

значение, а другой устанавливает это значение.

- События (events). События используются для асинхронного внесения изменений в объект.

Типы-интерфейсы

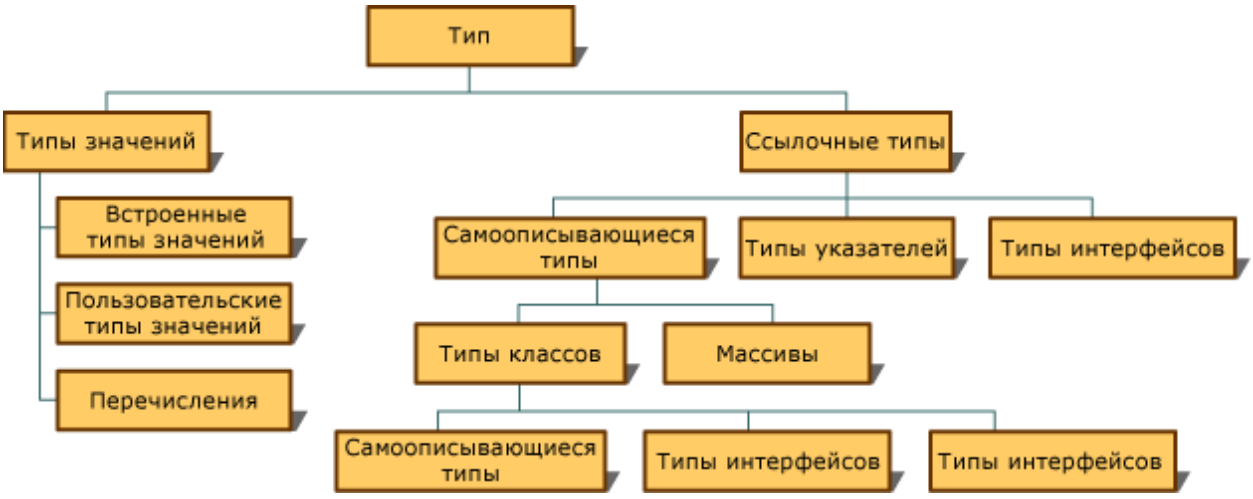
Интерфейсы служат для компенсации отсутствия в .NET множественного наследования. Они могут рассматриваться как чисто абстрактные классы, содержащие только перечисленные ниже элементы:

- Абстрактные методы.
- Статические методы.
- Статические поля.
- Абстрактные свойства.
- Абстрактные события.

Хотя любой класс может наследоваться только от одного базового класса, он может реализовывать произвольное количество интерфейсов. То есть, интерфейс определяет контракт, которому должен удовлетворять любой класс, реализующий этот интерфейс. Нужно отметить, что интерфейс может содержать реализации статических методов, но все остальные методы, включая методы свойств и событий, должны оставаться абстрактными.

Дополнительные элементы системы типов .NET

Из соображений эффективности выполнения программ разработчики платформы .NET добавили в общую систему типов дополнительные элементы, а именно: пользовательские типы-значения (структуры и перечисления) и указатели.



Структуры и перечисления

Как показал опыт платформы Java, которая была разработана задолго до платформы .NET, одной из основных причин ухудшения производительности Java-программ является медленная работа сборщика мусора, вызванная большим количеством мелких объектов в куче. Это явление можно наблюдать в двух случаях:

1. Интенсивное создание временных объектов с очень малым временем жизни. Зачастую такие объекты создаются и используются в теле одного метода.

2. Использование гигантских массивов объектов, при котором возникает ситуация, когда в массиве хранятся ссылки на огромное количество небольших объектов.

Разработчиками .NET был подмечен тот факт, что использование типов-значений вместо объектов позволяет избежать описанных выше проблем, потому что:

1. временные значения хранятся не в куче, а непосредственно в локальных переменных метода;

2. в массивах типов-значений содержатся не ссылки на значения, а непосредственно сами значения.

Поэтому в общую систему типов были добавлены так называемые пользовательские типы-значения. Эти типы могут быть объявлены программистом, но, как и встроенные типы-значения, размещаются не в куче, а в ячейках.

Пользовательские типы-значения делятся на структуры и перечисления.

Структуры являются аналогом классов. Они, как и классы, могут содержать поля, методы, свойства и события. Все структуры неявно наследуют от библиотечного класса System.ValueType, и, более того, встроенные типы-значения также наследуют от этого класса. Тут сразу следует заметить, что система типов не предусматривает никакого наследования структур, кроме данного неявного. Другими словами, структуры не могут наследоваться друг от друга и, тем более, не могут наследоваться от классов (кроме System.ValueType).

Перечисления представляют собой структуры с одним целочисленным полем Value. Кроме того, перечисления содержат набор констант, определяющих возможные значения поля Value. При этом для каждой константы в перечислении хранится ее имя. Перечисления неявно наследуют от библиотечного класса System.Enum, который, в свою очередь, является наследником все того же класса System.ValueType.

Указатели

Использование указателей может значительно увеличить производительность. Однако считается, что применение указателей чревато появлением в программах большого количества трудноуловимых неперехватываемых ошибок. Поэтому, например, система типов уже упоминавшейся платформы Java обходится без указателей.

Тем не менее, разработчикам .NET удалось добавить указатели в общую систему типов. При этом появилось две категории указателей: управляемые указатели (managed pointers) и неуправляемые указатели (unmanaged pointers).

Для того чтобы программы оставались безопасными, на использование управляемых указателей наложен целый ряд ограничений:

1. Управляемые указатели могут содержать только адреса ячеек, то есть они могут указывать исключительно только на глобальные и локальные переменные, параметры методов, поля объектов и ячейки массивов. Для полноты картины

следует заметить, что управляемые указатели могут содержать адрес, непосредственно следующий за последним элементом массива.

2. За каждым указателем закреплен тип ячейки, на которую он может указывать. Другими словами, void-указатели запрещены.

3. Указатели могут храниться только в локальных переменных и параметрах методов.

4. Запрещены указатели на указатели.

На использование неуправляемых указателей никаких ограничений не накладывается, то есть они могут содержать абсолютно любой адрес. Программа, в которой используются неуправляемые указатели, автоматически считается небезопасной и не может пройти верификацию.

Упакованные типы-значения

Наличие в общей системе типов структур, которые во многом напоминают классы, но в действительности классами не являются, в некоторых случаях вызывает некоторые неудобства. Например, в библиотеке классов .NET существуют достаточно удобные контейнерные классы (наиболее часто используется класс ArrayList, представляющий массив с динамически меняющимся размером). Эти классы могут хранить ссылки на любые объекты, но не могут работать с типами-значениями.

Для решения этой проблемы в общей системе типов предусмотрены так называемые упакованные типы-значения. Эти типы являются ссылочными и самоописывающими. Объекты этих типов предназначены для хранения значений типов-значений.

Упакованные типы-значения не могут быть объявлены программистом. Система автоматически определяет такой тип для любого типа-значения.

Получение объекта упакованного типа-значения осуществляется путем упаковки (boxing). Упаковка заключается в том, что в куче создается пустой объект нужного размера, а затем значение копируется внутрь этого объекта. Для упаковки используется следующий синтаксис:

__box(объект типа-значения)

С помощью упаковки мы можем превратить значение любого типа-значения (встроенного примитивного типа, структуры, перечисления) в объект и в дальнейшем работать с этим значением как с настоящим объектом (в том числе, мы можем положить его в ArrayList).

Если же нам требуется произвести обратное действие, мы можем осуществить распаковку (unboxing). Распаковка заключается в том, что мы получаем управляемый указатель на содержимое объекта упакованного типа-значения. Для распаковки используется ключевое слово __ unbox

Встроенные типы данных CTS

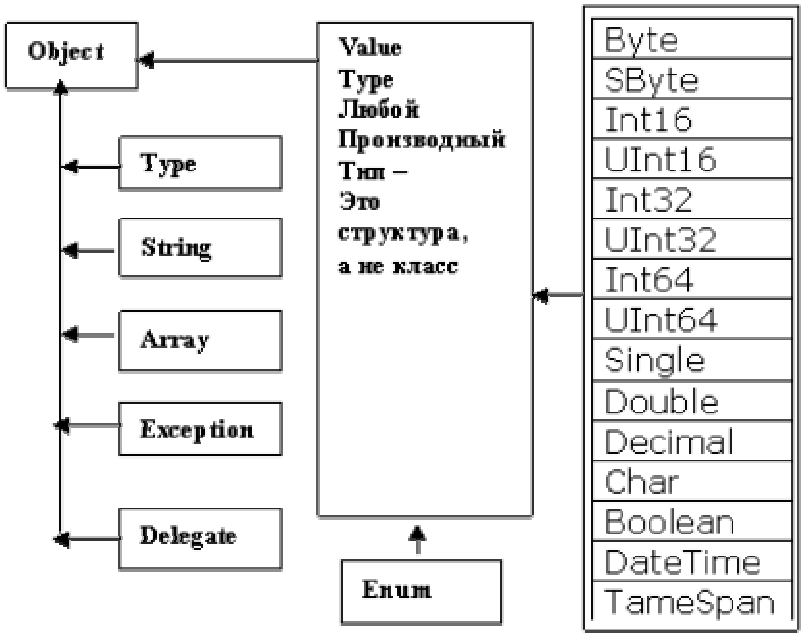
В среде .NET предусмотрен богатый набор встроенных типов данных который един для всех языков программирования Любой тип в CLR — производный от фундаментального

системного типа System::Object. Это обстоятельство отличает CLR от «классической» среды разработки на C++, где типы (базовые, такие как int. long и char) определяют в основном способ размещения в памяти. Типы CLR обладают встроенным механизмом отображения и имеют в своем распоряжении возможности System::Object.
Названия типов данных в языках .NET могут выглядеть по-разному, но эти названия — всего лишь псевдонимы для встроенных системных типов данных

Встроенный тип данных .NET	Название в Visual Basic .NET	Название в C#	Название в Managed C++
System::Byte	Byte	byte	char
System::SByte	Не поддерживается	sbyte	unsigned char
System::Int16	Short	short	short
System::UInt16	Не поддерживается	ushort	unsigned short
System::Int32	Interger	int	int или long
System::UInt32	Не поддерживается	uint	unsigned int
System::Int64	Long	long	_int64
System::UInt64	Не поддерживается	ulong	unsigned _int64
System::Single	Single	float	float
System::Double	Double	double	double
System::Decimal	Decimal	decimal	Decimal
System::Object	Object	object	Object *
System::Boolean	Boolean	bool	bool
System::Char	Char	char	_wchar_t
System::String	String	string	String *

Только часть типов C++ является CLS-совместимой. Если вы создаете свои типы, которые будут использоваться в многоязыковой среде, желательно ограничиться лишь CLS-совместимыми типами. Достаточно следовать правилу не использовать беззнаковые типы в определениях любых открытых членов типов. Этим вы гарантируете, что ваши классы, интерфейсы и структуры смогут нормально работать с любым языком .NET.

Иерархия системных типов



Большинство встроенных типов данных C++ являются производными от типа System::ValueType. Единственное назначение этого типа— замещение некоторых методов, определенных в System::Object, таким образом, чтобы они могли нормально работать со структурными типами. В действительности сигнатуры этих методов в System::Object и System::ValueType совпадают. За счет замещения, например, при сравнении значений двух целочисленных переменных используется структурная,а не ссылочная семантика.
В NET пользовательские размерные типы определяются ключевым словом struct. У размерных типов издержки меньше, чем у ссылочных, потому что они размещаются в стеке, а не в куче
У каждого системного типа данных (к примеру, Int32, Char, Boolean и т. п.) есть схожий набор членов, которые могут оказаться весьма полезными. Речь идет о свойствах MaxValue и MinValue. Первое позволяет получить информацию о максимальном значении, для хранения которого можно использовать данный тип, а второе, соответственно — о минимальном.

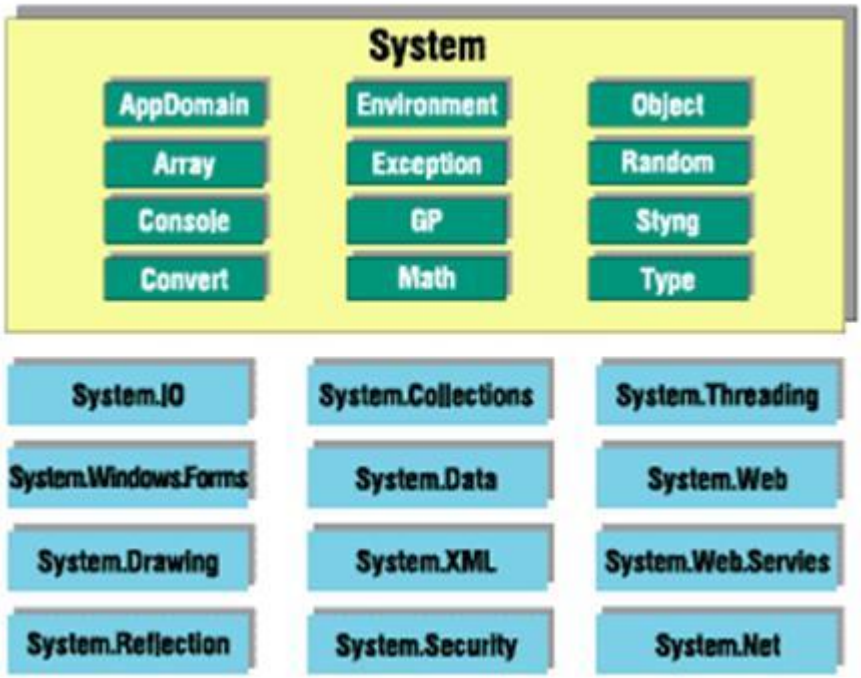
Преобразование типов с помощью класса System::Convert

Класс System.Convert предоставляет полный набор методов для поддерживаемых преобразований. Он обеспечивает не зависящий от языка способ выполнения преобразований и доступен во всех языках, обращающихся к общеязыковой среде выполнения. В то время как различные языки программирования могут иметь различающиеся методы преобразования типов данных, класс Convert гарантирует, что все обычные преобразования доступны в общем формате. Этот класс выполняет понижающие преобразования, а также преобразования несвязанных типов данных. Например, поддерживаются преобразования из типов String в числовые типы, из типов DateTime в типы String и из типов String в типы Boolean. Класс Convert выполняет преобразования с проверкой и всегда создает исключение, если данное преобразование не поддерживается. Значение, которое требуется преобразовать, можно передать в один из соответствующих методов класса Convert и инициализировать возвращаемое значение в новую переменную. Например, в следующем коде класс Convert используется дл я преобразования значения String в значение Boolean

```
String *MyString = S"true";
bool MyBool = Convert::ToBoolean(MyString);
```
Класс Convert полезен также в том случае, если имеется строка, которую нужно преобразовать в числовое значение и наоборот
```
String *newString = S"1234";
int MyInt = Convert::ToInt32(newString);
double d=3.14;
String *dString= Convert::ToString(d);
```

Библиотека базовых классов

На основе библиотеки базовых классов строятся все .NET-приложения. Принципиальная новизна заключается в том, что если ранее подобный набор создавался для каждого языка программирования, то теперь он — один для всех средств. Дополнительный стимул для использования единого набора функций — возможность улучшения управления оперативной памятью. Как известно, огромное число проблем надежности программ связано с использованием неодинаковых механизмов динамического распределения пространства в разных языках.
В .NET базовые функции перестали быть принадлежностью пользовательских приложений и превратились в неотъемлемый компонент операционной системы.
Все классы, реализованные в .NET Framework Class Library организованы в виде пространств имен (namespaces). **Пространство имен** — это логическая структура для организации имен, используемых в приложении .NET. Основное назначение пространств имен — предупредить возможные конфликты между именами в разных сборках. Каждое пространство имен содержит классы и другие типы, которые относятся к специфическим задачам или группе задач
Список наиболее важных классов и пространств имен в библиотеке классов .NET.



Пространство имен System

Пространство имен System является корневым пространством имен в Microsoft .NET Class Library и содержит фундаментальные типы данных, реализованные в .NET Framework. Это пространство имен содержит класс Object, который служит предком для всех классов в библиотеке классов .NET

Класс System::Object

Все типы данных (как структурные, так и ссылочные) производятся от единого общего предка: класса System.Object. Класс System.Object определяет общее полиморфическое поведение для всех типов данных .NET

Главные методы объекта System::Object

1. Метод Equals (Object) или Equals (Object, Object) — служит для проверки, являются ли два объекта одним и тем же экземпляром. Для данных со значениями этот метод переопределен (в классе ValueType) и позволяет проверить идентичность хранимых экземплярами объекта значений. Обратите внимание: когда мы сравниваем два ссылочных типа, метод Equals проверяет, являются ли оба экземпляра идентичными; для данных со значениями метод Equals также проверяет идентичность данных, хранимых экземплярами объекта.

2. Метод ReferenceEquals (Object, Object) проверяет, являются ли два объекта одним и тем же экземпляром класса.

3. Метод Finalize() по умолчанию не выполняет никаких действий. Этот метод может быть переопределен в унаследованных классах для выполнения финальной очистки памяти перед тем, как механизм сборки мусора (Garbage Collector) уничтожит данный объект.

4. Метод GetHashCode() служит для генерации хэш-значений (типа Integer), которые могут быть использованы для хранения объектов в хэш-таблицах.

5. Метод MemberwiseClone() создает точную копию объекта.

6. Метод ToString() возвращает текстовое представление объекта. В большинстве случаев этот метод возвращает полное имя класса для данного объекта.

7. Метод GetType() возвращает объект типа Type для данного экземпляра класса.

Расширения управляемого C++ (Managed C++)

1. Управляемые типы данных
2. Класс управляемого C++
3. Свойства в .NET
4. Делегаты
5. Реализация обработки событий в .NET
6. Управляемые массивы.
7. Класс System::Array
8. Управляемые двумерные массивы

В CLR-среде определены два типа управляемых элементов: управляемый код (managed code) и управляемые данные (managed data).
CLR-среда управляет данными приложения, т. е. их структурой, и ссылками на объекты в рамках приложения, освобождая ссылки, когда потребность в них пропадает. Программы используют реализованное в CLR автоматическое управление памятью, а именно - сборщик мусора (garbage collection). Работа сборщика мусора заключается в освобождении памяти, занятой ненужными объектами. При этом сборщик мусора также умеет "двигать" объекты в памяти, тем самым устраняя фрагментацию адресного пространства. Все это, возможно благодаря тому, что во время выполнения программы известны типы всех используемых в ней объектов. Другими словами, данные, с которыми работает программа, находятся под полным контролем среды выполнения и называются, соответственно, управляемыми данными (managed data).

Управляемый код тесно работает с CLR-средой — он обеспечивает CLR необходимыми метаданными, чтобы та могла предоставлять свои сервисы управления памятью, межъязыковой совместимости, безопасности доступа и автоматического управления жизнью объектов.Создавать на C++ управляемый код совместимый с .NET код довольно просто: достаточно разместить несколько новых расширений языка ключевых слов и символов в «нужных» местах.

Управляемые типы данных

CLR поддерживает автоматическую сборку мусора . Это избавлят программиста от необходимости помнить об освобождение памяти, занимаемой объектами. Объявление управляемого типа в MC++ производится с помощью ключевых слов **__gc** . Идентификатор __gc применяется для объявления сложных типов, массивов и указателей, размещаемых в куче среды исполнения CLR. Сокращение **gc,** скорее всего, происходит от garbage collection. Ключевое слово __gc перед объявлением класса говорит компилятору, что наш класс является управляемым и подчиняется всем правилам среды CLR. В частности, нам не нужно вызывать деструктор для удаления объекта из памяти, эту работу за нас сделает CLR

__gc class MyClass {

public:

~MyClass();

MyClass (); };

Класс управляемого C++

Класс в .NET Framework аналогичен классу в C++: представляет собой совокупность кода и данных, но в отличии от C++ наследовать он может только от одного базового класса. В среде .NET классы могут содержать: кроме • полей (fields) и • методов (methods), так же• свойства (properties) и события (events). Сбытия определяют уведомления, которые способен распознавать класс. Сойства предоставляют как и поля доступ к данным, но реализуются с применением методов получения и задания значения свойства аксессоров (get и set). Эти методы автоматически вызываются компилятором при занесении извлечении значений полей с помощью оператора присваивания Свойства компонент доступны инспектору свойств на этапе проектирования

Свойства в .NET

Свойства являются атрибутами компонента, определяющими его внешний вид и поведение. . Данные и методы доступны только во время выполнения программы. Установка значений данным во время выполнения программы требует написания соответствующих текстов кода для обработки событий. Свойства компонентов можно изменять на стадии проектирования приложения при помощи окна свойств. При помощи свойств может быть реализована собственная логика доступа к полям класса. Например, методы доступа могут применяться для согласованного отображения значений полей во всех элементах управления, где они используются. Другим примером применения свойств может служить проверка правильности значений вводимых в поля данных.

В классах управляемого C++ определение свойств начинаются с ключевого слова __property. Для свойств доступных для чтения и записи должны быть определены два метода get и set . После знака подчеркивания указывается название свойства, которое

будет отображаться в окне свойств. Имя закрытого поля класса, которое содержит значение свойства не должно совпадать с названием свойства

```
__gc class MyTime

{

double m_Time;

public:

__property void set_Time(double t){m_Time=t;}
__property double get_Time(){return m_Time; }

};
```

Делегаты

Делегат представляет собой оболочку функции обратного вызова, обеспечивающую контроль типов. Делегат является обобщением указателя на функцию языка Си. Обычно делегаты служат для определения сигнатур методов обратного вызова, которые осуществляют реакцию на событие. Реагирующие на событие приложения передают объекту ссылку на метод, который следует вызвать при наступлении этого события

Реализация обработки событий в .NET

Обработчик события реализуется в NET, как правило, в виде делегата. Обработчики событий не могут возвращать ничего, кроме void. В них отсутствует точка, которая могла бы служить для возврата значения. Обработчики должны принимать два параметра. Первый параметр является ссылкой на объект, который сгенерировал событие. Второй параметр должен быть ссылкой либо на базовый класс .NET System.EventArgs, либо на производный класс. Класс EventArgs представляет собой общий базовый класс для всех уведомлений о произошедших событиях. В окне свойств каждого элемента управления на вкладке событий перечислены все доступные события для этого элемента

Пример обработчики события нажатия кнопки

```
private: System::Void button1_Click(System::Object * sender, System::EventArgs * e)
{
}
```

Управляемые массивы.

В отличии от массивов C++ , которые являются просто указателями, управляемые массивы в среде NET – это самоопределённые объекты, реализованными в динамической памяти. Для определения неуправляемых массивов используется синтаксис C++. Для определения управляемого массива необходимо после его имени указать _gc[] , либо в качестве элементов массива использовать управляемые типы. При определении управляемого массива размерность не пишется. Размерность указывается в операторе new. Все управляемые массивы определяются в динамической памяти

Пример определения неуправляемых массивов

```
int  UnMan[5];
int *pUnMan = new int[5];
```

Пример определения управляемых массивов

1-случай:

```
int Man __gc[] = new int __gc[5]
```

2-случай:

```
Int String * sMan[] = new String *[5]
```

Класс System::Array

Все управляемые массивы являются потомками класса System::Array. Класс System::Array содержит свойства и методы для работы с массивами. Элемент массива может быть любого типа, так как в .NET все типы — это объекты, базирующиеся на классе System.Object. Массив хранит элементы типа System::Object. Для того чтобы найти тип, которым был объявлен массив, мы используем метод GetType. Ниже приведены базовые свойства и методы массивов

Свойство	Описание
IsFixedSize	Позволяет узнать, является ли массив массивом фиксированного размера
IsReadOnly	Позволяет узнать, является ли массив массивом только для чтения
IsSynchronized	Позволяет узнать, является ли доступ к массиву синхронизированным
Length	Возвращает общее число элементов во всех размерностях массива
Rank	Возвращает число размерностей массива
SyncRoot	Возвращает объект, используемый для синхронизации доступа к массиву

Метод	Описание
BinarySearch	Выполняет поиск в одномерном массиве, используя алгоритм бинарного поиска
Clear	Обнуляет обнуляет значения указанного диапазона элементов (для типов со значениями) или присваивает им значение Null (для ссылочных типов).
Copy	Копирует указанные элементы массива в другой массив и при необходимости выполняет преобразование типов

IndexOf	Возвращает первый индекс указанного элемента в одномерном массиве
LastIndexOf	Возвращает последний индекс указанного элемента в одномерном массиве
Reverse	Изменяет порядок следования элементов одномерного массива на противоположный
Sort	Сортирует элементы одномерного массива
CopyTo	Копирует все элементы одномерного массива в указанный одномерный массив, начиная с указанного индекса в принимающем массиве
Clone	Создает точную копию массива
GetEnumerator	Возвращает интерфейс IEnumerator для данного массива
GetLength	Возвращает число элементов в указанной размерности массива
GetLowerBound	Возвращает нижний индекс массива
GetUpperBound	Возвращает верхний индекс массива
GetValue	Возвращает значения указанных элементов массива
SetValue	Устанавливает значения указанных элементов массива

Управляемые двумерные массивы

При работе с неуправляемым массивом используется старый синтаксис доступа к элементам массива [ ] [ ], тогда как при работе с управляемым массивом, который является истинно двумерным, используется синтаксис [,]. При использовании синтаксиса [ ] [ ] каждый из подмассивов может иметь разные размеры (т.н. массив с неровным правым краем). Синтаксис [,] предполагает использование истинно прямоугольного массива.

```
void main(void)
{ // ("Использование прямоугольного массива [,] " ) ;
int rect2DArray [,] = new int __gc [3,43]; // сборщик мусора управляемый
for(int row=0; row< rect2DArray ->GetLength(0); row++)
// по строкам
for(int col=0; col< rect2DArray->GetLength(1); col++)
//по столбцам
rect2DArray [row,col] = row*10 + col;
```

Работа со строками в Windows

1. Использование кодировки UNICODE
    1. Представление символов в UNICODE
    2. Разработка универсальных программ для работы со строками
    3. Перекодирование однобайтовых символов в Unicode и обратно с учетом кодовой страницы
    4. Пример перекодирования символов Unicode в однобайтовые
2. Класс System::String
    1. Инициализация строк при помощи строковых констант
    2. Массивы строк
    3. Сравнения объектов класса String
    4. Методы класса String
3. Строковое представление данных
4. Форматирование строк
    1. Строки стандартных числовых форматов
    2. Составное форматирование
    3. Синтаксис элементов форматирования
    4. Разбор строк

Использование кодировки UNICODE

Представление символов в UNICODE

Представление символов национальных алфавитов в однобайтовых переменных типа char привело к появлению большого числа кодовых страниц и к большим неудобствам при использовании программ, содержащих строки с символами из различных национальных алфавитов. Для решения проблемы была разработана кодировка UNICODE, в которой для хранения номеров символов используется от одного до четырех байтов. В стандарт языка C99 введен специальный тип данных для хранения многобайтовых символов wchar_t. Этот тип данных занимает в Windows два байта,а в Unix - четыре. В двухбайтовой кодировке UNICODE символы с номерами
0x0000 - 0x007f отведены для хранения ASCII кодов,
0x0080 - 0x00ff для хранения символов Latin1,
0x0400 - 0x04ff для хранения Кириллицы.
Для работы со строками , состоящими из UNICODE символов, введены функции,

аналогичные тем, что использовались при работе со строками однобайтовых символов, но в их именах первые символы str заменяются на wcs (от wide character set). Например, для определения длины строки используется функция size_t wcslen(const wchar_t *string);. Все заголовки этих функций объявлены в заголовочном файле string.h.

Разработка универсальных программ для работы со строками

Фирма MICROSOFT разработала ряд макросов, позволяющих писать программы, в которых строки могут интерпретироваться компилятором либо как строки UNICODE, либо как строки из однобайтовых символов, в зависимости от того, определен ли макрос _UNICODE в вашей программе или нет.

Вот пример текста программы с использованием этой директивы (из заголовочного файла tchar.h):

```
#ifdef _UNICODE
typedef wchar_t TCHAR;
#else
typedef char TCHAR;
#endif
```

Если в вашей программе определен макрос _UNICODE, тогда будет вставлена строка кода typedef wchar_t TCHAR; ,

а если макрос _UNICODE, не определен в программу вставляется строка typedef char TCHAR; .

Таким образом, если у Вас определен макрос _UNICODE в программе, макрос TCHAR будет заменен на wchar_t, а в противном случае на char.

Вот еще несколько полезных макросов из этого заголовочного файла tchar.h:

```
#ifdef  _UNICODE
#define  __T(x)       L##x
#else
#define __T(x)        x
#endif
```

Два символа ## - это операция сшивки лексем. Если в вашей программе определен макрос _UNICODE и имеется строка

TCHAR a[255]=__T("my string"); тогда эта строка будет преобразована препроцессором в строку

wchar_t a[255]=L"my string".

А если вы не определили макрос _UNICODE, то в вашей программе будет вставлена строка

char a[255]="my string".

Для макроса __T(x) в заголовочном файле tchar.h определены два синонима

```
#define _T(x)   __T(x)
#define _TEXT(x)   __T(x)
```

Для работы со строками определены макросы PTSTR и PCTSTR.

```
#ifdef _UNICODE
typedef wchar_t * PTSTR;
typedef const wchar_t * PCTSTR;
#else
typedef char *  PTSTR;
typedef const char * PCTSTR;
#endif .
```

В заголовочном файле tchar.h определены универсальные макросы для всех функций ,

работающих со строками. Вот небольшой фрагмент этого файла:

```
#ifdef _UNICODE
#define _tmain      wmain
#define _tprintf    wprintf
#define _tfopen  _wfopen
#define _tstoi      _wtoi
#define _tcscat     wcscat
#define _tcslen     wcslen
#else
#define _tmain      main
#define _tWinMain   WinMain
#define _tprintf    printf
#define _tfopen     fopen
#define _tstoi      atoi
#define _tcscat     strcat
#define _tcslen     strlen
#endif
```

Использование в программе

```
#define _UNICODE
#include <stdio.h>
#include<tchar.h>
void _tmain(int argc,TCHAR *argv[]) {
_tprintf(_T("Hello Wold\n"));
}
```

Перекодирование однобайтовых символов в Unicode и обратно с учетом кодовой страницы

В Windows имеется функция WideCharToMultiByte для преобразования текстов из кодировки UNICODE в однобайтовую кодировку с заданной кодовой страницей
Для обратного преобразования можно использовать функцию MultiByteToWideChar. У функции MultiByteToWideChar пять параметров. Первый - это номер кодовой страницы для исходной строки. Страницу можно задать как непосредственно, так и константой типа CP_ACP (установленная в системе кодовая страница ANSI), CP_OEMCP (установленная в системе страница OEM), CP_THREAD_ACP (страница текущего потока). Второй параметр - набор флагов, определяющий способ обработки диакритик (надстрочных и подстрочных знаков) и неправильных символов. Флаг MB_PRECOMPOSED означает, что базовый символ и диакритика объединены, несовместимый с ним флаг MB_COMPOSITE - что диакритики записаны как отдельные символы. Флаг MB_ERR_INVALID_CHARS задает проверку корректности символов в исходной строке. Оставшиеся параметры определяют адрес и длину входной и выходной строк.
Функция WideCharToMultiByte дополнительно к перечисленным принимает еще два параметра, которые нужны, если применяется обработка неправильных символов.

Пример перекодирования символов Unicode в однобайтовые

```
#define UNICODE
#include <windows.h>
char ch;
wchar_t wch;
WideCharToMultiByte(CP_ACP, 0, &wch, 1, &ch, 1, NULL, NULL);
```

Класс System::String

Строка представляет собой последовательный набор знаков Юникода, обычно используемых для представления текста. Каждый знак Юникода в строке определен скалярной величиной Юникода, которая кодируется с помощью кодировки UTF-16. Числовое значение каждого элемента кодировки представлено с помощью System::Char. System::String представляет собой последовательный набор объектов System::Char, образующих строку. Для представления одного символа может потребоваться более одного элемента кодировки. Номер позиции Char в String в строке может не совпадать с порядковым номером соответствующего знака Юникода, так как знак Юникода может быть закодирован несколькими Char.
Свойство Length служит для извлечения количества объектов Char в строке, а свойство Chars — для доступа к самим объектам Char в строке. Свойство Chars позволяет получить данные только для чтения.
Тип String представляет собой конечный (sealed) класс; это означает, что он не может быть базовым для другого класса.
Объект String— неизменяемый, т.е. будучи однажды инициализированным, он не может быть изменен. Класс String содержит методы, которые можно использовать для изменения объекта String (такие, как Insert (Вставка), Replace (Замена) и PadLeft). Однако, в действительности, указанные возвращают новый объект String, содержащий измененный текст. Чтобы получить возможность изменять исходные данные, необходимо использовать класс System::Text::StringBuilder .

Инициализация строк при помощи строковых констант
Строковые константы, могут быть определенные с префиксом и без него. Строковая константа , определенная с использованием только кавычек, является указателем на char (символ), т.е. указателем на последовательность символов ASCII, заканчивающуюся нулем. Такой указатель не является указателем на объект String. А строковая константа, определенная с префиксом S, является указателем на управляемый объект String. Префикс L обозначает строку символов Unicode, которая также не является объектом String.
Следующий фрагмент демонстрирует эти три типа строк:

```
char *psl = "ASCII string l i t e r a l " ; // неуправляемый
// символ *psl = "строковый литерал ASCII ";
wchar_t *ps2 = L"Unicode string l i t e r a l " ; // неуправляемый
// L " строковый литерал Уникода ";
String *ps3 = S"String object l i t e r a l " ; // управляемый
// Строка *ps3 = S " строковый литерал - объект String ";
```

Управляемые строковые константы String и неуправляемые строковые константы ASCII и Unicode (благодаря автоматической упаковке) можно использовать в выражениях, в которых ожидается использование управляемого строкового объекта String Однако управляемый строковый объект String нельзя использовать там, где ожидается появление переменных неуправляемых типов.

Массивы строк

```
String* S[]=new String* [3];
S[0]=S”111”;          S[1]=S”2222”;          S[2]=S”333333”;
```

Сравнения объектов класса String
Члены String выполняют порядковые или лингвистические операции над String. Порядковые операции выполняются над числовыми значениями каждого Char. Лингвистические операции выполняются над значениями String с учетом правил учета

регистров, сортировки, форматирования и синтаксического разбора в соответствующей культуре. Лингвистические операции выполняются в контексте культуры — определенной явно либо текущей (см. CultureInfo.CurrentCulture).
Для сравнения объектов на равенство можно использовать оператор == или метод Equals (Равняется). Метод Equals (Равняется) проверяет равенство содержимого объектов, тогда как оператор == проверяет лишь равенство указателей (т.е. равенство адресов объектов в памяти). Для сравнения с пустой строкой необходимо использовать статический член класса String::Empty. Например,
if (textBox1->Text->Equals(String::Empty))
Правила сортировки определяют алфавитный порядок знаков Юникода и принципы сравнения двух строк. Например, метод Compare выполняет лингвистическое сравнение, в то время как метод CompareOrdinal осуществляет порядковое сравнение.
.NET Framework поддерживает правила сортировки по словам, строкам и порядковым номерам. Сортировка по словам выполняет сравнение строк с учетом культуры, при котором некоторые знаки Юникода, отличные от букв и цифр, могут иметь специально присвоенные им весовые коэффициенты. Сортировка по строкам аналогична сортировке по словам, за исключением того, что особых случаев нет и все буквы и цифры следуют после всех остальных знаков Юникода. При сортировке по порядковому номеру строки сравниваются на основе числовых значений каждого Char в строке. Дополнительные сведения о сортировке по словам, строкам и порядковым номерам см. в разделе System.Globalization.CompareOptions.

Методы класса String

Для получения индекса подстроки или знака Юникода в строке используются методы
IndexOf - возвращает индекс первой обнаруженной String или одного или нескольких знаков в пределах данного экземпляра ,
IndexOfAny Возвращает индекс первого обнаруженного в данном экземпляре знака из указанного массива знаков Юникода
, LastIndexOf - возвращает индекс последнего вхождения указанного знака Юникода или Stringв пределах данного экземпляра,
LastIndexOfAny - возвращает индекс последнего вхождения в данном экземпляре какого-либо одного или нескольких знаков, указанных в массиве знаков Юникод.

Методы Copy и CopyTo применяются для копирования строки или подстроки в другую строку или в массив Char.
Copy - создает новый экземпляр String, имеющий то же значение, что и указанная String
CopyTo - копирует заданное число знаков, начиная с указанной позиции в этом экземпляре, до указанной позиции в массиве знаков Юникода

Substring Split применяются для создания одной или нескольких новых строк из частей исходной строки.
Substring - извлекает подстроку из данного экземпляра.
Split - определяет подстроки в данном экземпляре, ограниченные одним или несколькими знаками, указанными в массиве, затем помещает подстроки в массив String
ToCharArray - копирует знаки данного экземпляра в массив знаков Юникода

Методы Concat и Join используются для создания новой строки из одной или нескольких подстрок.
Concat - объединяет один или несколько экземпляров строки String или представления String значений одного или нескольких экземпляров Object.
Join - вставляет заданные разделители типа String между элементами заданного массива

String, создавая одну сцепленную строку.

Для модификации всей строки или ее части служат методы:
Insert - вставляет указанный экземпляр String по заданному индексу в данном экземпляре ,
Replace - заменяет все вхождения указанного знака Юникода или String в данном экземпляре другим заданным знаком Юникода или String,
Remove - удаляет заданное число знаков из данного экземпляра, начиная с указанной позиции,
а также PadLeft, PadRight, Trim, TrimEnd и TrimStart.
Для изменения регистра знака Юникода в строке используются ToLower и ToUpper.
Format служит для замены одного или нескольких заполнителей в строке строковым представлением одного или нескольких значений

Строковое представление данных
Средства .NET позволяют представлять числовые значения, перечисления, время и даты в виде строк. В .NET Framework существует настраиваемый механизм преобразования значения в строку, подходящую для отображения. Например, числовое значение может быть представлено в шестнадцатеричном виде, научном представлении или в виде последовательности цифр, разделенных на группы знаком препинания. Время и даты можно форматировать в соответствии со стандартом страны, региона или культуры. Перечисляемая константа может быть отформатирована как числовое значение или его имя.
Метод ToString наиболее часто используется для вывода информации о типе. Метод Object::: ToString возвращает полное имя класса данного объекта. Метод ToString обеспечивает для любого управляемого типа данных строковое представление в виде объекта String . Метод ToString перегружен для числовые типы данных, таких, как int или float и может использоваться для перевода числа в строку в соответствии с заданным форматом. В приведенных ниже примерах используются параметры форматирования по умолчанию.
double d=3.14;
String* s=d.ToString();
Класс String содержит набор методов преобразования строкового представления данных в их реальные типы с учетом параметров культуры.
String* s=S”134”;
int i=s->ToInt32(0);
В свою очередь, многие типы данных, включая и примитивные, содержат метод Perser, который позволяет по строковому представлению создать объект данного типа.
String *s=S”123”;
int i=int::Parse(s);

Форматирование строк

Форматирование задается строкой **указателя формата**, определяющего представление значения базового типа. Например, указатели формата могут задавать научное представление числа или указывать, должен ли месяц быть представлен в формате даты числом или именем. Для форматирования чисел, дат, времени и перечислений определены стандартные и **настраиваемые спецификаторы формата**. Указатели формата используются различными методами, форматирующими выходные строки, например **String::Format** и **ToString**, и методами базовых типов, разбирающими входные строки, такими, как **Parse**.
Параметры культуры также могут влиять на представление базовых типов. Можно

задать пользовательские параметры культурной среды или использовать параметры по умолчанию, связанные с текущим потоком. Например, при форматировании переменной типа currency (значок валюты), разделитель групп и разделитель дробной и целой частей определяются параметрами культуры.
Управление форматированием сводится к заданию строки формата и выбору поставщика форматирования или использованию значений по умолчанию. Строка формата содержит один или несколько указателей формата, задающих способ преобразования значения. Дополнительные параметры форматирования и сведения о культуре, необходимые для форматирования, передаются **поставщиком форматирования**. Поставщики формата определяют знаки, применяемые при форматировании, но не сами указатели.
Составное форматирование позволяет совместить в выходной строке несколько отформатированных значений, используя несколько строк форматирования. Выходную строку можно обрабатывать, отображать в консоли или выводить в поток .
Форматирование служит для преобразования стандартных типов данных.NET Framework в строку и последующего отображения. Чтобы отформатировать базовый тип, при вызове метода **ToString** для требуемого вида следует задать указатель формата, поставщик формата или оба параметра. Если указатель формата не задан или в качестве параметра передано значение **null** , по умолчанию будет использован указатель «G» (общий формат). Если поставщик формата не задан, или в качестве параметра передано значение **null** , или если для поставщика не заданы требуемые для выполнения форматирования свойства, будет использован поставщик формата, связанный с текущим потоком.

Строки стандартных числовых форматов служат для форматирования стандартных числовых типов. Стандартная строка числового формата имеет вид *Axx*, где *A* — это буква-указатель формата (указатель формата), а *xx* — необязательный параметр (целое число — указатель точности). Указателем формата должен быть один из встроенных знаков формата. Указателем точности может изменяться от 0 до 99, с его помощью задается число значащих цифр дробной части. Строка формата не должна содержать пробелов.

Указатель формата	**Имя**
C или c	Валютные
D или d	Десятичные
E или e	Научные (экспоненциальные)
F или f	Фиксированная запятая
G или g	Общий

N или n	Число
P или p	Проценты

X или x	Шестнадцатеричные

Пример.

```
double MyDouble = 123456789;
System::Threading::Thread::CurrentThread->CurrentCulture = new
System::Globalization::CultureInfo("ru-RU");
String*s=MyDouble.ToString("C");
//получим 123456789,00р.
```

Платформа .NET Framework позволяет определять пользовательские схемы форматирования и параметры культуры. Таким образом можно расширить схемы форматирования существующих для использования в пользовательских сценариях или создать пользовательские схемы форматирования для пользовательских типов.

В приведенной ниже таблице представлены знаки, используемые для создания **настраиваемых строк числовых форматов**

Знак формата	**Имя**
0	Знак-заместитель нуля
#	Заместитель цифры
.	Разделитель
,	Разделитель тысяч
%	Заместитель процентов
E0 E+0 E-0 e0 e+0 e-0	Научная нотация
\	Escape-знак
'ABC' "ABC"	Строка букв
;	Разделитель секций

Пример.

```
double a=1234567890;
String *s =a.ToString(“(###)### ####”);
// получим (123)456 7890
```

Составное форматирование позволяет форматировать строку фиксированного текста с внедренными в нее элементами форматирования. Выходная строка будет состоять из фиксированного текста, а на месте значений будут расположены их отформатированные представления. Составное форматирование применяется с помощью метода String::Format, возвращающего отформатированную строку, метода Console::WriteLine, выдающего выходную строку в консоль, и с помощью реализации интерфейса TextWriter::WriteLine, выдающего выходную строку в поток или файл.
Каждый элемент формата или замещающий символ связан с одним из элементов списка значений. В результате составного форматирования получается строка, в которой каждый элемент форматирования исходной строки заменен соответствующим отформатированным значением.
Исходная строка состоит из произвольного числа фрагментов фиксированного текста и одного или нескольких элементов форматирования. Содержание фиксированного текста произвольно.

Синтаксис элементов форматирования
Элементы форматирования должны быть заданы в следующем виде.
{*index*[,*alignment*][:*formatString*]}
Компонент *индекс* (спецификатор параметра) является обязательным. Индекс представляет собой число больше нуля, определяющее соответствующий элемент списка значений. Иными словами элемент форматирования с индексом 0 служит для форматирования первого по списку значения, элемент форматирования с индексом 1 служит для форматирования второго значения и т. д..
Например, String::Format("{0:X}, {1:E} ,{2:N}",param1,param2,param3).
Любой элемент форматирования может ссылаться на любой параметр.

```
String* myName = S"Fred";
String* myd=String::Format("Name = {0}, hours = {1:hh}, minutes = {1:mm}",myName,
__box(DateTime::Now));
```

Необязательный компонент ***alignment*** задается целым числом со знаком и определяет требуемую ширину отформатированного поля. . Данные в строке выравниваются по правому полю, если значение компонента ***alignment*** положительное, по левому полю — если отрицательное. При необходимости отформатированная строка дополняется пробелами.

```
    int myInt=33;
String* FormatPrice = String::Format("Price ={0,-10:N}", __box(myInt));
```

Разбор строк
Метод **Parse** преобразует строку, предоставляющую базовый тип .NET Framework, в реальный базовый тип .NET Framework. Поскольку разбор — это операция, обратная форматированию (преобразования базового типа в строковое представление), к нему применимы многие из правил и соглашений, используемых для форматирования.
Для всех числовых типов имеется статический метод **Parse**, который можно использовать для преобразования строкового представления числового типа в реальный числовой тип. Эти методы позволяют выполнять разбор строк, полученных с помощью указателей формата.

Знаки, используемые для представления символов денежной единицы, разделителей групп разрядов и разделителей целой и дробной частей числа, описываются в поставщиках формата. Метод **Parse** допускает применение поставщика формата, что позволяет задавать и выполнять явный разбор строк, связанных с культурной средой. Если поставщик формата не указан, используется поставщик, связанный с текущим потоком.

```
Boolean val;

String* input = Boolean::TrueString;

val = Boolean::Parse( input );

String*output=String::Format( "{0} parsed as {1}", input, __box(val) );
```

ГЛАВА 6. ПРИНЦИПЫ РАЗРАБОТКИ WINDOWS ПРИЛОЖЕНИЙ В .NET (WINDOWS FORMS APPLICATION)

1. Визуальная разработка приложений.
    1. Формы
    2. Компонентная модель.
    3. Конструирование приложения.
    4. Автоматическая генерация кода.
    5. Механизм двунаправленной разработки.
2. Создание проекта при помощи шаблона Windows Forms Application
3. Генерация кода
4. Режимы дизайна и кода

Визуальная разработка приложений.

Создание Windows-приложения заключается в расположении компонентов на форме, изменении их свойств, написании кода для обработки возникающих событий и написании кода, определяющего логику самого приложения.

Формы

Форма - это прямоугольное окно на экране. Форма — это каркас, на котором проектируется интерфейс программы. Форма — это экранный объект, который содержит элементы управления и обеспечивает функциональность программы.

Форма в .NET реализуется классом Form из глубоко вложенного пространства имен System.Windows.Forms. Для главного окна приложения Visual Studio .NET строит по умолчанию класс Form1, который является наследником класса Form и автоматически наследует его функциональность - свойства, методы, события.

public __gc class Form1 : public System::Windows::Forms::Form

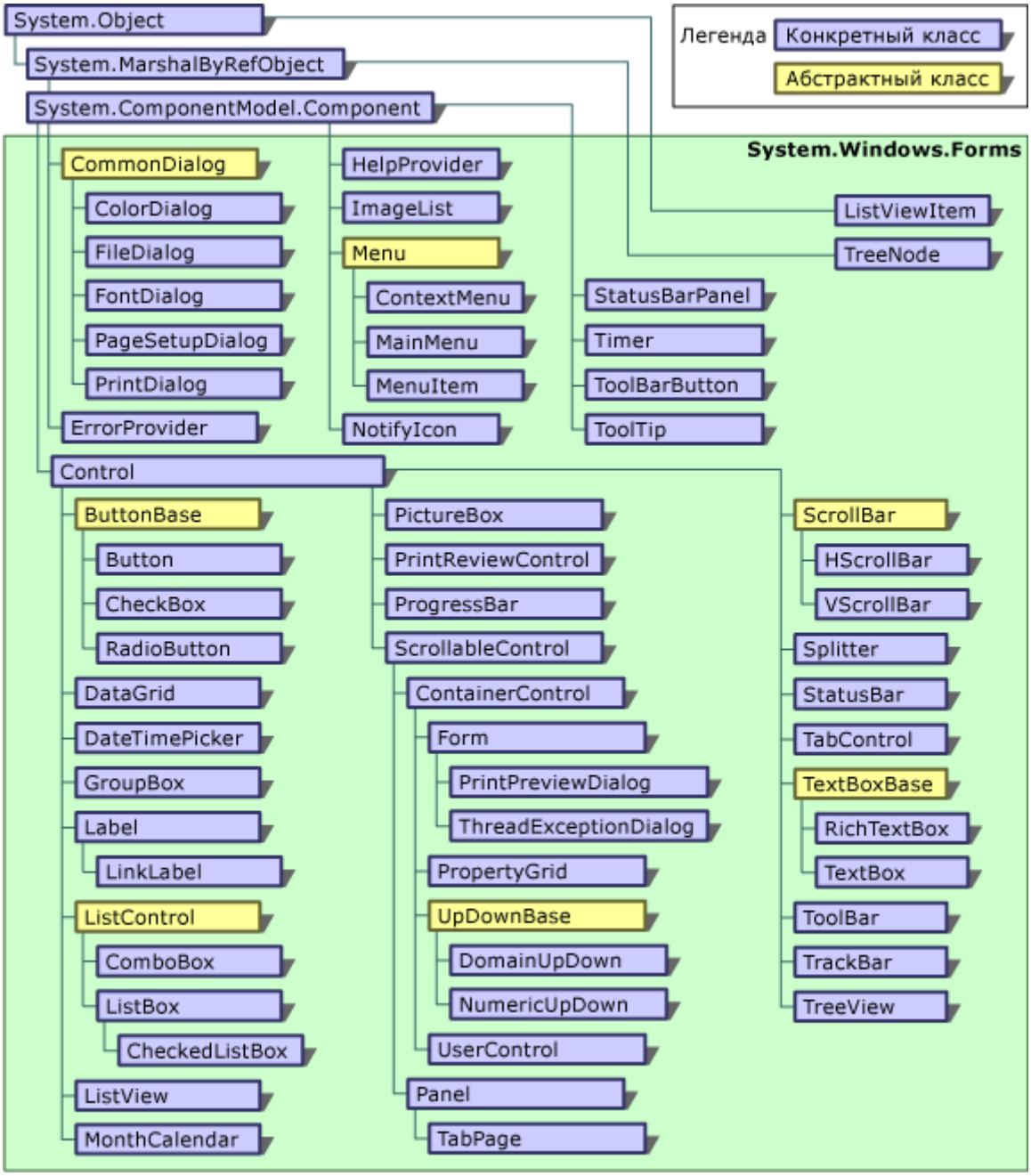
Visual Studio .NET унифицирует процедуру создания форм для Web и Windows. Кроме того, в Visual Studio .NET реализован механизм визуального наследования Windows-форм, что упрощает повторное использование кода

Компонентная модель.

Использование готовых компонентов - ключевая технология программирования на протяжении всей истории этого вида деятельности. Понятие компоненты выходит за рамки понятий интерфейсного элемента и класса. Главное отличие заключается в полной функциональности компонентов на стадии проектирования. Находясь в интегрированной среде, компоненты можно использовать непосредственно, менять их свойства, облик и

поведение или порождать производные элементы, обладающие нужными характеристиками.

Следующая иллюстрация показывает иерархию компонентов и элементов управления в пространстве имен System.Windows.Forms.



Конструирование приложения.

Конструирование приложения осуществляется по способу "drag and drop", и заключается в перетаскивании захваченных мышью визуальных компонентов из окна Toolbox на форму приложения. Форма представляет собой окно приложения с управляющими компонентами, которые переносятся на стадии проектирования или создаются динамически в процессе работы программы.

Автоматическая генерация кода.

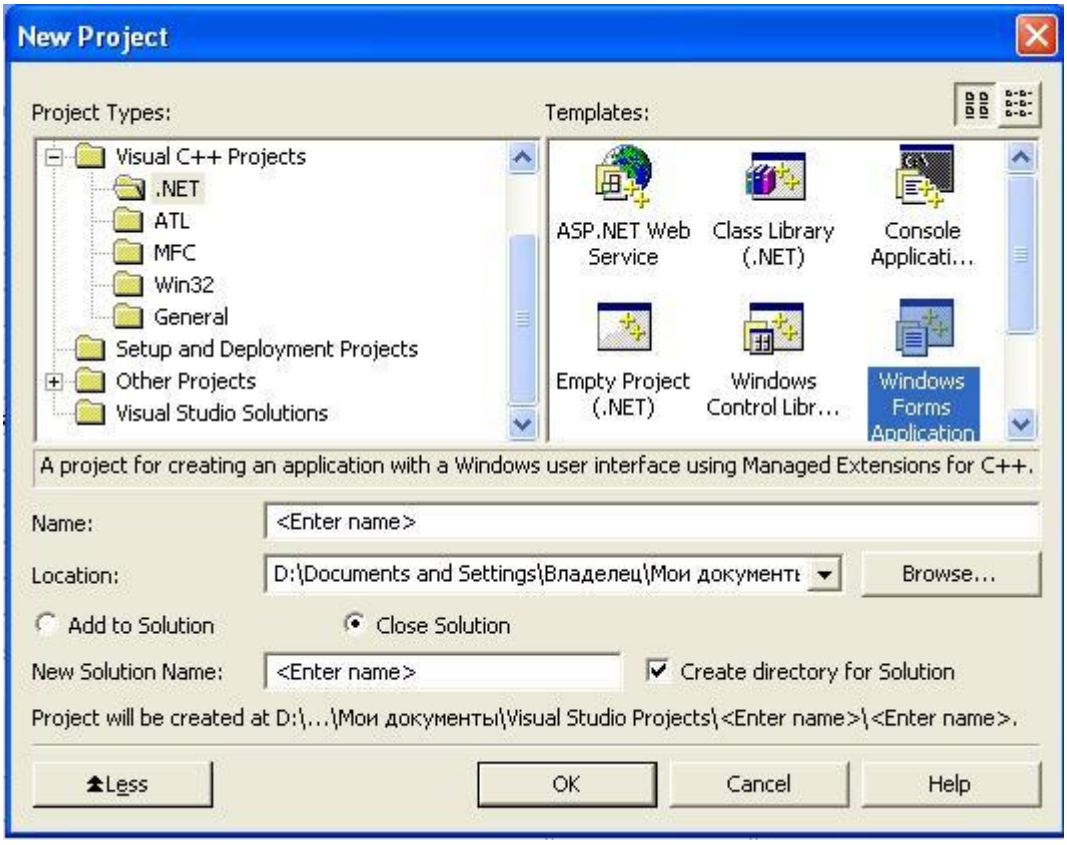
При перемещении объекта на форму приложения объявление объекта появляется в заголовочном файле. Воздействие пользователя на объект в процессе конструирования формы или выбор события из заданного списка приводит к генерации объявления метода-обработчика этого события. Синхронизация процессов проектирования формы и автоматической генерации кода ускоряет визуальную разработку приложений, полностью сохраняя контроль над исходным текстом.

Механизм двунаправленной разработки.

Технология двунаправленной разработки обеспечивает контроль над кодом посредством интегрированного и синхронизированного взаимодействия между инструментами визуального проектирования и редактором кода.

Создание проекта при помощи шаблона Windows Forms Application

Создание Windows-приложений начинается с того, что мы создаем новый проект типа Windows Application, выбирая соответствующий шаблон для одного из установленных языков программирования — C++ и т.д.



Нажатие кнопки OK приводит к загрузке выбранного нами шаблона и к появлению основных окон среды разработчика. Этими окнами являются:

- окно дизайнера форм;
- палитра компонентов;

- окно для просмотра компонентов приложения (Solution Explorer);
- окно для установки свойств (Properties).

Генерация кода

Microsoft Visual Studio .NET автоматически генерирует необходимый код для инициализации формы, добавляемых нами компонентов и т.п. Для каждой формы создается отдельный файл, который имеет то же имя, что и класс, описывающий форму.

Файл, в котором располагается код, отвечающий за функционирование формы, состоит из двух частей. Первая из них, с меткой “Windows Form Designer generated code”, содержит код, который копируется из шаблона приложения, — этот код не должен редактироваться программистами! Чтобы ограничить доступ к нему, в редакторе кода в Visual Studio .NET он помечается специальным образом.

Пространству имен предшествует 6 предложений using; это означает, что используются не менее 6-ти классов, находящихся в разных пространствах имен библиотеки FCL.

```
namespace MyApp
{
using namespace System;
using namespace System::ComponentModel;
using namespace System::Collections;
using namespace System::Windows::Forms;
using namespace System::Data;
using namespace System::Drawing;
```

Одним из таких используемых классов является класс Form из глубоко вложенного пространства имен System.Windows.Forms. Построенный по умолчанию класс Form1 является наследником класса Form и автоматически наследует его функциональность - свойства, методы, события. При создании объекта этого класса, характеризующего форму, одновременно Visual Studio создает визуальный образ объекта - окно, которое можно заселять элементами управления. В режиме проектирования эти операции можно выполнять вручную, при этом автоматически происходит изменение программного кода класса. Появление в проекте формы, открывающейся по умолчанию при запуске проекта, означает переход к визуальному, управляемому событиями рограммированию. Сегодня такой стиль является общепризнанным, а стиль консольного приложения следует считать устаревшим, правда, весьма полезным при изучении свойств языка.

В класс Form1 встроено закрытое (private) свойство - объект components класса Container. В классе есть конструктор, вызывающий закрытый метод класса InitializeComponent. В классе есть деструктор, освобождающий занятые ресурсы, которые могут появляться при добавлении элементов в контейнер components.

```
public __gc class Form1 : public System::Windows::Forms::Form
{
public:
Form1(void)
{
```

```
InitializeComponent();
}

protected:
void Dispose(Boolean disposing)
{
if (disposing && components)
{
components->Dispose();
}
__super::Dispose(disposing);
}
private:
/// <summary>
/// Required designer variable.
/// </summary>
System::ComponentModel::Container * components;

    /// <summary>
/// Required method for Designer support - do not modify
/// the contents of this method with the code editor.
/// </summary>
void InitializeComponent(void);
};
}
```

Режимы дизайна и кода

При создании нового проекта запускается режим дизайна — форма представляет собой основу для расположения элементов управления. Для работы с программой следует перейти в режим кода. Это можно сделать несколькими способами: щелкнуть правой кнопкой мыши в любой части формы и в появившемся меню выбрать View Code, в окне Solution Explorer сделать то же самое на компоненте Form 1.h или просто дважды щелкнуть на форме — при этом сгенерируется метод Form1_Load. После хотя бы однократного перехода в режим кода в этом проекте появится вкладка Form1.h , нажимая на которую, тоже можно переходить в режим кода. Для перехода в режим кода также можно использовать клавишу F7, а для возврата в режим дизайна — сочетание Shift+F7.

ГЛАВА 7. ОКНА ИНСТРУМЕНТОВ СРЕДЫ РАЗРАБОТКИ VISUAL STUDIO

1. Окно Toolbox
2. Форматирование элементов управления
3. Окно свойств (Properties window)
    1. Интерфейс окна Properties
    2. Редакторы свойств
    3. Обработка событий
4. Окно проводника классов (Class View)
5. Окно Object Browser
6. Окно Server Explorer
7. Окно динамическая справка (Dynamic Help)
8. Перемещение и изменение размеров окон инструментов
    1. Закрепление инструмента в Visual Studio
    2. Скрытие инструмента в Visual Studio

Основные инструменты в среде разработки Visual Studio - это Solution Explorer (Обозреватель решений) , окно Properties (Свойства), Dynamic Help (Динамическая справка), Windows Forms Designer (Конструктор Windows Forms), Toolbox (Область элементов) и окно Output (Вывод). Есть также несколько специализированных инструментов, в их числе Server Explorer (Обозреватель серверов) и Class View (Представление классов). Они чаще всего представлены в виде закладок вдоль границ среды разработки или в нижней части окон инструментов, или вообще невидимы. Редко используемые инструменты перенесены в подменю Other Windows (Другие окна). Если какой-либо инструмент не виден, то его можно отобразить при помощи команд меню или панели инструментов.

Точный размер и форма окон и инструментов зависят от настроек вашей среды разработки. Visual Studio позволяет располагать и закреплять окна так, чтобы сделать видимыми все необходимые элементы. Вы также можете скрыть инструменты так, что они примут форму закладок по краям среды разработки и останутся невидимыми до тех пор, пока они снова вам не понадобятся.

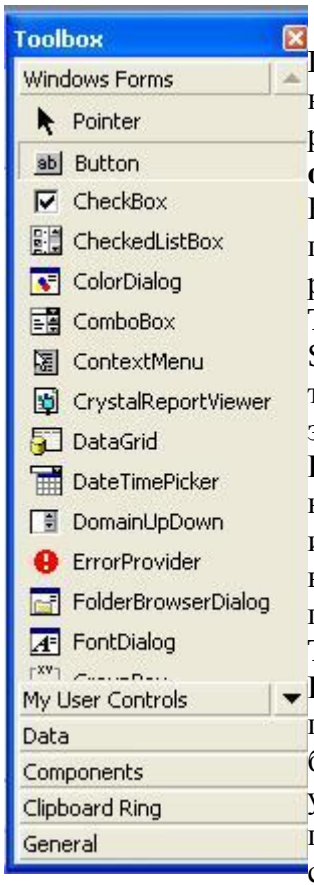
Скрывающиеся панели, расположенные по бокам окна можно выдвинуть, просто щелкнув на них. Мы можем закрепить их на экране, нажав на значок  , или совсем убрать с экрана, а затем снова отобразить, используя соответствующий пункт меню View (или эквивалентное сочетание клавиш).

Окно Toolbox

Окно Toolbox (панель инструментов, View —> Toolbox, или сочетание клавиш Ctrl+Alt+X) содержит компоненты , называемые также элементами управления, которые размещаются на форме. Элементы управления — это компоненты, обеспечивающие взаимодействие между пользователем и программой.

То, какой набор компонентов доступен в данный момент, зависит от типа разрабатываемого приложения. Например, если в данный момент разрабатывается приложение типа Windows Forms, в этом окне будут присутствовать элементы управления, которые можно использовать в Windows-приложениях; если же разрабатывается Web-форма, в этом окне будут находиться инструменты для работы с элементами управления Web Controls, и т.д. Наиболее часто употребляемой закладкой является Windows Forms. Для размещения нужного элемента управления достаточно просто щелкнуть на нем в окне Toolbox или, ухватив, перетащить его на форму.

Набор компонентов Windows Forms можно сгруппировать по нескольким функциональным группам.



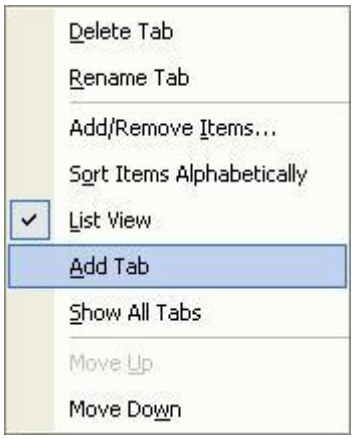
Группа командных объектов Элементы управления Button, LinkLabel, ToolBar реагируют на нажатие кнопки мыши и немедленно запускают какое-либо действие. Наиболее распространенная группа элементов. **Группа текстовых объектов**
Большинство приложений предоставляют возможность пользователю вводить текст и, в свою очередь, выводят различную информацию в виде текстовых записей. Элементы TextBox, RichTextBox принимают текст, а элементы Label, StatusBar выводят ее. Для обработки введенного пользователем текста, как правило, следует нажать на один или несколько элементов из группы командных объектов.
Группа переключателей Приложение может содержать несколько предопределенных вариантов выполнения действия или задачи; элементы управления этой группы предоставляют возможность выбора пользователю. Это одна из самых обширных групп элементов, в которую входят ComboBox, ListBox, ListView, TreeView, NumericUpDown и многие другие.
Группа контейнеров С элементами этой группы действия приложения практически никогда не связываются, но они имеют большое значение для организации других элементов управления, их группировки и общего дизайна формы. Как правило, элементы этой группы, расположенные на форме, служат подложкой кнопкам, текстовым полям, спискам — поэтому они и называются контейнерами. Элементы Panel, GroupBox, TabControl, кроме всего прочего, разделяют возможности приложения на логические группы, обеспечивая удобство работы.
Группа графических элементов Даже самое простое приложение Windows содержит графику — иконки, заставку, встроенные изображения. Для размещения и отображения их на

форме используются элементы для работы с графикой — Image List, Picture Box.

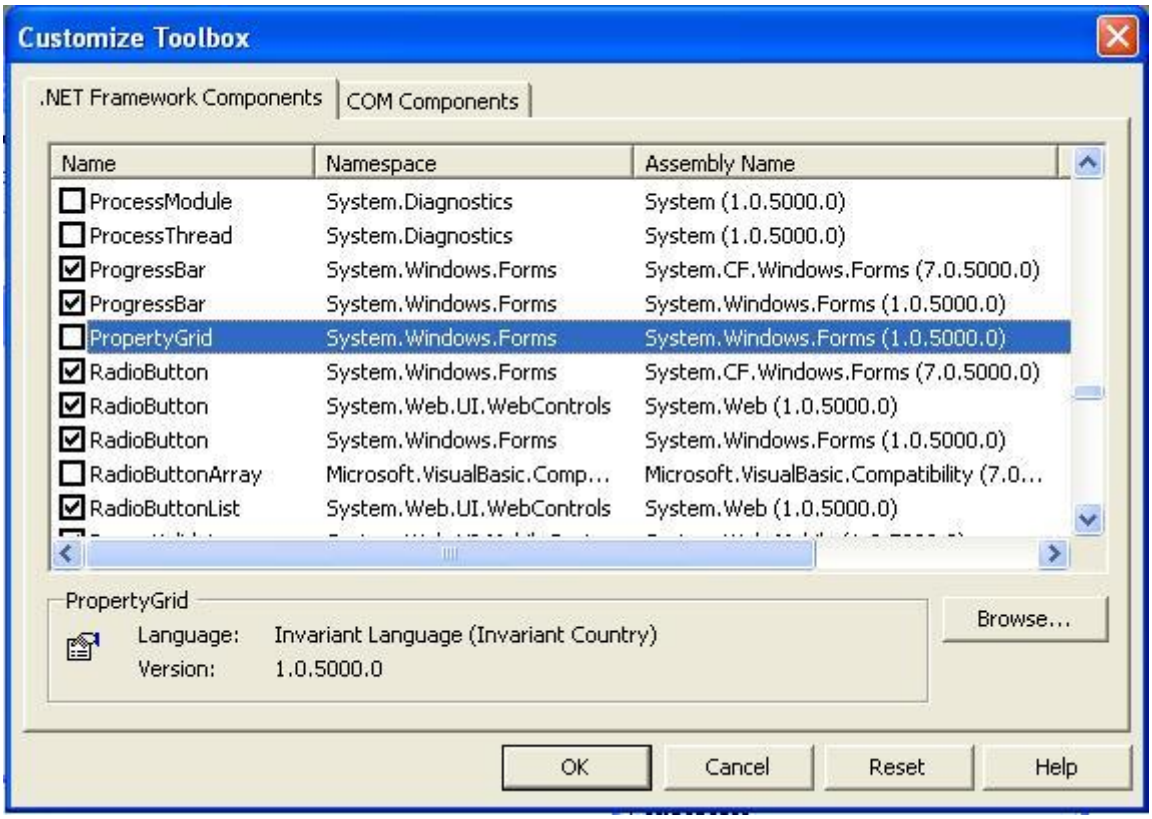
Диалоговые окна Выполняя различные операции с документом — открытие, сохранение, печать, предварительный просмотр, — мы сталкиваемся с соответствующими диалоговыми окнами. Разработчикам .NET не приходится заниматься созданием окон стандартных процедур: элементы OpenFileDialog, SaveFile Dialog, ColorDialog, PrintDialog содержат уже готовые операции.

Группа меню Меню обеспечивают доступ ко всем возможностям и настройкам приложения. Элементы MainMenu, ContextMenu представляют собой готовые формы для внесения заголовков и пунктов меню.

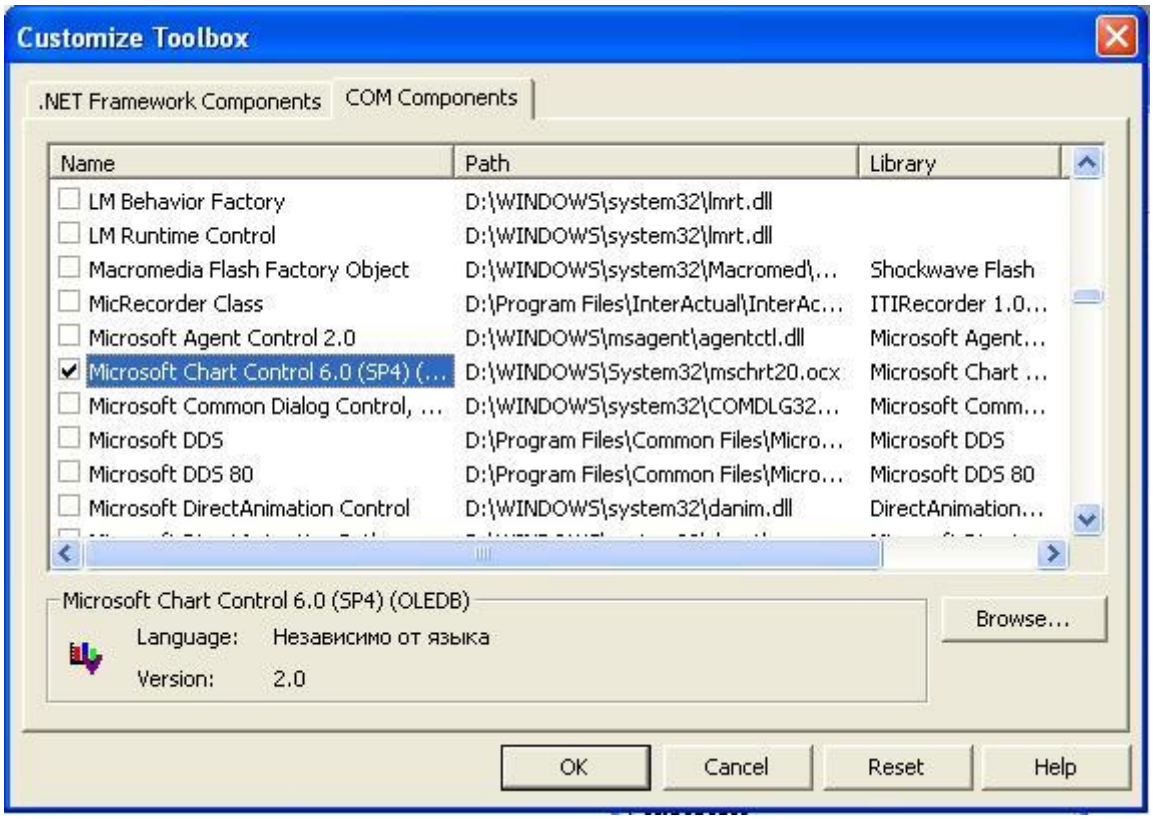
Контекстное меню содержит команды, с помощью которых можно управлять видом отображения групп компонентов, а также удалять и добавлять новые.



При необходимости можно изменить отображаемый в окне Toolbox набор элементов управления, добавив другие компоненты .NET или элементы ActiveX (в том числе созданные независимыми производителями). Для этой цели можно использовать команду меню Tools | Customize Toolbox и с помощью диалоговой панели Customize Toolbox выбрать элементы управления ActiveX или элементы управления .NET, которые мы хотим отобразить в окне Toolbox.



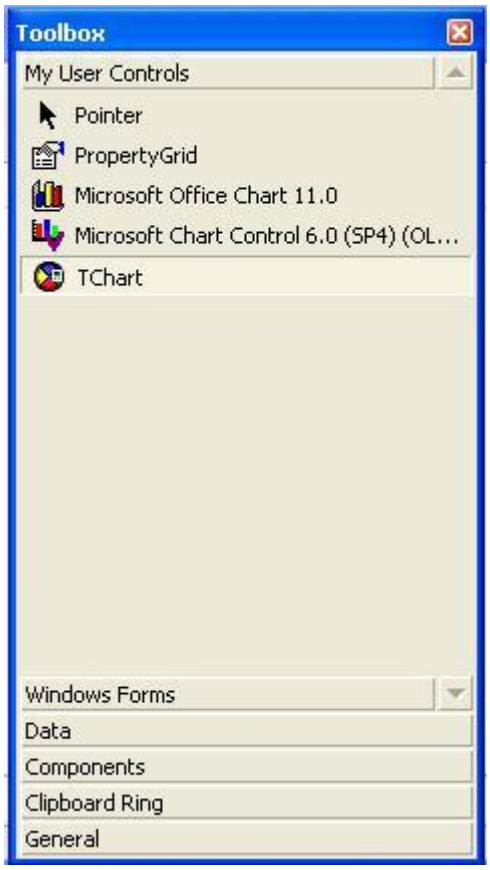
NET компонент PropertyGrid - это окно свойств , которое можно использовать в приложениях. На стадии выполнения программы PropertyGrid позволяет работать со свойствами элементов управления , расположенных на форме. Для того, чтобы связать PropertyGrid с элементов управления , необходимо его свойство SelectedObject установить равным имени элемента управления. Как видно из рисунка по умолчанию компонент PropertyGrid не отображается на панели Toolbox.



На вкладке COM Components содержаться доступупные, зарегистрированные в реестре элементы ActiveX. Не зарегистрированные в реестре компоненты можно поместить в виде файлов dll в папку сборки приложения

Окно Toolbox состоит из нескольких закладок: My User Controls, Components, Data, Windows Forms и General . Все доступные вкладки можно отобразить , используя команду контекстного меню Show All Tabs

Используя команду контекстного меню Add Tab можно создать свою собственную закладку и хранить собственные списки элементов управления. Наиболее часто используемые элементов управления имеет смысл перетащить на эту закладку



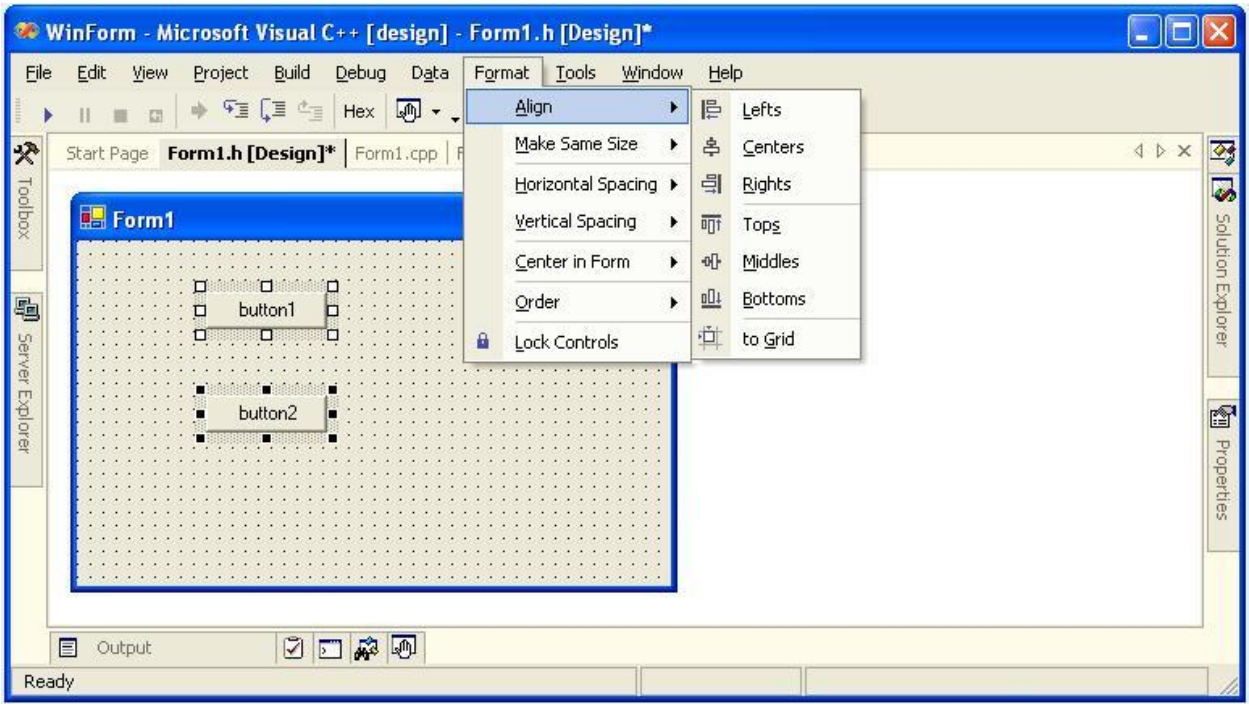
Переключение вида значков позволяет разместить их без полосы прокрутки, а также в виде списка.

Созданные таким образом закладки можно переименовать или удалить, выбрав в контекстном меню пункты Rename Tab и Delete Tab соответственно.

Все закладки, кроме Clipboard Ring и General, содержат компоненты, которые можно перетащить на форму. Закладка Clipboard Ring представляет собой аналог буфера обмена в Microsoft Office 2003, отображающего содержимое буфера за несколько операций копирования или вырезания. Для вставки фрагмента достаточно дважды щелкнуть по нему.

Форматирование элементов управления

Расположение элементов на форме во многом определяет удобство работы с готовым приложением. Пункт главного меню Format содержит опции выравнивания, изменения размера и блокировки элементов управления



Пункт главного меню Format

При выделении нескольких элементов управления около одного из них появляются темные точки маркера Свойства выбранных элементов будут изменяться относительно этого, главного элемента управления. Для последовательного выделения нескольких элементов удерживаем клавишу Shift, главным элементом будет последний выделенный элемент. Значение пунктов меню Format приводятся в таблице

Таблица

Пункт меню Format	**Описание**
Align	Выравнивание выбранных элементов управления
Make Same Size	Установка равного размера
Horizontal Spacing	Пробел между элементами по горизонтали
Vertical Spacing	Пробел между элементами по вертикали
Center in Form	Расположение элементов управления относительно формы
Order	Вертикальный порядок элементов управления
Lock Controls	Блокировка элементов

Окно свойств (Properties Window)

Окно свойств Properties — основной инструмент настройки формы и ее компонентов. Содержимое этого окна представляет собой весь список свойств выбранного в данный момент компонента или формы. При создании проекта, в окне Properties отображаются свойства самой формы. При активизации элемента управления на форме окно Properties

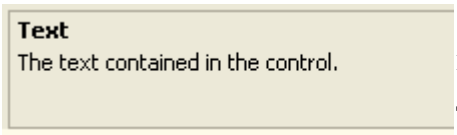
автоматически покажет свойства и события, которые могут быть использованы с этим элементом управления.

Окно Properties содержит селектор объектов и две страницы свойств и событий. Выпадающие вниз список селектора объектов показывает имена и типы объектов каждого элемента управления в текущей форме, включая и сами форму. Селектор объектов используется для быстрого переключения между каждым из элементов управления.

В окне Properties (Свойства) есть две удобные кнопки, которые можно использовать для организации свойств. Кнопка Alphabetic (По алфавиту) сортирует все свойства в алфавитном порядке и сворачивает их в нескольких категориях. (Если вам известно имя свойства, и вы хотите его быстро найти, нажмите эту кнопку.) Кнопка Categorized (По категориям) распределяет свойства по различным логическим группам. (Если вы не знаете всех свойств настраиваемого объекта и хотите сгруппировать свойства по их типам, нажмите эту кнопку.)

В таблице приводится описание интерфейса окна Properties.

Таблица

Элемент	Изображение	Описание
Object name		В поле этого списка выводится название данного выбранного объекта, который является экземпляром какого-либо класса. Здесь Form1 — название формы по умолчанию, которая наследуется от класса System.Windows.Forms.Form
Categorized		При нажатии на эту кнопку производится сортировка свойств выбранного объекта по категориям. Можно закрывать категорию, уменьшая число видимых элементов. Когда категория скрыта, вы видите знак (+), когда раскрыта — (–)
Alphabetic		Сортировка свойств и событий объекта в алфавитном порядке
Properties		При нажатии на эту кнопку отображается перечисление свойств объекта
Events		При нажатии на эту кнопку отображается перечисление событий объекта
Description Pane		Панель, на которую выводится информация о выбранном свойстве. В данном случае в списке свойств формы

было выбрано свойство Text

Редакторы свойств

Каждый элемент управления обладает своим заранее определенным набором свойств. Visual Studio организует их по категориям и отображает в виде структуры. Окно Properties позволяет использовать несколько способов для показа свойств и изменения их значений. Эти механизмы называют редакторами свойств. Существует несколько типов редакторов свойств:

1. Строка ввода.
2. Редактор с выпадающим списком значений. Его признаком является стрелка вниз в колонке значений.
3. Диалоговая панель. Признак – кнопка с многоточием (…) в колонке значений.
4. Набор вложенных свойств. На существовании вложенного списка подсвойств указывает знак (+) в имени свойства.

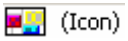
Свойств формы

Окно Properties позволяет определять в первую очередь дизайн формы и ее элементов управления. В таблице приводится описание некоторых свойств формы, обычно определяемых в режиме дизайна. При выборе значения свойства, отличного от принятого по умолчанию, оно выделяется жирным шрифтом, что облегчает в дальнейшем определение изменений.

Таблица . Некоторые свойства формы

Свойство	Описание	Значение по умолчанию
Name	Название формы в проекте. Это не заголовок формы, который вы видите при запуске формы, а название формы внутри проекта, которое вы будете использовать в коде	Form1, Form 2 и т.д.
AcceptButton	Устанавливается значение кнопки, которая будет срабатывать при нажатии клавиши Enter. Для того чтобы это свойство было активным, необходимо наличие по крайней мере одной кнопки, расположенной на форме	None
BackColor	Цвет формы. Для быстрого просмотра различных вариантов просто щелкайте прямо на названии "BackColor"	Control
BackgroundImage	Изображение на заднем фоне	None

CancelButton	Устанавливается значение кнопки, которая будет срабатывать при нажатии клавиши Esc. Для того чтобы это свойство было активным, необходимо наличие по крайней мере одной кнопки, расположенной на форме	None
ControlBox	Устанавливается наличие либо отсутствие трех стандартных кнопок в верхнем правом углу формы: "Свернуть", "Развернуть" и "Закрыть"	
Cursor	Определяется вид курсора при его положении на форме	Default
DrawGrid	Устанавливается наличие либо отсутствие сетки из точек, которая помогает форматировать элементы управления. В любом случае сетка видна только на стадии создания приложения	True
Font	Форматирование шрифта, используемого для отображения текста на форме в элементах управления	Microsoft Sans Serif; 8,25pt
FormBorderStyle	Определение вида границ формы. Возможные варианты: · None — форма без границ и строки заголовка; · FixedSingle — тонкие границы без возможности изменения размера пользователем; · Fixed3D — границы без возможности изменения размера с трехмерным эффектом; · FixedDialog — границы без возможности изменения, без иконки приложения; · Sizable — обычные границы: пользователь может изменять размер границ;	Sizable

	· FixedToolWindow — фиксированные границы, имеется только кнопка закрытия формы. Такой вид имеют панели инструментов в приложениях; · SizableToolWindow — границы с возможностью изменения размеров, имеется только кнопка закрытия формы	
Icon	Изображение иконки, располагаемой в заголовке формы. Поддерживаются форматы .ico	 (Icon)
MaximizeBox	Определяется активность стандартной кнопки "Развернуть" в верхнем правом углу формы	True
MaximumSize	Максимальный размер ширины и высоты формы, задаваемый в пикселях. Форма будет принимать указанный размер при нажатии на стандартную кнопку "Развернуть"	0;0 (Во весь экран)
MinimizeBox	Определяется активность стандартной кнопки "Свернуть" в верхнем правом углу формы	True
MinimumSize	Минимальный размер ширины и высоты формы, задаваемый в пикселях. Форма будет принимать указанный размер при изменении ее границ пользователем (если свойство FormBorderStyle имеет значение по умолчанию Sizable)	0;0
Size	Ширина и высота формы	300; 300
StartPosition	Определение расположения формы при запуске приложения. Возможны следующие значения: · Manual — форма появляется в верхнем левом углу экрана; · CenterScreen — в центре экрана; · WindowsDefaultLocation	WindowsDefaultLocation

	— расположение формы по умолчанию. Если пользователь изменил размеры формы, то при последующем ее запуске она будет иметь тот же самый вид и расположение; · WindowsDefaultBounds — границы формы принимают фиксированный размер; · CenterParent — в центре родительской формы	
Text	Заголовок формы. В отличие от свойства Name, это именно название формы, которое не используется в коде	Form1, Form 2 и т.д.
WindowState	Определение положения формы при запуске. Возможны следующие значения: · Normal — форма запускается с размерами, указанными в свойстве Size; · Minimized — форма запускается с минимальными размерами, указанными в свойстве MinimumSize; · Maximized — форма разворачивается на весь экран	Normal

События, на которые реагируют компоненты.

События, на которые может реагировать данный компонент, содержится в списке страницы событий окна Properties. Набор событий заранее определен для данного класса компонентов.

Примеры событий

On Change Происходит при изменении значения введеного в поле ввода.
On Click щелчок мыши на компоненте
On Dblclick двойной щелчок
On Mouse Down нажатие кнопки мыши когда курсор находится на компоненте
On Mouse Move перемещение курсора мыши над компонентом
On Mouse Up пользователь отпускает кнопку мыши при условии что курсор был на компоненте
On Enter компонент получает фокус ввода
On Exit теряет фокус ввода

On Key Down нажатие любой клавиши, когда компонент имеет фокус ввода
On Key Press нажатие ASCII клавиши
On Key Up пользователь отпускает нажатую клавишу

Обработка событий

Кнопка окна свойств Events (События)  переключает окно Properties в режим управления обработчиками различных событий (например, мыши, клавиатуры) и одновременно выводит список всех событий компонента.
Окно Properties в позволяет связать с событием функцию, которая будет вызываться в случае, если данное событие произойдет. Есть стандартный способ включения событий. Достаточно выделить нужный элемент в форме, в окне свойств нажать кнопку событий (со значком молнии) и из списка событий выбрать нужное событие и щелкнуть по нему. Двойной щелчок мыши на колонке значений вызывает генерацию заголовка функции для обработки события. Курсор распологается внутри фигурных скобок, где пользователь должен набрать свой собственный код. Обработчик событий может иметь параметры, которые указываются после имени в круглых скобках. Имя обработчика формируется из имени объекта, которое включает его порядковый номер на форме и названия события. Задать обработчик события для кнопки можно еще проще. Двойной щелчок по кнопке включает событие, и автоматически строится заготовка обработчика события с нужным именем и параметрами

Изменение свойств шрифта

· Нажмите кнопку многоточия в строке Font.

Visual Studio покажет диалоговое окно Font (Шрифт), в котором можно уточнить параметры шрифта для выбранной надписи. Каждому выбранному вами параметру соответствует отдельная строка в окне Properties (Свойства), и ее значение будет изменено соответствующим образом.

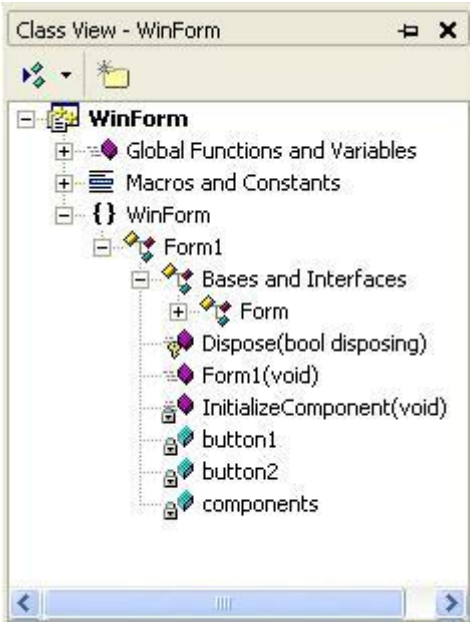
· Измените размер шрифта с 10 до 12 точек, а затем измените начертание с обычного на **курсив**. Чтобы подтвердить сделанные изменения, нажмите кнопку OK.

· Прокрутите окно Properties (Свойства) до свойства ForeColor, а затем щелкните мышкой в левом столбце.

В правом столбце нажмите кнопку раскрытия списка, а в нем выберите закладку Custom (Польз.) и на ней на синий цвет.

Окно проводника классов (Class View)

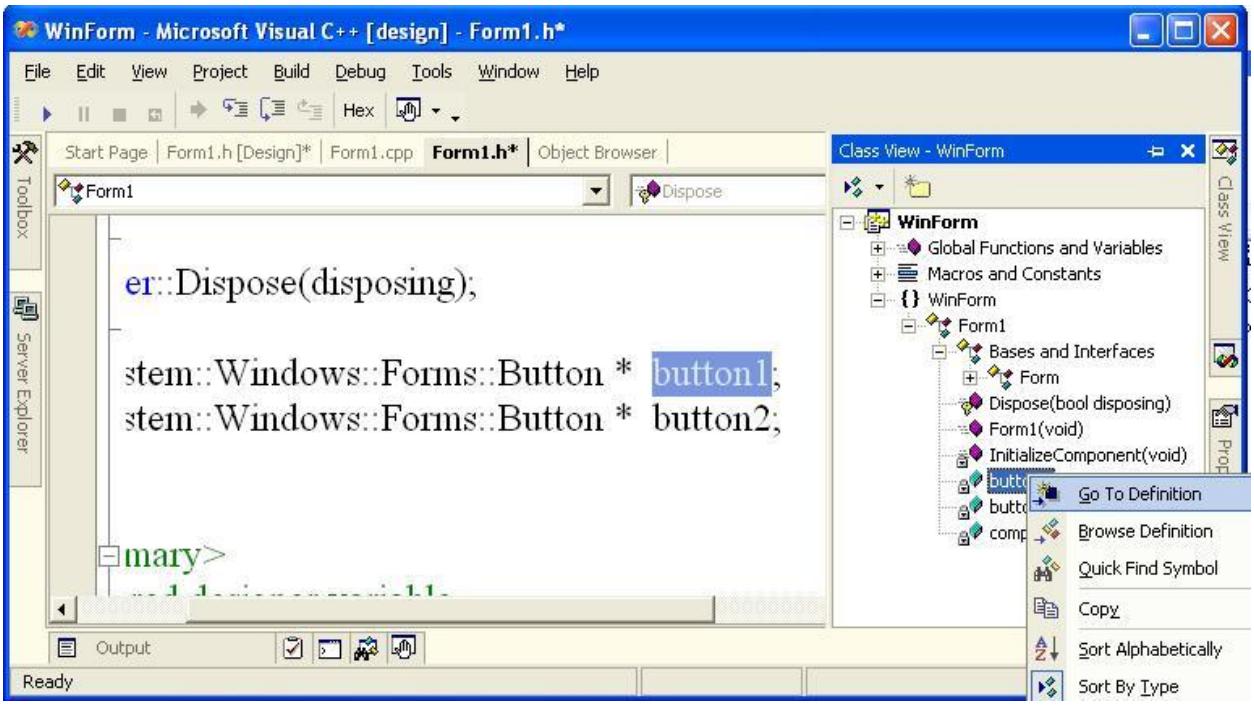
Для навигации внутри модуля используется проводник классов, содержащий в виде иерархической структуры все типы, классы, глобальные переменные и функции определённые в модуле.



Содержимое проводника классов синхронизировано с содержимым редактора кода. Изменения в коде отображаются в проводнике классов и, наоборот, изменения в проводнике приводят к соответствующимизменениям в коде приложения. ClassView позволяет при помощи drag&drop вытаскивать имена методов, свойств и классов напрямую в окно редактирования текста, . В Visual Studio .NET, Class View также содержит список методов и свойств базовых классов, что позволяет легко переходить к описанию этих методов и свойств в Object Browser (двойной щелчок по выбранному элементу списка).

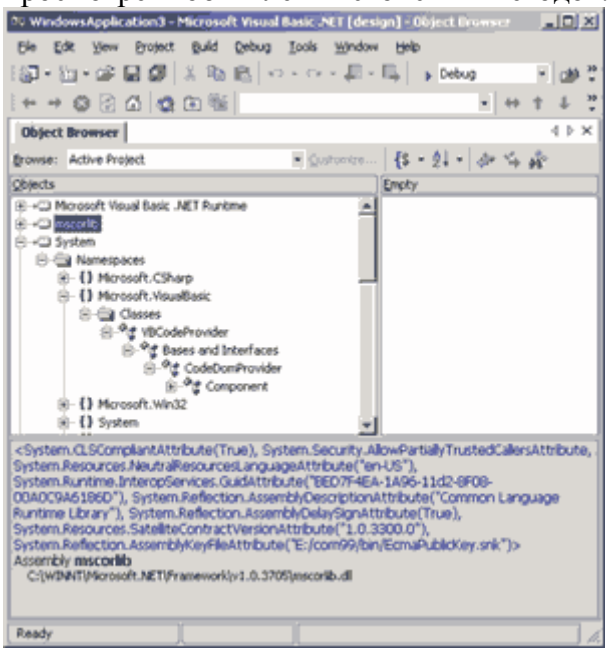
Добавить в проект новый класс

С помощью контекстного меню можно найти объявление и реализацию выбранного класса, а также осуществлять поиск определения функции.



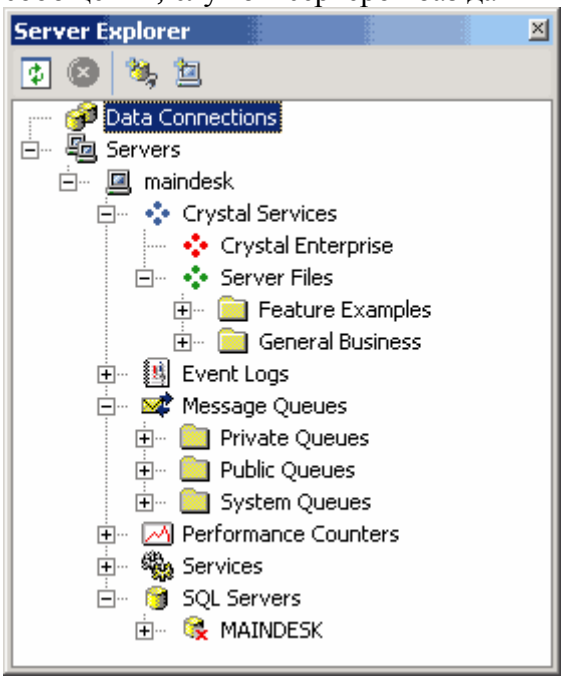
Окно Object Browser

Окно Object Browser, доступное с помощью команды меню View | Other Windows | Object Browser , так же как и окно Class View, позволяет просмотреть список классов, их свойств и методов. Однако Object Browser позволяет просмотреть все компоненты, на которые ссылается класс, а также, при необходимости, компоненты, на которые нет ссылок в данном проекте, тогда как с помощью окна Class View можно просматривать сведения только о классах из данного проекта. С помощью Object Browser можно также просмотреть объявления свойств и методов.



Окно Server Explorer

Окно Server Explorer (команда меню View | Server Explorer), позволяет просматривать сведения о службах, выполняющихся на конкретных серверах. К таким службам, в частности, относятся службы Crystal Reports Services, журнал событий, очереди сообщений, службы серверов баз данных, таких как Microsoft SQL Server.



Окно Server Explorer
Большинство этих служб представлено в окне в виде иерархического дерева, позволяющего просматривать сведения, связанные с данными службами, и иногда добавлять новые элементы. С помощью перетаскивания значка службы или ее элемента в дизайнер можно организовать ее использование в приложении. Так, при переносе значка таблицы сервера баз данных на форму разрабатываемого приложения можно создать компонент DataAdapter для извлечения данных из этой таблицы.

Окно динамическая справка (Dynamic Help)

В комплект Visual Studio .NET входит оперативное справочное руководство, в котором можно найти необходимые сведения о среде разработки Visual Studio, языках программирования ресурсах .NET Framework и остальных инструментах, входящих в набор Visual Studio.

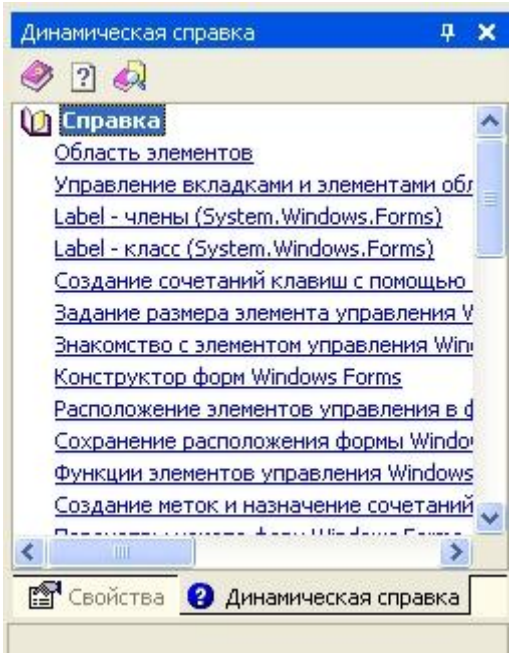
Доступ к справочной информации

Инструмент Dynamic Help (Динамическая справка) , в зависимости от того, над чем вы работаете в данный момент, заранее подбирает справочные разделы и показывает их в своем окне. Для ограничения выбора материалов используется анализ контекста, и вы увидите только те из них, которые относятся к конкретному компилятору или инструменту. (Другими словами, в Dynamic Help (Динамической справке) не появятся заголовки, относящиеся к Visual C#, если вы не работаете с этими инструментами.)

Динамическая справка позволяет выполнять полнотекстовый поиск по справочной системе Visual Studio. Полнотекстовый поиск полезен, когда нужно разыскать текст по отдельным ключевым словам.

Получение справки с помощью Dynamic Help (Динамической справки)

1. Щелкните на закладке Dynamic Help (Динамическая справка) в среде разработки или выберите строку Dynamic Help (Динамическая справка) в меню Help (Справка). Появится окно динамической справки, показанное ниже.



Окно Dynamic Help (Динамическая справка) входит в состав Visual Studio. Его можно перемещать, изменять размер, закреплять или скрывать. Вы можете оставить его всегда открытым или открывать только тогда, когда оно нужно.

2. Выберите какой-нибудь пункт в окне динамической справки. Скорее всего, это будет текст про область элементов или про окно Properties (Свойства).

3. Просмотрите несколько других статей из списка динамической справки. Обратите внимание, насколько эти темы соответствуют реальным действиям, которые вы выполняете.

Отбор и показ справочной информации можно настроить. В меню Tools (Сервис) выберите команду Options (Параметры) и в окне свойств раскройте папку Environment (Среда). Затем выберите пункт Dynamic Help (Динамическая справка) и определите, какие темы вы хотите видеть в справке и сколько ссылок будет показано одновременно. Эти настройки помогут вам

Перемещение и изменение размеров окон инструментов

Когда на экране одновременно видны много инструментов программирования, среда разработки Visual Studio выглядит очень перегруженной. Visual Studio позволяет перемещать, изменять размеры, закреплять и использовать функцию автоматического скрытия для большинства элементов интерфейса, которые используются для создания программ.

Чтобы передвинуть любое из окон инструментов Visual Studio, просто щелкните на его строке заголовка и перетащите объект в другое место. Если вы придвинете одно окно к краю другого окна, то оно закрепится. Преимущество закрепляемых окон в том, что они всегда видны (они не скрываются за другими окнами). Если вы хотите увеличить закрепляемое окно, просто раздвиньте его границу.

Чтобы полностью закрыть окно, нужно нажать кнопку Close (Закрыть) в верхнем правом углу этого окна. Вы всегда сможете открыть это окно снова, выбрав соответствующую команду в меню View (Вид).

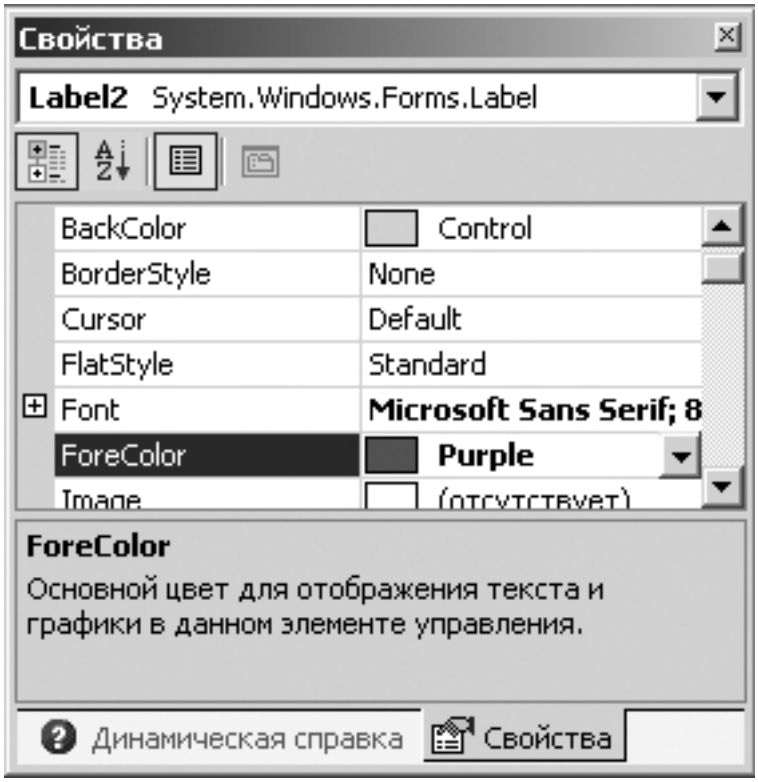
В среде разработки Visual Studio имеется функция автоскрытия окна инструмента, которую можно задействовать, щелкнув в правой части заголовка окна по канцелярской кнопке Auto Hide (Автоскрытие). Это действие убирает окно с его закрепленной позиции, а заголовок окна остается на краю среды разработки на закладке, которая не занимает много места. При выполнении автоскрытие окна, окно инструмента будет оставаться видимым до тех пор, пока вы держите на нем указатель мыши. Как только мышь перемещается в другую часть среды разработки, это окно свернется и превратится в закладку.

Чтобы восстановить окно, для которого активизировано автоскрытие, нажмите на его закладку на краю среды разработки или некоторое время подержите над ней указатель мыши. Определить, включена ли функция автоскрытия для данного окна, можно по канцелярской кнопке в его правом углу. Удерживая указатель мыши над заголовком, вы можете быстро вызвать скрытое окно, щелкнув на его закладке, проверить или задать требуемую информацию и затем передвинуть мышь, чтобы окно исчезло. Если вам нужно, чтобы окно было открыто постоянно, снова нажмите кнопку Auto Hide (Автоскрытие), чтобы острие канцелярской кнопки смотрело вниз. Окно останется видимым.

Перемещение и изменение размеров окна Properties (Свойства)

1. Если окно Properties (Свойства) не отображается в среде разработки, нажмите кнопку Properties Window (Окно свойств) на стандартной панели инструментов. При этом окно появится, а его заголовок будет подсвечен.

2. Чтобы сделать окно Properties (Свойства) плавающим (незакрепленным), дважды щелкните мышью на заголовке окна. Окно будет выглядеть так, как показано на следующем рисунке.



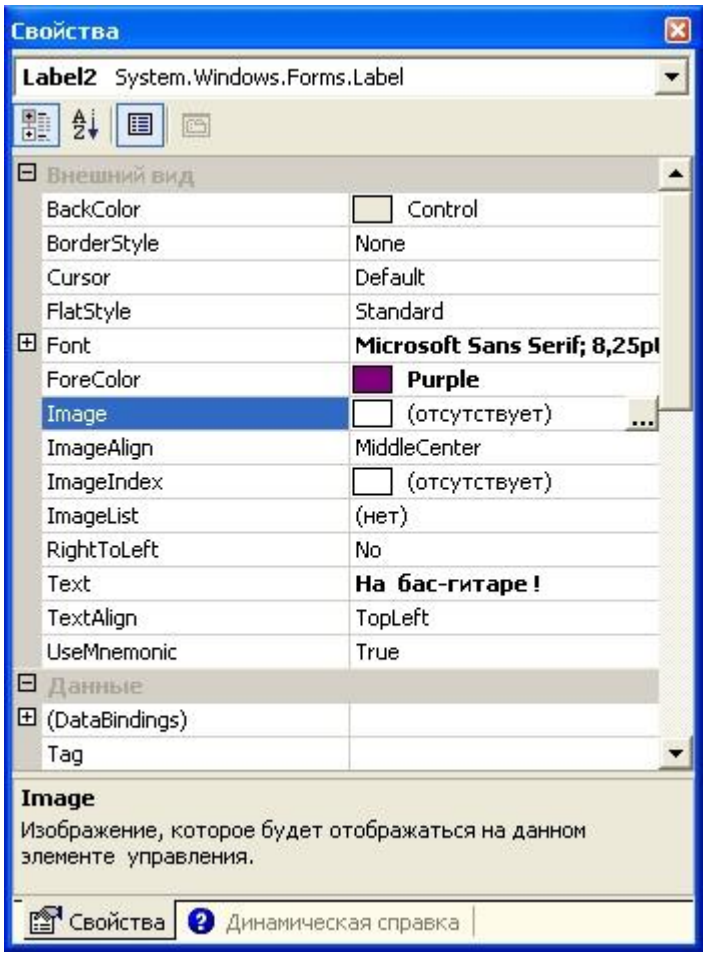
3. Используя строку заголовка окна Properties (Свойства), перетащите его в другое место в среде разработки, но не позволяйте ему закрепиться. Возможность перемещать окна инструментов в среде разработки Visual Studio позволяет организовать ее удобным для вас образом. Теперь давайте изменим размеры окна Properties (Свойства), чтобы увидеть больше свойств объекта.

4. Передвиньте указатель мыши к нижнему правому углу окна Properties (Свойства), так, чтобы он принял форму стрелки для изменения размера.

Размеры окон Visual Studio можно изменить тем же способом, что и окна других приложений в операционной системе Microsoft Windows.

5. Чтобы увеличить окно Properties (Свойства), потяните нижний правый угол окна вниз и вправо.

Теперь окно стало больше.



Окно большего размера позволяет работать быстрее и легче находить нужную информацию. Вы можете свободно изменять размеры окна, если необходимо увидеть большую часть его содержимого.

Закрепление инструмента в Visual Studio

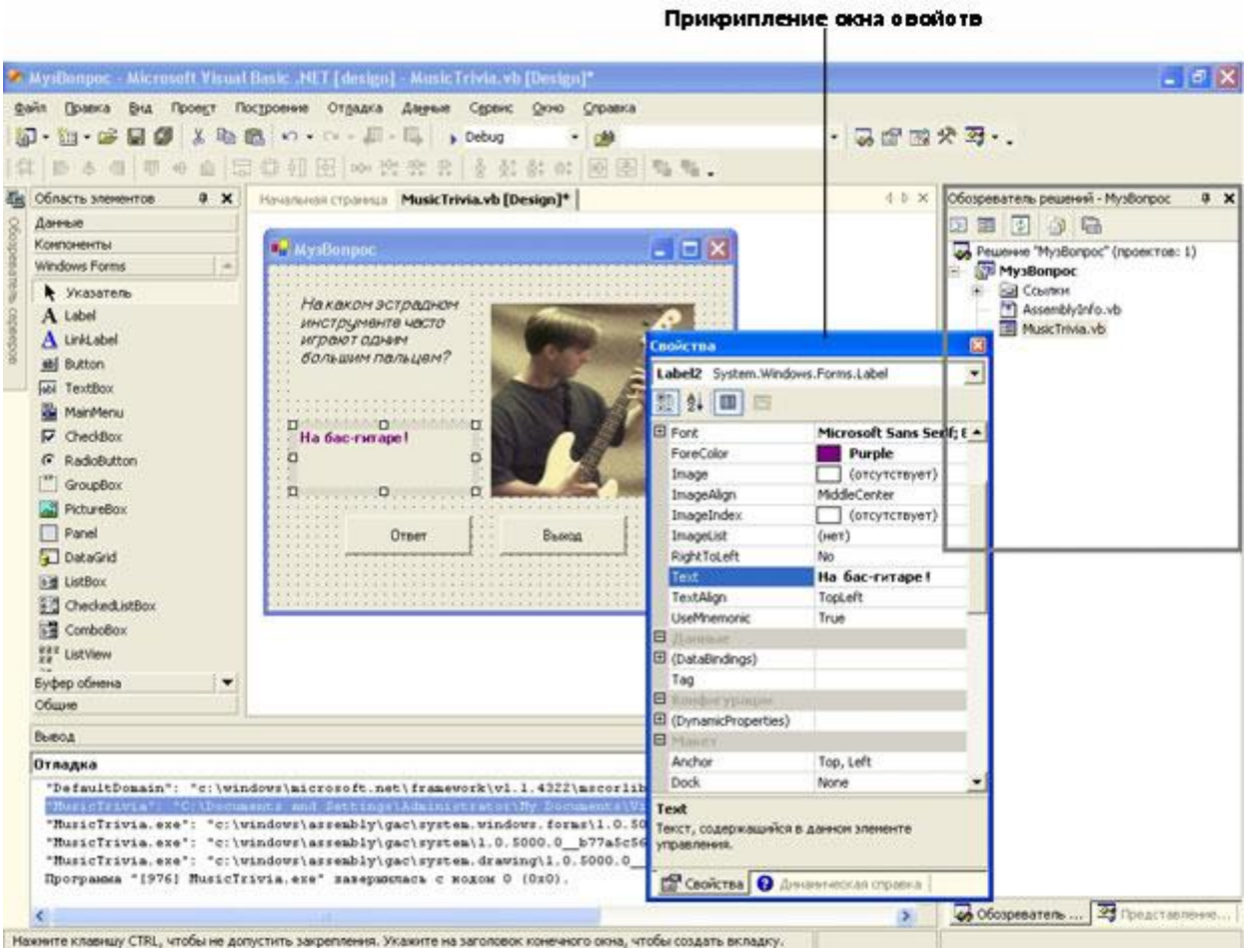
Если инструмент в среде разработки представлен плавающим окном, вы можете вернуть его в его первоначальное закрепленное положение, дважды щелкнув мышью на строке его заголовка. Когда окно находится в плавающем незакрепленном режиме, его можно закрепить в новом месте.

Закрепление окна Properties (Свойства)

1. Убедитесь, что окно Properties (Свойства) (или другой инструмент, который вы хотите передвинуть) представлено в среде разработки Visual Studio плавающим окном и не закреплено.

2. Перетащите окно Properties (Свойства) за его строку заголовка к верхнему, нижнему, правому или левому краю среды разработки, пока граница окна не "прилипнет" к краю окна среды разработки.

3. Чтобы закрепить окно Properties (Свойства), отпустите кнопку мыши. Окно будет находиться в том месте, где вы его оставили.



Чтобы предотвратить закрепление при перетаскивании окна, нажмите и удерживайте при перетаскивании клавишу (Ctrl). Если вы хотите, чтобы передвигаемое окно было совмещено с другим окном в виде закладки, перетащите это окно непосредственно на строку заголовка другого окна. Когда окна связаны друг с другом подобным образом, в нижней части их общего окна появится строка с закладками для каждого из них, и вы сможете переключаться между этими окнами, выбирая их по закладкам. Окна с закладками позволяют эффективно использовать пространство одного окна для двух и более задач. (Например, часто объединяют в виде закладок окна Solution Explorer (Обозреватель решений) и Class View (Представление классов).)

Скрытие инструмента в Visual Studio

В Visual Studio .NET имеется механизм для быстрого скрытия и отображения инструментов, который называется автоскрытие. Функция автоскрытия доступна для большинства окон инструментов. Преимущество автоскрытия для окон заключается в том, что при этом они освобождают значительное место в рабочей области Visual Studio, оставаясь при этом доступными.

Чтобы скрыть окно инструмента, нажмите кнопку Auto Hide (Автоскрытие) в правой части заголовка окна, что приведет к сворачиванию его в закладку на краю среды разработки. А чтобы закрепить окно, нажмите кнопку автоскрытия еще раз. Те же действия можно выполнить с помощью команды Auto Hide (Автоскрытие) из меню Window (Окно). Заметьте, что функция и кнопка автоскрытия доступны только для закрепленных окон. Вы не увидите команды Auto Hide (Автоскрытие) или этой кнопки

для активного окна, плавающего поверх среды разработки.

Использование функции автоскрытия

1. Найдите в среде разработки Toolbox (Область элементов) (ее окно обычно открыто в левой части Конструктора Windows Forms). В области элементов находятся элементы управления, которые можно использовать для создания приложений .

2. Найдите кнопку Auto Hide (Автоскрытие) на заголовке окна Toolbox (Область элементов). Канцелярская кнопка на ней нарисована в вертикальном, или приколотом, положении, что означает, что область элементов "приколота" в открытом положении и автоскрытие отключено.

3. Нажмите кнопку автоскрытия в строке заголовка и удерживайте указатель мыши в окне области элементов. Кнопка автоскрытия теперь выглядит как канцелярская кнопка, которая смотрит влево. Это говорит о том, что область элементов больше не "приколота" в открытом положении и будет работать автоскрытие. На левой стороне среды разработки появилась закладка с надписью Toolbox (Область элементов). Также обратите внимание, что Конструктор Windows Forms сдвинулся влево. Однако если указатель мыши все еще находится в рамках окна области элементов, в самой области элементов ничего не изменится. Разработчики Visual Studio решили, что будет лучше, если окно с включенным автоскрытием не будет исчезать до тех пор, пока пользователь не передвинет указатель мыши в другую область среды разработки.

4. Передвиньте мышь из окна Toolbox (Область элементов). Как только вы это сделаете, Toolbox (Область элементов). скроется за пределы экрана и останется только маленькая закладка. (Над закладкой области элементов также видна закладка Server Explorer (Обозреватель серверов) - указание на то, что автоскрытие включено еще для одного окна инструмента. В зависимости от настроек Visual Studio, могут быть видны и другие закладки для окон среды разработки, для которых включено автоскрытие.)

5. Подержите указатель мыши над закладкой Toolbox (Область элементов). (При желании можно щелкнуть на этой закладке.)

Ее окно немедленно выдвинется обратно в пределы видимости, и вы сможете использовать элементы управления для создания интерфейса пользователя.

6. Передвиньте указатель мыши за пределы области элементов, и инструмент снова исчезнет.

7. Наконец, снова раскройте область элементов, а затем нажмите кнопку автоскрытия в строке ее заголовка. Область элементов вернется в знакомое закрепленное положение, и вы сможете использовать ее постоянно.

СПИСОК ЛИТЕРАТУРЫ

1. Оберг Роберт Дж., Торстейнсон,Питер. Архитектура .NET и программирование с помощью Visual C++.:Пер.с англ. М.:Издательский дом "Вильяме",2002. 656 с.

2. Федоров А., Елманова Н.. Изучаем Visual Studio .NET. Часть 1. Знакомство со средой разработки// "КомпьютерПресс" № 4,2002

3. Федоров А., Елманова Н.. Изучаем Visual Studio .NET. Часть 2. Создание Windows-приложений// "КомпьютерПресс" № 5, 2002

4. Федоров А., Елманова Н.. Изучаем Visual Studio .NET Studio .NET. Часть 3. Компоненты Windows-приложений //"КомпьютерПресс" № 6, 2002

5. Колесов А.. Среда разработки Visual Studio .NET//"BYTE/Россия" № 8, 2002

6. Биллиг В. А.. Язык C# и каркас среды .NET //"BYTE/Россия" № 12, 2001

7. Чеповский А. М., Макаров А. В., Скоробогатов С. Ю.. Common Intermediate Language и системное программирование в Microsoft .NET //Интернет-Университет Информационных Технологий http://www.INTUIT.ru

8. Кариев Ч. А.. Создание Windows-приложений на основе Visual C# //Интернет-Университет Информационных Технологий http://www.INTUIT.ru

9. Хальворсон М.. Visual Basic .NET //Интернет-Университет Информационных Технологий http://www.INTUIT.ru

10. Биллиг В. А. Основы программирования на C# //Интернет-Университет Информационных Технологий http://www.INTUIT.ru

11. Троелсен.Э. C# и платформа .NET. Библиотека программиста. СПб. : Питер, 2004.796 с.

12. Рихтер Д. Программирование на платформе Microsoft .NET Framework СПб.: Русская редакция, 2005. 512 с

13. Робисон У. C# без лишних слов. ДМК, 2002. 352 с

14. Шилдт Г. C#. Учебный курс. СПб., 2002. 511 с

15. Шилдт Г. Полный справочник по C# . Вильямс, 2004, 752 с

16. Фролов А., Фролов Г. Язык C#. Самоучитель. М.: Диалог-МИФИ, 2002. 560 с