Введение в .NET AutoCAD .NET API

Задачи курса

- Понять:
  - Что такое основы AutoCAD .NET API
  - Как заниматься самообучением AutoCAD .NET API
  - Где получить помощь
- Чего не будет:
  - Полного описания всех функций API
- Специальная тема:
  - Научить Вас основам .NET framework на языке VB.NET

Повестка дня

- Обзор .NET
- Основы написания приложений AutoCAD .NET API
- Взаимодействие с пользователем
- Основы базы данных (dwg)
- Словари
- Пользовательский интерфейс
- События
- Input PointMonitor
- Jigs

Обзор .NET: Основы Framework

- Почему .NET?
- Наиболее часто используемые функции
- Среда разработки

Почему мы используем .NET?

- Стандартный инструмент разработки Microsoft
- Autodesk использует .NET как среду разработки для большинства своих продуктов
- Мы можем использовать те же самые навыки программирования для различных продуктов Autodesk

Преимущества .NET

- Огромная коллекция готовых для использования функций, доступных внутри .NET Framework
- Множество доступных языков программирования – выбирай любой
- Легкость обучения – быстрое начало разработки

Часто используемые возможности .NET

- Базовые типы
  - Позволяют легко манипулировать базовыми данными (целые, плавающие, строки, массивы)
- WinForms
  - Легкость создания форм (диалогов)
    - Создаются при помощи перетаскивания (drag’n’drop)
    - Огромное количество готовых контролов (control)
- Многие сторонние разработчики создают и предоставляют возможности посредством .NET

Продукты Autodesk предоставляют .NET

- Пока не все, но многие
- Для использования Ваше приложение должно ссылаться на API-сборку продукта (.dll)
- Глубокая интеграция с Microsoft .NET
  - Методы можно использовать вместе, смешивать, объединять…

IDE - Visual Studio

- Профессиональные и бесплатные версии
  - Для AutoCAD R18 (2010/2011/2012): Visual Studio 2008 SP1 и 2010 (с некоторыми ограничениями)

- Интелисенс
  - Поможет вам с «автозавершением» кода

Ключевые моменты

- .NET используется во многих продуктах Autodesk
  - Мощный и простой в изучении
- Мы обычно используем основные типы и WinForms
- Ссылаемся на другие сборки для доступа к новым функциям
- IDE - Visual Studio
  - Express edition - бесплатный

Повестка дня

- Обзор .NET
- Основы написания приложений AutoCAD .NET API
- Взаимодействие с пользователем
- Основы базы данных
- Словари
- Пользовательский интерфейс
- События
- Input PointMonitor
- Jigs

Основы приложений: Организация кода

- Пространства имен - Namespaces
- Методы и свойства
- Советы и рекомендации
- Параметры и возвращаемые значения

Пространства имён

- Группируют объекты по особенностям
  - Пример:
    Пространство имён **Autodesk.AutoCAD.Geometry** группирует всё, что относится к геометрии в AutoCAD

- Определяют полное квалификационное имя объекта
  - Пример:
    Autodesk.AutoCAD.EditorInput.Editor. WriteMessage

- Использование ключевого слова **imports** позволяет сократить код программы
  - Пример:
    Imports Autodesk.AutoCAD.EditorInput
    Тогда можно писать короче: **Editor.WriteMessage**

Методы

- Параметры – некоторые методы требуют данные
- Возвращаемое значение – мы можете использовать возвращаемое значение
  - Для примера – сравнить или присвоить другой переменной/объекту

Вызов методов

- Некоторые методы выполняют действия – просто вызовем:
  - line.**DrawOnScreen**()

- Если методу требуются данные – их нужно создать до вызова метода:
  - myInteger = 10
    line.**Resize**(myInteger)

- Или передать их непосредственно:
  - line.**Resize**(10)

- Некоторые методы возвращают значения
  - myInteger = spatialPoint.**CalculateDistanceTo**(anotherPoint)

Свойства

- Это методы, которые не требуют параметров
- Обычно не выполняют действий
- Всегда возвращают значение

Методы и свойства обектов

- Методы выполняют действие только в контексте объекта
  - line.**Resize**(10)
    Только <u>этот</u> отрезок изменит длину, а не другие
- Свойства возвращают значения, которые определяются объектом
  - myInteger = line.**Length**
    Длина только <u>этого</u> отрезка

Методы или Свойства

- Методы – они как глаголы
  - Определяют действие
  - Выполняют задачу каждый раз, когда их вызывают

- Свойства – они как существительные или прилагательные
  - Обращаются к характеристикам

Рекомендации по созданию Методов

- Используйте имена имеющие смысл:
  - Вместо CrNLine, используйте CreateNewLine
- Отдавайте предпочтению созданию небольших методов, которые выполняют одну задачу:
  - Задача должна соответствовать названию метода

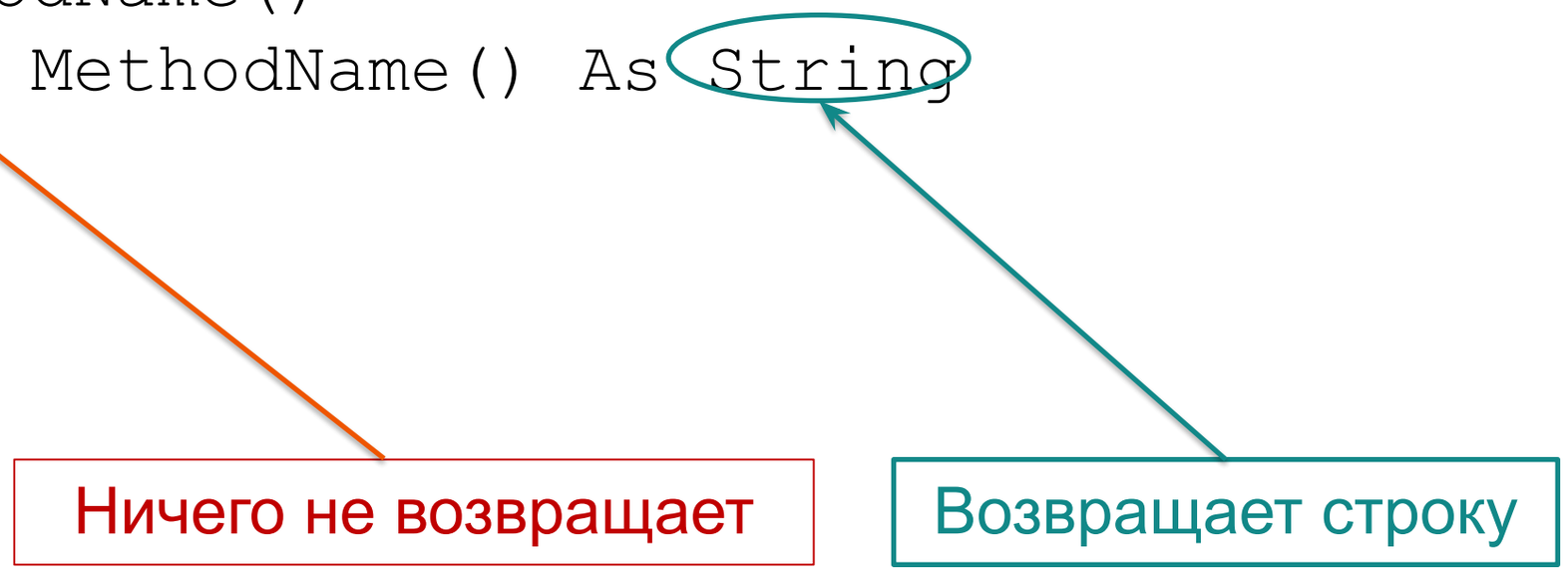
Создание Метода

- Сначала объявите уровень доступа
  - **Public**: доступен снаружи, требуется для создания команд
  - **Private**: доступен только изнутри, обычно используется для внутренних методов
- Имя метода не может содержать спецсимволов или пробелов и не может начинаться с цифры

Возвращаемое значение

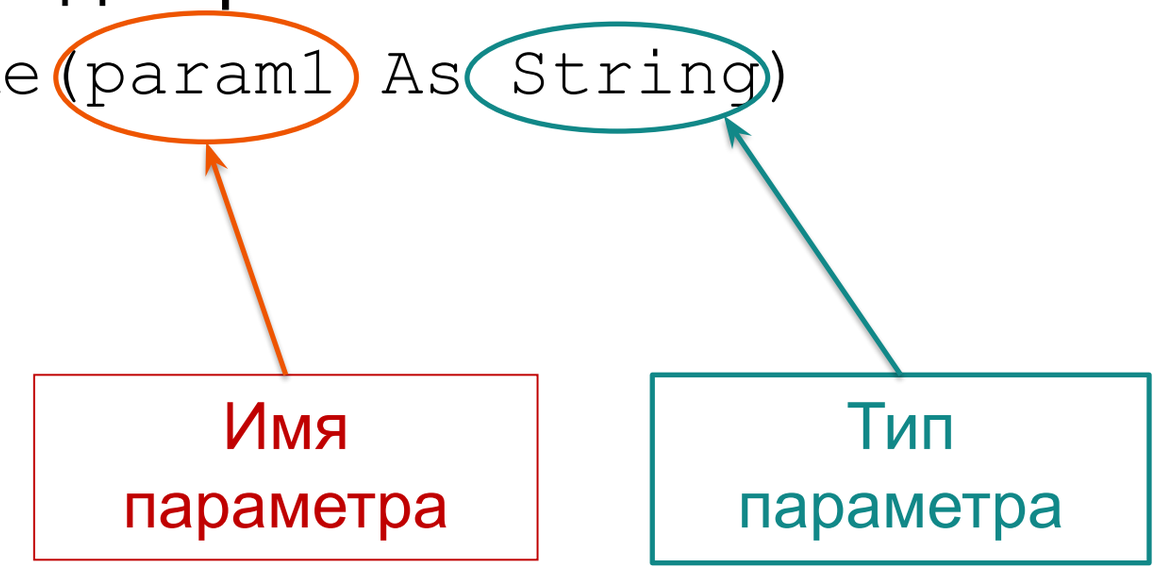
- Возвращаемое значение
  - **Sub** ничего не возвращает, а **Function** возвращает значение

```
Public Sub MethodName()
Public Function MethodName() As String
```



Параметры

- Имя параметра – это имя переменной
  - Вы можете использовать эту переменную внутри метода
- Может получать значение для работы с ним
  - `Public Sub MethodName(param1 As String)`



Пример метода

- Перемножение чисел
  - ```
    Public Function Multiply(number1 As Integer, _
                             number2 as Integer) as Integer
        Return number1 * number2
    End Sub
    ```

Указывает на переход на следующую строку
Обратите внимание на предваряющий пробел
    ```

Ключевые моменты

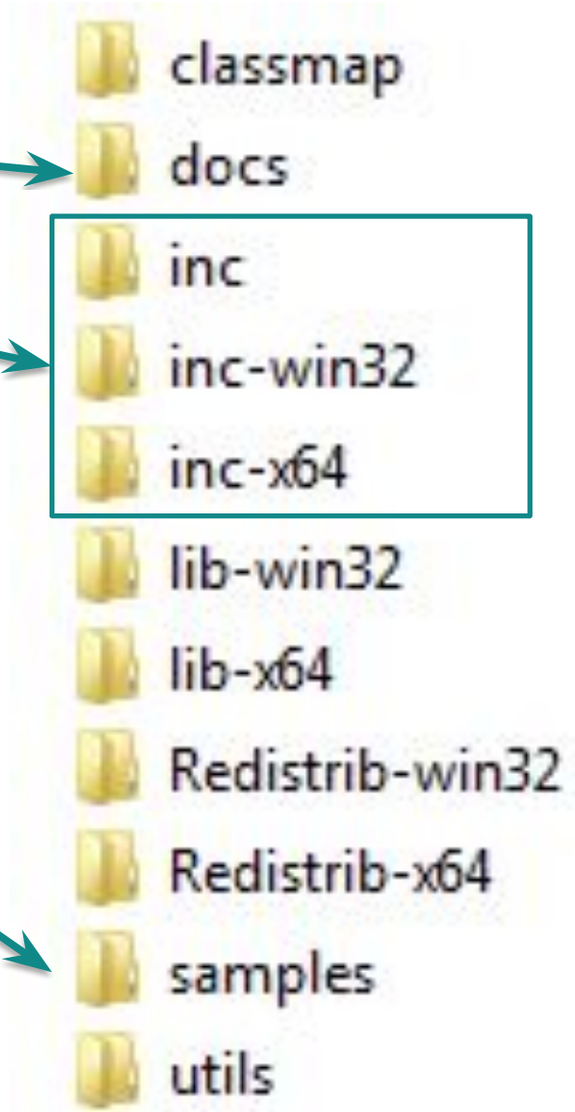
- Методы могут получать параметры и возвращать значение

- Свойства всегда возвращают значение

- Эффективная работа:
  - Методы - глаголы
  - Свойства – существительные/прилагательные

Подготовка первого приложения: ObjectARX SDK

- ObjectARX SDK
  - Файлы помощи
  - Файлы поддержки
  - Примеры



Как работают приложения в AutoCAD?

Проект VB.NET

```
Imports Autodesk.AutoCAD.Runtime
Imports Autodesk.AutoCAD.ApplicationServices
Imports Autodesk.AutoCAD.Interop

Public Class Commands
    Implements IExtensionApplication

    Private WithEvents docs As DocumentCollection

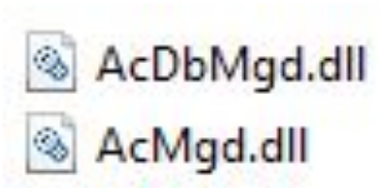
    Private Sub docs_DocumentActivated(ByVal sender As Object, ByVal e As Autodesk.AutoCAD.Ap
        ThisDrawing = e.Document.AcadDocument
    End Sub

    Private Sub ThisDrawing_BeginDocClose(ByRef Cancel As Boolean) Handles ThisDrawing.BeginI
        ThisDrawing = Nothing
    End Sub

    Private Sub ThisDrawing_EndSave(ByVal FileName As String) Handles ThisDrawing.EndSave
        MsgBox("EndSave called")
    End Sub
End Class
```

Код на Visual Basic .NET

Ссылаемся на DLL-файлы AutoCAD. Используем их из каталога ObjectARX INC



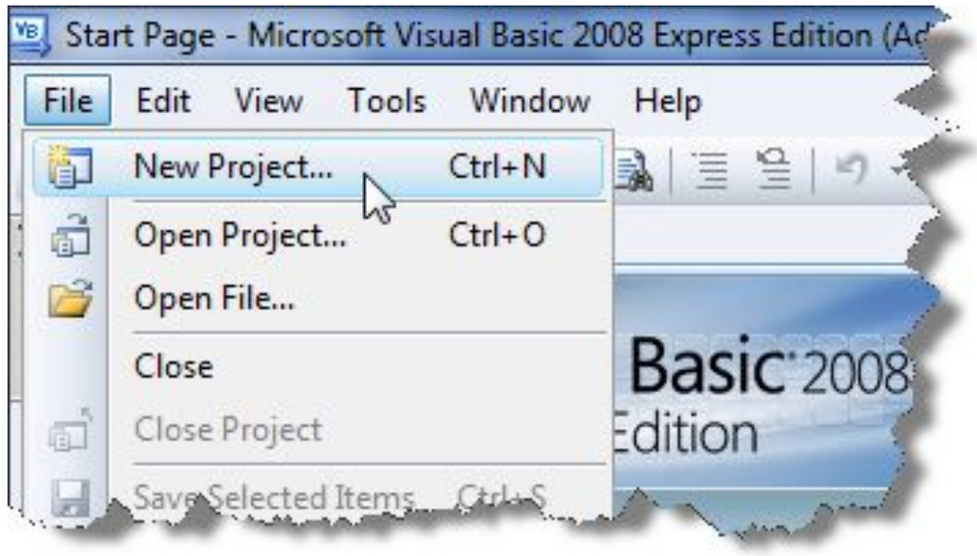
Компиляция

Сборка (.dll)

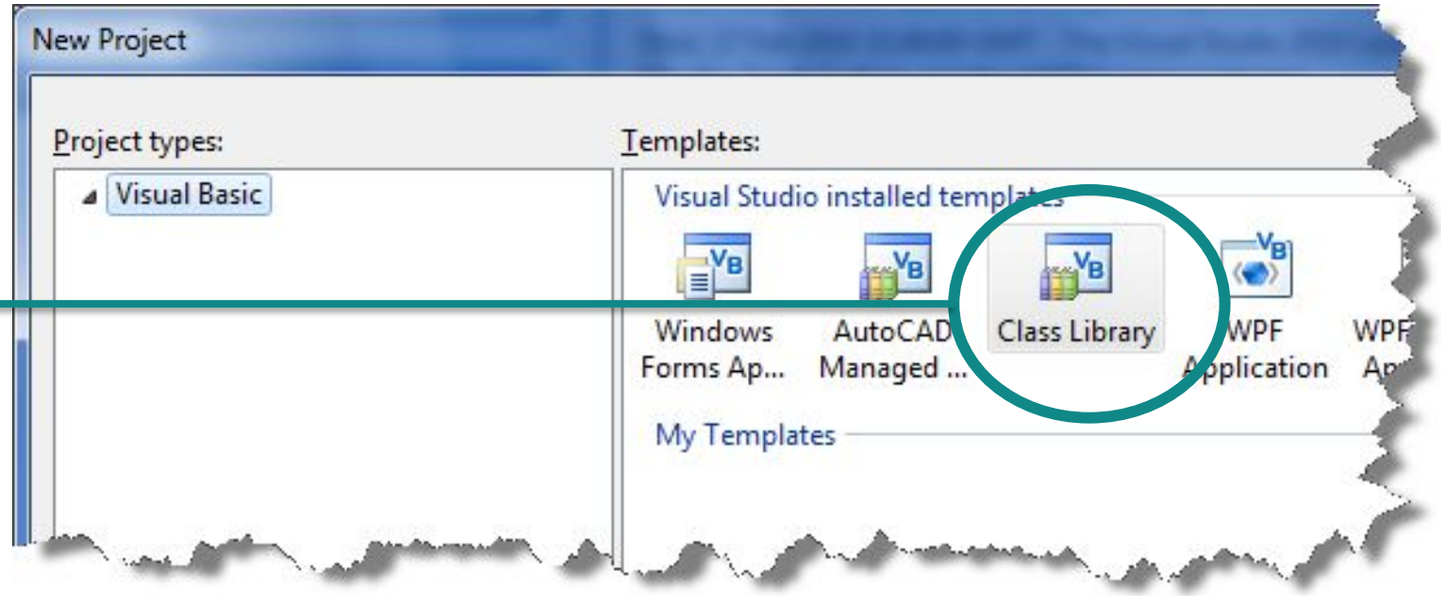
Загружаем внутрь AutoCAD командой NETLOAD

Создаем новый проект VB.NET

- Создаем новый проект

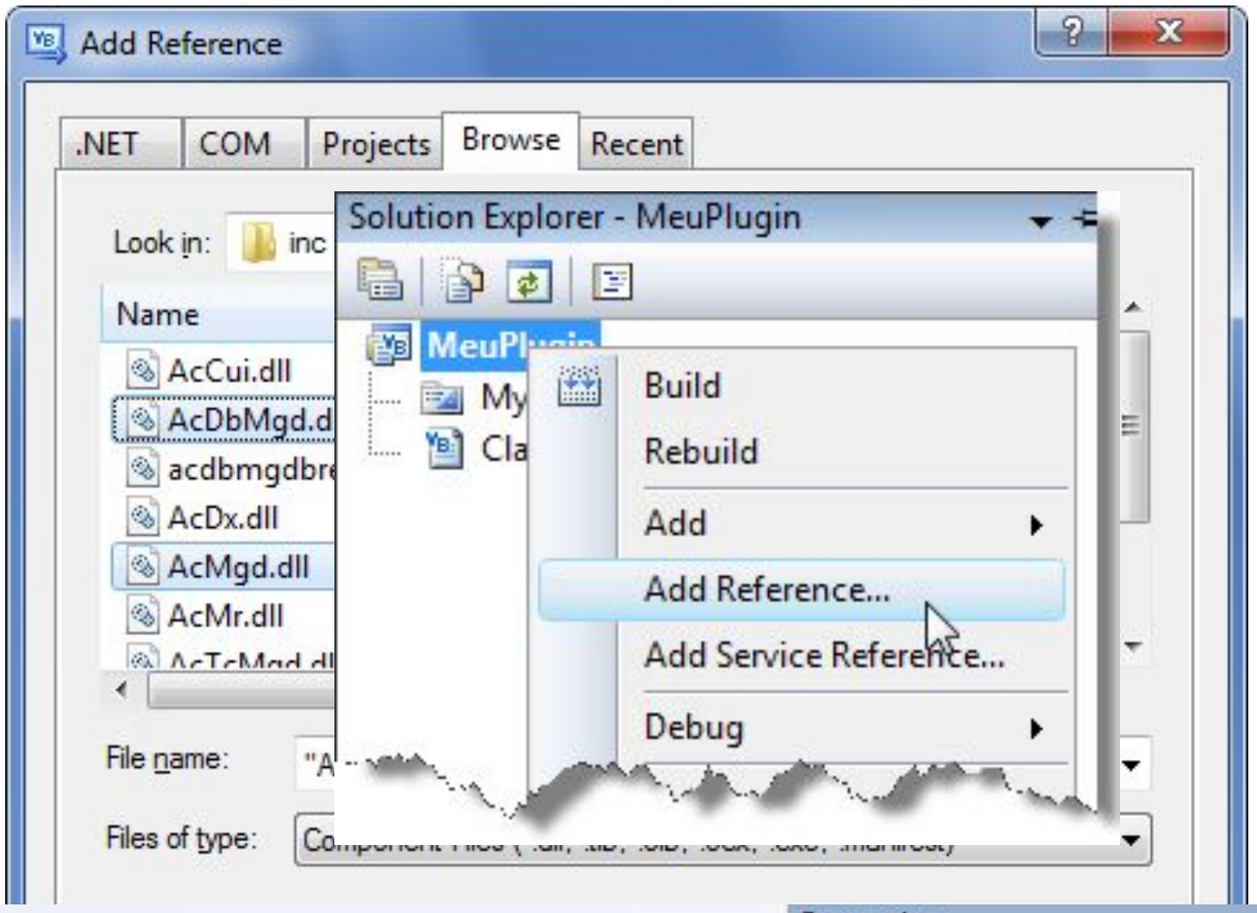


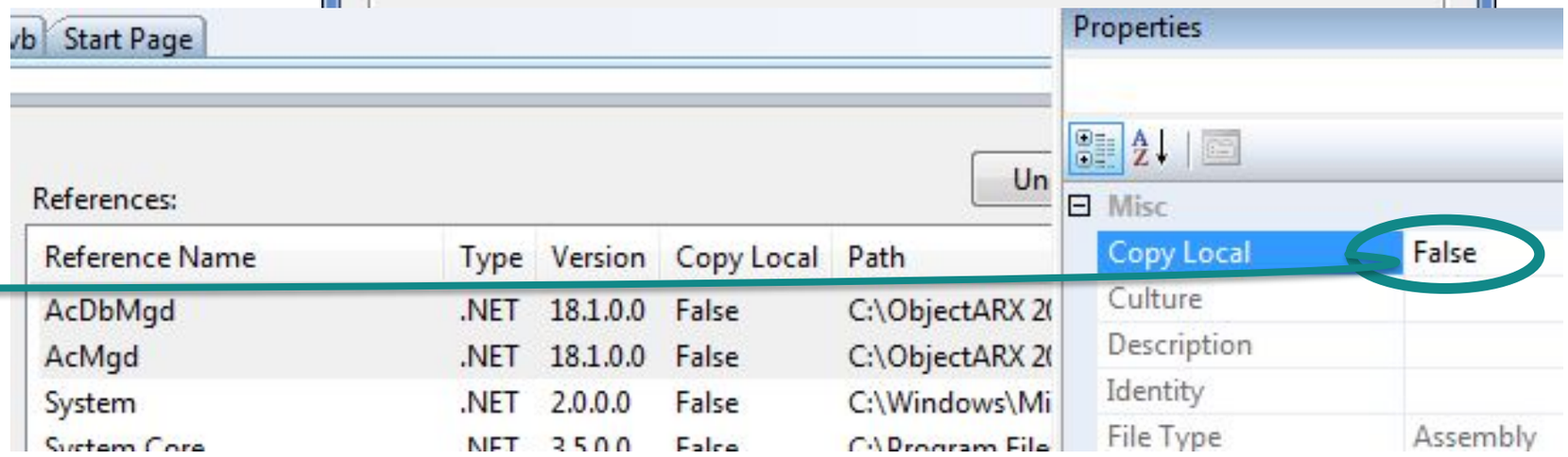
- Приложение для AutoCAD должно быть типа *Class Library*



Добавление ссылки на AutoCAD

- Добавление ссылок
- **AcMdg**
  Ресурсы интерфейса
  C:\ObjectARX 2011\inc\AcMgd.dll
- **AcDbMgd**
  Ресурсы базы данных
  C:\ObjectARX 2011\inc\AcDbMgd.dll
- ВАЖНО:
  Установить
  Copy Local
  в **FALSE**





Процедура как пользовательская команда

1. Обычная VB процедура
2. AutoCAD *Imports*
3. Процедура как команда
4. Доступ к редактору
5. Печать сообщения

```
Imports Autodesk.AutoCAD.Runtime
Imports Autodesk.AutoCAD.ApplicationServices
Imports Autodesk.AutoCAD.EditorInput

Public Class Class1

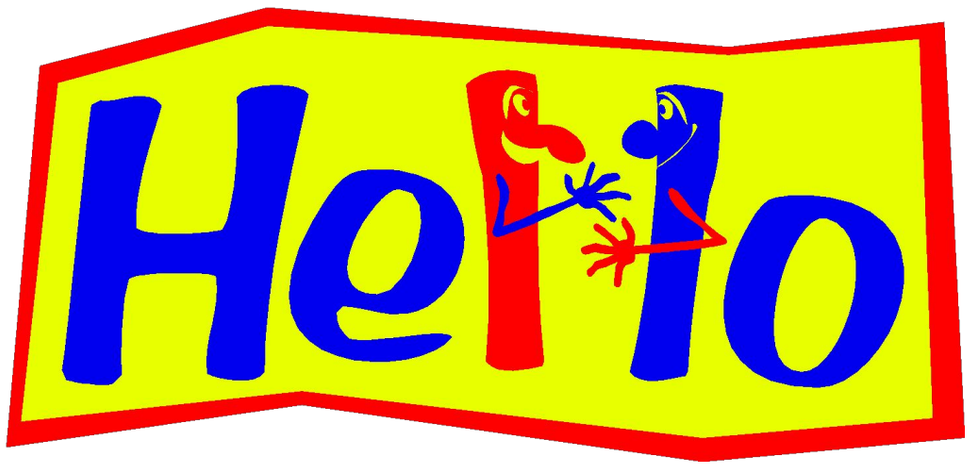
    <CommandMethod("myCommand")> _
    Public Sub myRoutine()

        'Доступ к редактору AutoCAD
        Dim ed As Editor = Application. _
                           DocumentManager. _
                           MdiActiveDocument.
                           _
                           Editor
        'Печатаем простое сообщение
        ed.WriteMessage("Здравствуй, Мир! ")
    End Sub

End Class
```

Теперь компилируем, загружаем внутрь AutoCAD командой NETLOAD, и вызываем **myCommand**

Lab 1 – Здравствуй, Мир!



NETLOAD или ключи реестра

HKEY_CURRENT_USER — Для текущего пользователя

HKEY_LOCAL_MACHINE — Для всех пользователей

SOFTWARE

Autodesk — R17.0: 2007
.1: 2008
.2: 2009
R18.0: 2010
.1: 2011
.2: 2012

AutoCAD

R18.1

ACAD-9001:409 — X000: Civil3D, X001: AutoCAD; 409: Английский, 419: Русский

Applications

YourAppName

```
"DESCRIPTION"="Custom App Name"
"LOADER"="C:\\folder\\appName.dll"
"LOADCTRLS"=dword:0000000e
"MANAGED"=dword:00000001
```

Object ARX Wizards

- Модуль расширения
  - <<ObjectARX SDK folder>>\utils\ObjARXWiz\ArxWizards.msi
- Шаблоны для VB.NET или C# приложений
- Обсуждение и актуальные версии в блоге
  - http://blogs.autodesk.com/through-the-interface

Средства отладки .NET

- Reflector
  - Просмотр .NET сборок, дисассемблирование, декомпиляция
    - http://sharptoolbox.madgeek.com
- Ildasm
  - дисассемблирование .NET сборок
    - В составе инструментов Visual Studio
- Fuslogv
  - Диагностика проблем при загрузке
    - В составе инструментов Visual Studio
- FxCop
  - Проверка соответствия рекомендациям по проектированию
    - http://www.gotdotnet.com/team/fxcop/

Повестка дня

- Обзор .NET
- Основы написания приложений AutoCAD .NET API
- Взаимодействие с пользователем
- Основы базы данных
- Словари
- Пользовательский интерфейс
- События
- Input PointMonitor
- Jigs

Запрос ввода у пользователя

- Используем Prompt***XXX***Options для установки параметров запроса
  - ***XXX*** – это тип значения, которое мы хотим запросить – Angle (угол), String (строка), Distance (расстояние)
  - Используем свойства **Message** и **Keywords**
  - Используем Allow***YYY*** для создания условий для запроса. *Например, AllowNegative – запрещает ввод отрицательных*

- Для запроса используем функции Ge**tXXX** класса Editor
  - Примеры - GetAngle, GetString, GetDistance
  - Передаём Prompt***XXX***Options методам Editor.Get***XXX***

- Результат помещается в PromptResult или его наследники
  - Примеры – PromptDoubleResult, PromptIntegerResult и т.д.

Запросим точку на экране

- Конфигурируем опции для выбора точки на экране

```
Dim pointOptions As New PromptPointOptions("Укажите точку: ")
```

- Запрашиваем у пользователя точку и сохраняем результат указания

```
Dim pointResult As PromptPointResult = ed.GetPoint(pointOptions)
```

- Создаём переменную класса Point3d для сохранения выбранной точки
  - Требуется дополнительный *imports* для Point3d: **Autodesk.AutoCAD.Geometry**

```
Dim selectedPoint As Point3d = pointResult.Value
```

- Печатаем координаты точки (XYZ)

```
ed.WriteMessage(selectedPoint.ToString())
```

Еще о пользовательском вводе

- Prompts**XXX**Options используются для управления запросами Например, так:
- Установим *Message* (сообщение)
  - Введите число сторон:
- Установим *Keywords* (ключевые слова)
  - Введите число сторон [Треугольник/Квадрат/Пентагон] :
- Установим *Defaults* (значение по умолчанию)
  - Введите число сторон [Треугольник/Квадрат/Пентагон] <3>:
- Установим *Allowed* (допустимые) значения
  - Введите число сторон [Треугольник/Квадрат/Пентагон] <3>: -5
    - Значение должно быть положительным и ненулевым.
- Prompt**XXX**Result используется для получения результата запроса

Lab 2 – Пользовательский ввод

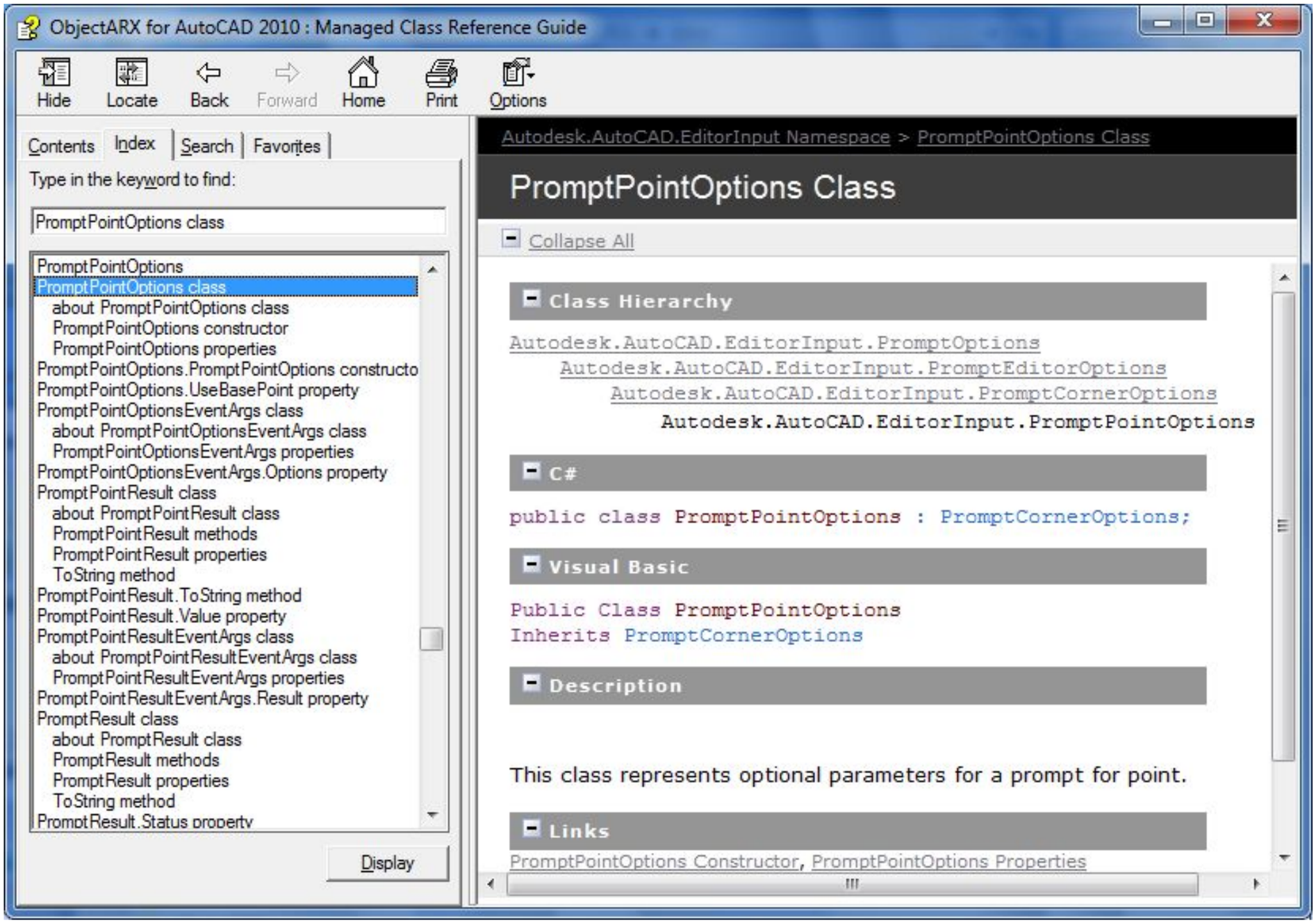
Дополнительные запросы

- Типы:
  - PromptPointOptions
  - PromptStringOptions
  - PromptDoubleOptions
  - PromptAngleOptions
  - PromptCornerOptions
  - PromptDistanceOptions
  - PromptEntityOptions
  - PromptIntegerOptions
  - PromptKeywordOptions
  - PromptNestedEntityOptions
  - PromptNestedEntityOptions
  - PromptSelectionOptions
  - И т.д.

Файл помощи поможет!

Файл помощи: Справочное руководство

- <<ObjectARX SDK folder>>\docs\arxmgd.chm



Повестка дня

- Обзор .NET
- Основы написания приложений AutoCAD .NET API
- Взаимодействие с пользователем
- Основы базы данных
- Словари
- Пользовательский интерфейс
- События
- Input PointMonitor
- Jigs

AutoCAD DWG как база данных

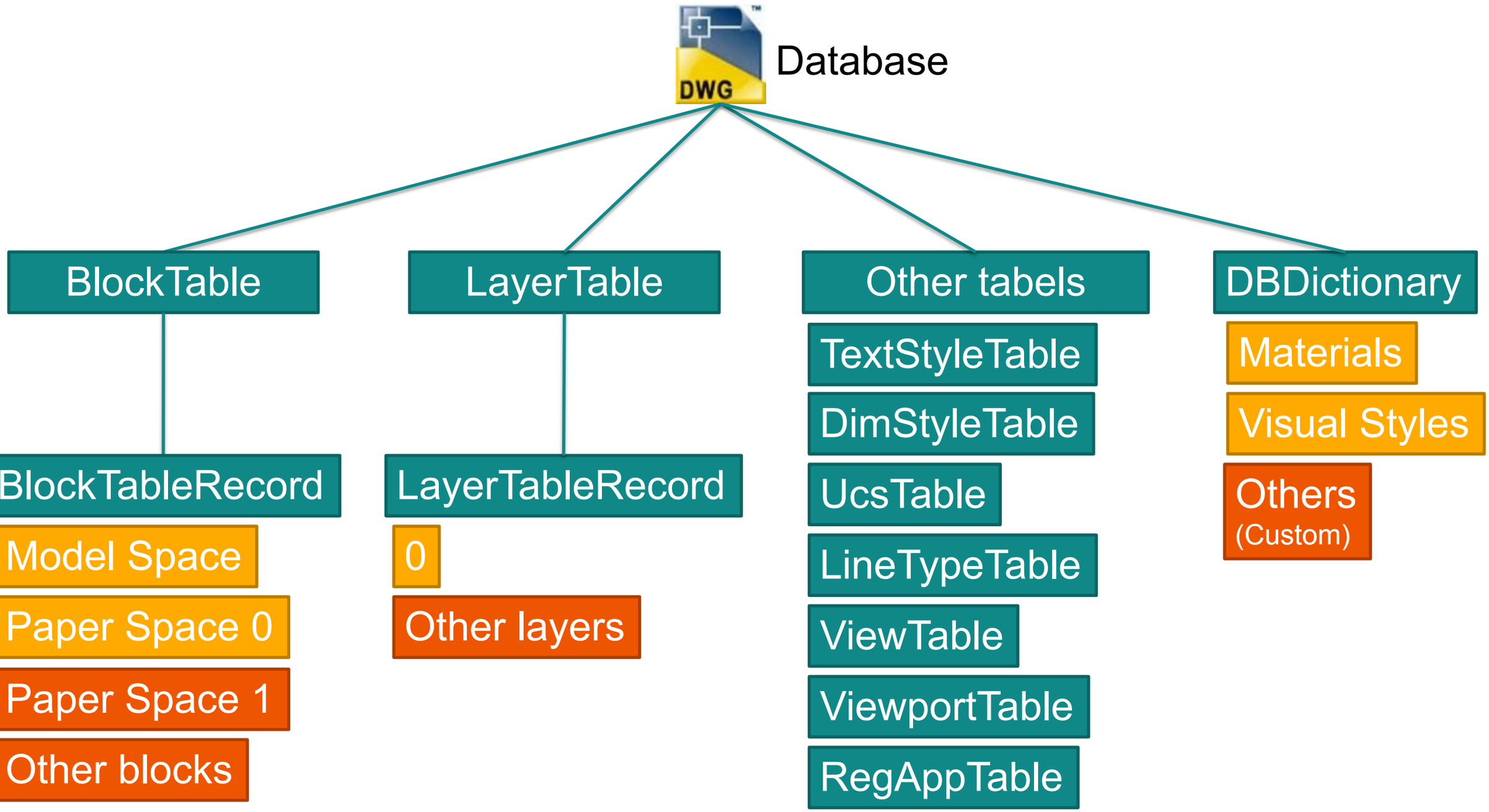
- Представление DWG файла в памяти
- Объекты помещаются в иерархическую базу данных – Db структура
- Все объекты имеют идентификаторы – **ObjectId** аналог первичного ключа в реляционной базе данных
- Объекты могут ссылаться друг на друга – как отрезок ссылается на слой

Первичный ключ базы: Идентификатор объекта (ObjectID)

- Все объекты, которые есть в базе, имеют **ObjectId**
  - Уникальны в течении сессии AutoCAD
  - Не постоянны (при следующем запуске AutoCAD - другие)
  - Генерируются автоматически для каждого объекта при добавлении в базу
  - Для не находящихся в базе объектов ObjectId не установлен
  - Получаем посредством свойства ObjectId

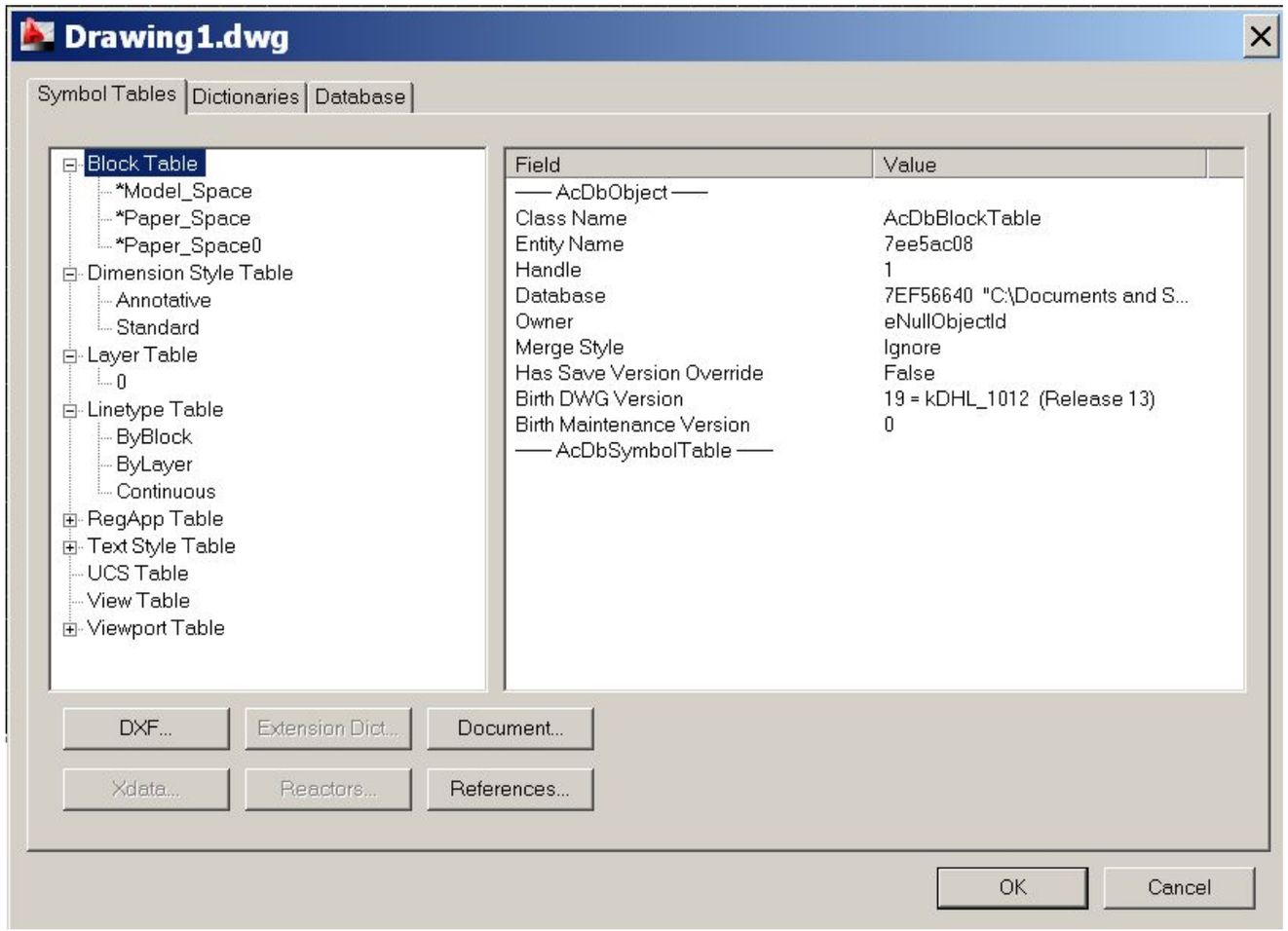
- Все объекты, которые есть в базе, имеют Handle
  - *Не уникальны* в течении сессии (в разных dwg-файлах)
  - Постоянны (с некоторыми оговорками)
  - Получаем посредством свойства Handle

Структура базы: Обзор



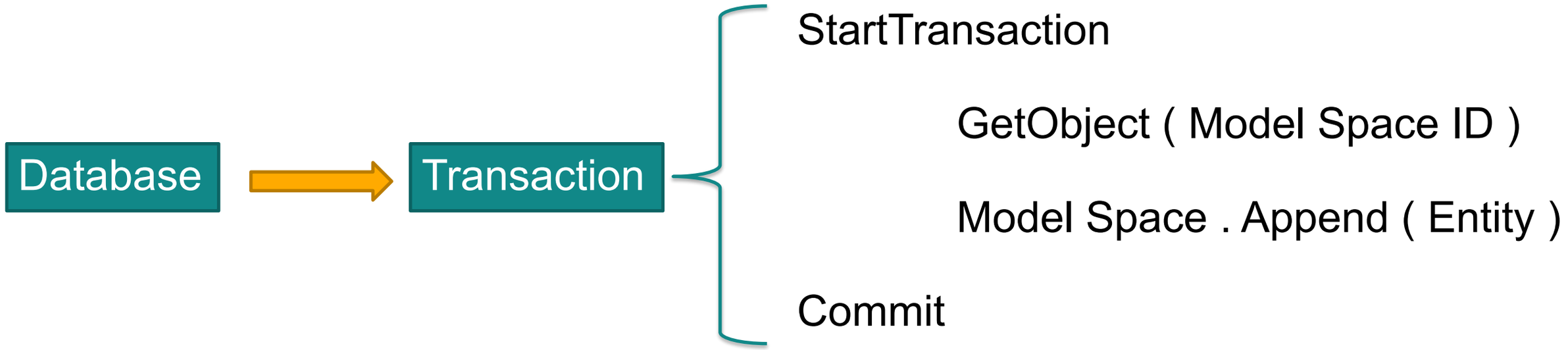
Шпионские инструменты (для базы AutoCAD)

- ArxDbg (C++) ObjectARX SDK
- MgdDbg(C#) ADN



Транзакции: Обзор

- Любая операция чтения/записи должна осуществляться внутри транзакции
- AutoCAD никогда не возвращает объект/примитив непосредственно, только ObjectId
- Используем ObjectId в транзакции (класс Transaction) метод GetObject
- Каждая внешняя транзакция соответствует операции UNDO



Метод Transaction.GetObject

- Используем GetObject для открытия объекта
  - Transaction.GetObject(ObjectId , OpeningMode)

- ObjectId: получается различными путями

- Метод открытия (OpeningMode):
  - **ForRead**: только для чтения информации, быстрее
  - **ForWrite**: для чтения и записи информации, обеспечивает UNDO механизм

- Рекомендуемые действия:
  - Открыть **ForRead**, затем при необходимости, **UpgradeOpen** для **ForWrite**
  - Открыть **ForWrite,** если Вы будете писать в любом случае

Использование транзакций

- Сначала получаем доступ к Database
- Используем ключевое слово **Using** для освобождения (завершения) транзакции
- Открываем объект используя его objectId

```
‘Доступ к базе (Database)
Dim db As Database = Application.DocumentManager.MdiActiveDocument.Database

‘Начинаем транзакцию
Using trans As Transaction = db.TransactionManager.StartTransaction

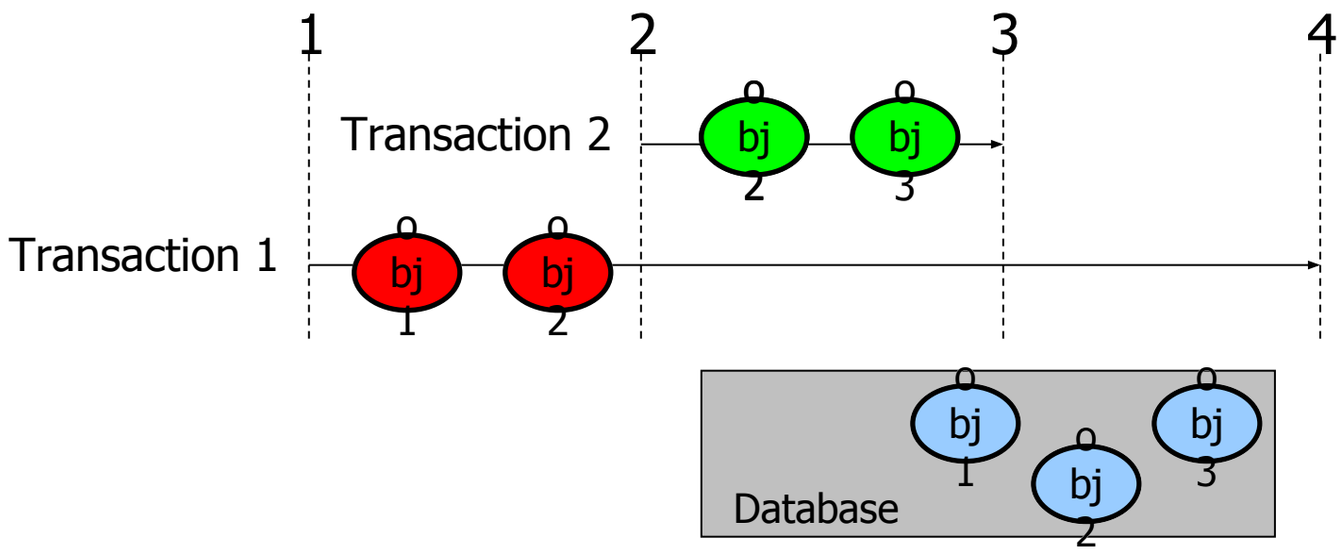
    ‘Получаем таблицу блоков (BlockTable)
    Dim bt As BlockTable = trans.GetObject(db.BlockTableId, OpenMode.ForRead)

End Using ‘Освобождаем транзакцию
```

Транзакции: Границы

- Могут быть:
  - Завершены: все операции сохранены - Transaction.Commit()
  - Прерваны: все операции отменены – Transaction.Abort()

- Могут быть вложены:
  - Внешняя управляет всем, т.е. если она прервана, то и вложенные тоже.



Компоненты базы: Символьные таблицы

- Сивольные таблицы: BlockTable, LayerTable и другие

- Символьные таблицы
  - Контейнеры, содержащие *записи* (Symbol Table *Records)*
  - Пример: **BlockTable*Record***, **LayerTable*Record***

- У всех символьных таблиц есть общие методы контейнеров, такие как
  - **Add** – добавить запись
  - **Item** – найти запись по имени
  - **Has** – узнать есть ли запись с таким именем

- Перечисляемые – Итерация посредством ‘For Each’

Использование транзакций: создание определения блока

```
Dim db As Database = Application.DocumentManager.MdiActiveDocument.Database
Using trans As Transaction = db.TransactionManager.StartTransaction
    Dim bt As BlockTable = trans.GetObject(db.BlockTableId, OpenMode.ForRead)

    'Проверяем есть ли уже блок 'myBlock'
    If Not (bt.Has("myBlock")) Then

        'Создаём новый блок
        Dim myBlock As New BlockTableRecord
        myBlock.Name = "myBlock"

        'Переключаем таблицу блоков в режим
        bt.UpgradeOpen()

        'Добавляем новый блок к таблице блок
        bt.Add(myBlock)
        'Информируем транзакцию о новых объе
trans. AddNewlyCreatedDBObject(myBlock, True)
    End If
    'Сохраняем изменения в базе
    trans.Commit()
End Using 'Освобождаем (удаляем) тр
```

Как на предыдущем слайде

Проверяем есть ли уже такой блок

Создаём новый блок (в памяти)

Переключаем таблицу блоков в режим ForWrite

Добавляем новый блок к таблице блоков. Теперь он в базе данных.

ВСЕГДА информируем транзакцию о новых объектах

Сохраняем изменения
Подсказка: Commit быстрее чем Abort

Использование транзакций: список всех записей таблицы блоков

- Выведем список имён всех BTR, включая Пространство Модели, все Пространства Листа и все другие блоки (встроенные/анонимные и созданные пользователем)

```
Dim db As Database = Application.DocumentManager.MdiActiveDocument.Database
Using trans As Transaction = db.TransactionManager.StartTransaction
    Dim bt As BlockTable = trans.GetObject(db.BlockTableId, OpenMode.ForRead)

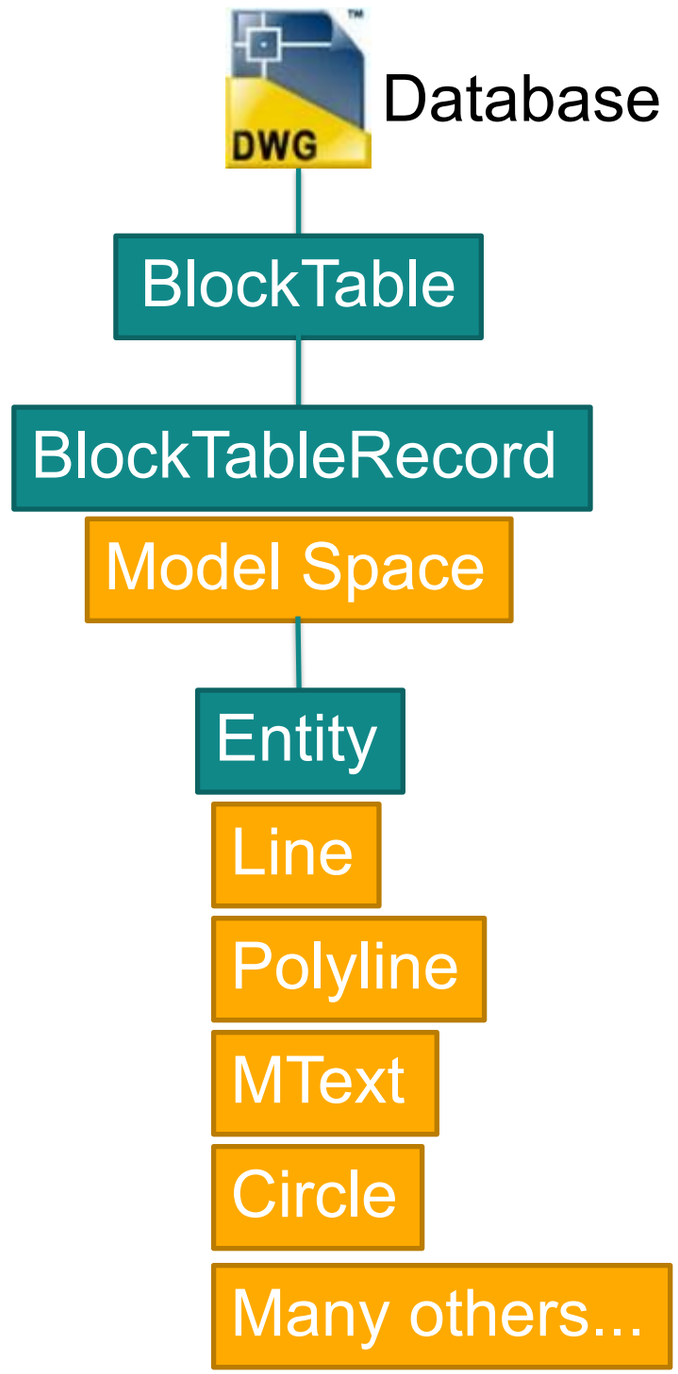
    'Проходимся по всем записям таблицы блоков (BTR)
    For Each btrId As ObjectId In bt

        'Открываем каждую запись таблицы блоков (BTR)
        Dim btr As BlockTableRecord = trans.GetObject(btrId, OpenMode.ForRead)

        'Печатаем имя блока
        ed.WriteMessage(btr.Name)
    Next
End Using
```

Структура базы: Пространство Модели

- В таблице блоков
- Пространство Модели – это запись таблицы блоков (BTR)
  - Это также относится к Пространствам Листов и другим блокам
- Каждая BTR содержит примитивы (Entities)
- Один тип примитива для каждого графического типа
- Перечисляемые – Итерация посредством ‘For Each’



Добавление в Пространство Модели

```
Dim db As Database = Application.DocumentManager.MdiActiveDocument.Database
Using trans As Transaction = db.TransactionManager.StartTransaction

    'Открываем текущее пространство (любая BTR, обычно пространство модели)
    Dim mSpace As BlockTableRecord = trans.GetObject(db.CurrentSpaceId, _
                                                     OpenMode.ForWrite)

    'Создаём и настраиваем отрезок
    Dim newLine As New Line
    newLine.StartPoint = New Point3d(0, 0, 0)
    newLine.EndPoint = New Point3d(10, 10, 0)

    'Добавляем отрезок к текущему пространству
    mSpace.AppendEntity(newLine)

    'Информирует транзакцию о новых объектах
    trans.AddNewlyCreatedDBObject(newLine, True)

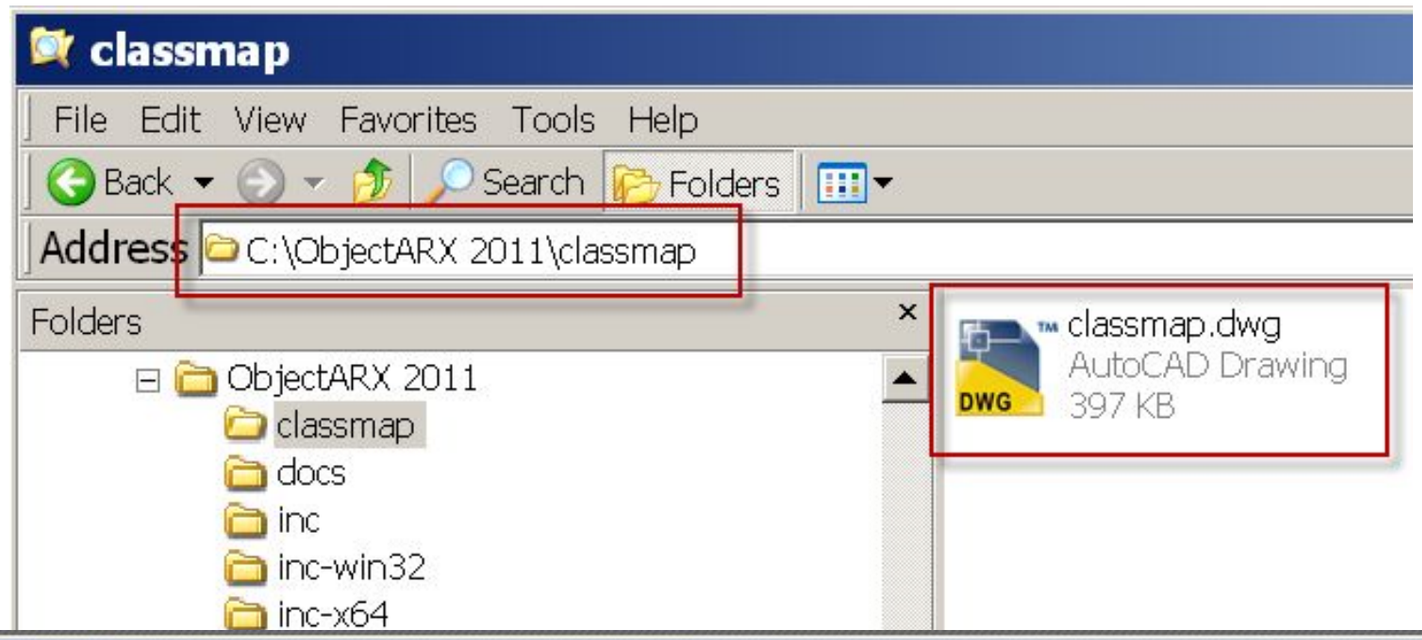
End Using 'Освобождаем транзакцию
```

Открываем текущее пространство для

Создаём и настраиваем отрезок (в памяти)

Добавляем к текущему пространству. Теперь отрезок в базе данных

Компоненты базы: Дерево классов



Управление памятью объектов

- Управляемые объекты – суть *обертки* для неуправляемых объектов C++!
- Т.е. если мы их создаём при помощи New, должны ли мы их удалять?
  - Нет – сборка мусора освобождает объекты когда они становятся не нужны, для того чтобы вернуть память
    - Если объект не в базе – удаляется (delete) и соответствующий неуправляемый объект
    - Если объект в базе – закрывается (Close) и соответствующий неуправляемый объект
- Если объект в транзакции, удаление или завершение транзакции закрывает его автоматически!
- Ручная очистка требуется только в ряде случаев, например, Transaction (ключевое слово Using)

Ошибки или Исключения

- Генерация ошибки/исключения, когда не может выполнится строка кода
- Несколько различных причин, иногда непредсказуемых
- Основное приложение Autodesk обычно генерирует обрабатываемые исключения
  - Т.е. неправильная переменная, данные или контекст

Обработка исключений

- .NET приложение может обработать **НЕ ВСЕ** исключения
  - Особенно те, которые возникают в базом приложении, не могут быть обработаны в Вашем приложении

- Обычно мы можем обрабатывать исключения уровня приложения

- Хороший метод – попробовать что-то, в чем мы не уверены, что оно работает правильно, – часто используется в наших уроках

Будем бесстрашны: Попробуем (Try)

- VB.NET

```
Try
    'Попробуем здесь что-
нибудь
Catch

End Try
```

Требуется
Больше на следующем слайде

Попробуем какой-то код внутри TRY

Исследуем: Catch

- VB.NET

```
Try

Catch (ex As Exception)
    'Что-то пошло не так
-
    'попробуем исправить
End Try
```

Если что-то происходит не так в блоке TRY, тогда выполнение прекращается и переходит в CATCH

Необязательное
Переменная **ex** содержит доступную информацию для понимания исключения

Будем организованы: Finally

- VB.NET

```
Try

Catch

Finally
    'Здесь очистка
End Try
```

Необязательное

Мы обычно чистим переменные или закрываем соединение. Этот блок выполняется **всегда**, в не зависимости от того было ли исключение или нет

Итоги Try/Catch/Finally

- Для обработки исключений, требуются try и catch
  - ex необязательный параметр
  - Finally необязательный блок

- Примеры:
  - Пробуем (**Try**) получить коллекцию, если не существует - создадим её внутри ловушки (**Catch)**
  - Открываем документ, пробуем (**Try)** изменить его, отлавливаем (**Catch)** если он только для чтения, в завершении (**Finally)** закрываем его (и если всё нормально, и если ошибка)

Транзакции - повторно используемый код

- Перетаскиваем (Drag’n’drop) в Visual Studio Toolbox

```
Dim db As Database = Application.DocumentManager.MdiActiveDocument.Database
Using trans As Transaction = db.TransactionManager.StartTransaction
    Try
        trans.Commit()
    Catch ex As Exception

        trans.Abort()
    End Try
End Using
```

Lab 3 – Создание примитива, блока и вставки блока

Повестка дня

- Обзор .NET
- Основы написания приложений AutoCAD .NET API
- Взаимодействие с пользователем
- Основы базы данных
- Словари
- Пользовательский интерфейс
- События
- Input PointMonitor
- Jigs

Словари

- Словари (Тип **DbDictionary**)
  - Контейнер для хранения только данных
  - Хранятся другие **Словари**
  - Хранятся **неграфические** Объекты (наследуемые от DbObject, **но не от** Entity!)
  - Перечисляемый
    - Каждый элемент имеет строковый ключ
    - Элементы можно получить по строковому ключу при помощи метода GetAt() или свойства Item

Named Object Dic. и Extension Dic.

- Два ***корневых*** словаря

- Словарь именованных объектов (NOD)
  - Владелец - база
  - Всегда присутствует в базе
  - Используется для хранения данных уровня базы
- Расширенный словарь (Extension Dictionary)
  - Владелец - примитив (Entity)
  - Создаётся пользователем (программой) только при необходимости
  - Используется для хранения данных уровня примитива

- Используем SnoopDb для просмотра содержащихся словарей

Как открыть словарь

- Словарь именованных объектов NOD (уровень базы)

```
' NOD всегда существует, можем просто открыть
Dim nod As DBDictionary = trans.GetObject(db.NamedObjectsDictionaryId, OpenMode.ForRead)
```

- Расширенный словарь Ext. Dic. (уровень примитива)

```
'Расширенного словаря может и не быть для определенного примитива
Dim line As Line = trans.GetObject(lineId, OpenMode.ForRead)

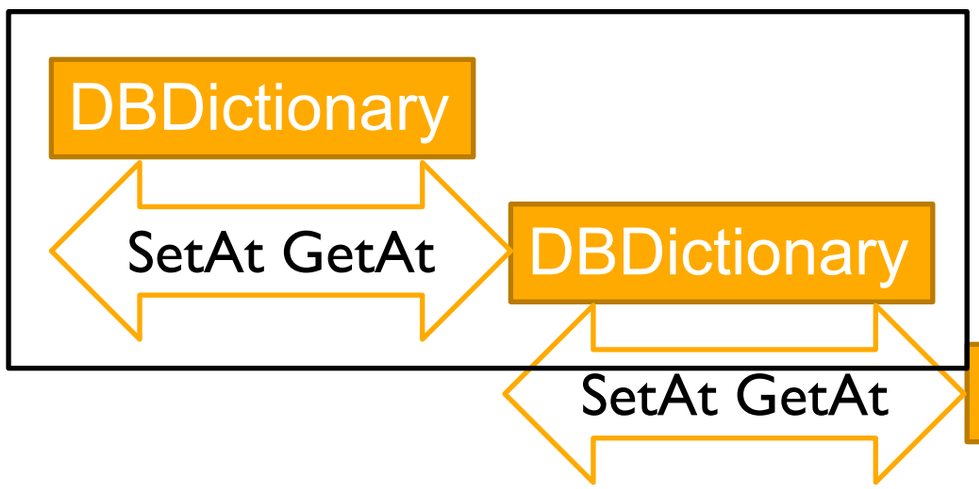
‘Проверим ObjectId на null, создаём если нужно
If (line.ExtensionDictionary.IsNull) Then
    line.CreateExtensionDictionary() 'Создаём Ext. Dic.
End If

'Открываем Ext. Dic.
Dim extDic As DBDictionary = trans.GetObject(line.ExtensionDictionary, OpenMode.ForRead)
```

Иерархия словарей

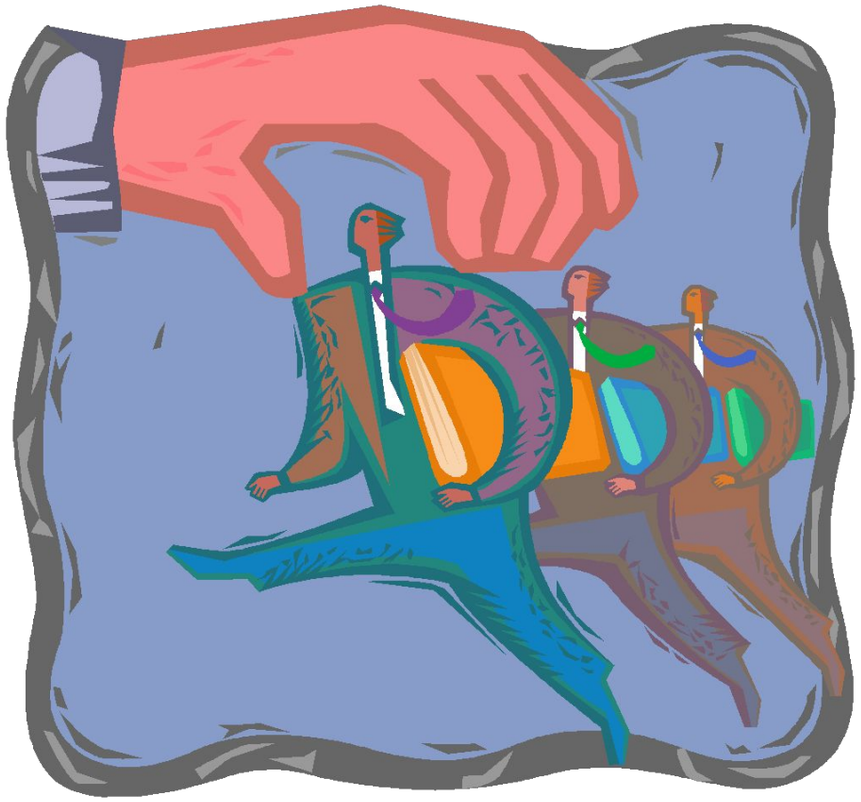
Named Object Dictionary

Extension Dictionary



Один словарь (DBDictionary) может содержать другие. Хорошо бы организовать структуру и предотвратить конфликты с другими приложениями

Lab 5 - Словари



Динамическая идентификация типов

- Зачем нам знать какого типа примитив (Entity)?
- Некоторые коллекции могут содержать объекты различных типов
- Получить тип каждого элемента, а затем решить, что делать
- Это очень распространенная и полезная для создания Autodesk приложений практика

Получить тип переменной

- Каждая переменная имеет метод GetType

```
Dim myString As String
myString = "здесь какой-нибудь текст"

Dim variableType As Type
variableType = myString.GetType()

MessageBox.Show(variableType.ToString())
```

Сравнение типов

- Сравнение типов различных переменных
- Полезно когда мы не знаем точный тип
  - Очень распространено в Autodesk APIs

```
If (someVariable.GetType() Is GetType(String)) Then
 'Теперь мы знаем, что это строка!
End If
```

Преобразование между типами

- Первое и очень важное:
  Мы можем преобразовать далеко не из кадого типа в любой другой - есть ограничения
  - Для .NET типов есть методы преобразования
  - Для AutoCAD .NET API требуется уточнение в дереве классов
- Техническое название: **cast**

Прямое преобразование: Простой путь

- Вы знаете что это возможно, но не уверены

- Часто используется:
  - CInt(variable): преобразование в целое (Integer)
  - CStr для строк, CDbl для плавающих чисел

- Параметром *variable* может быть практически всё, что имеет смысл (строка или число)

Прямое преобразование: Быстрый путь

- Вы уверены что это возможно
- Быстрее чем CInt, CStr, и т.д.
- CType(variable, type)
- Генерируется исключение, если преобразование не возможно

Давайте попробуем, прежде чем преобразовывать

- Удобно если вы не знаете возможно ли это преобразование
  - При неудаче возвращается Nothing (и нет исключения)

- TryCast(variable, type)

```
Dim myString As String
myString = TryCast(someVariable, String)
If (myString IsNot Nothing) Then
    'Ура! Преобразование получилось
End If
```

Ключевые моменты

- GetType возвращает тип любой переменной
- Используйте ключевые слова `Is` или `IsNot` для сравнения типов
- Прямое преобразование при помощи `CInt`, `CStr`, и т.д., и `CType`
- `TryCast` если вы не уверены - сравните результат с `Nothing`

Повестка дня

- Обзор .NET
- Основы написания приложений AutoCAD .NET API
- Взаимодействие с пользователем
- Основы базы данных
- Словари
- Пользовательский интерфейс
- События
- Input PointMonitor
- Jigs

Формы и основы UI

- Основы WinForm API
- Как создать форму
- Наиболее часто используемые контролы
- Ответ на действия пользователя
- Использование форм

Windows® OS основана на окнах

- Всё что есть на экране - это некоторый тип окна
- Вы перемещаете фокус между окнами
  - Окно с фокусом обычно "подсвечивается" и может получать клавиатурный ввод
- Формы - это специализированные окна
  - Могут управлять другими окнами, в этом случае элементами управления

Формы с WinForms API

- Требуется ссылка на System.Windows.dll
  - В шаблоне DevCamp уже эта ссылка есть
- Пространство имён **System.Windows.Forms**
- Основные черты
  - Формы
  - Контролы для форм (кнопки, текстовые поля, комбополя)
  - Готовые формы (диалоги открытия, сохранения, каталоги)
  - Другие (меню, лента, подсказка)

Формы в реальной жизни

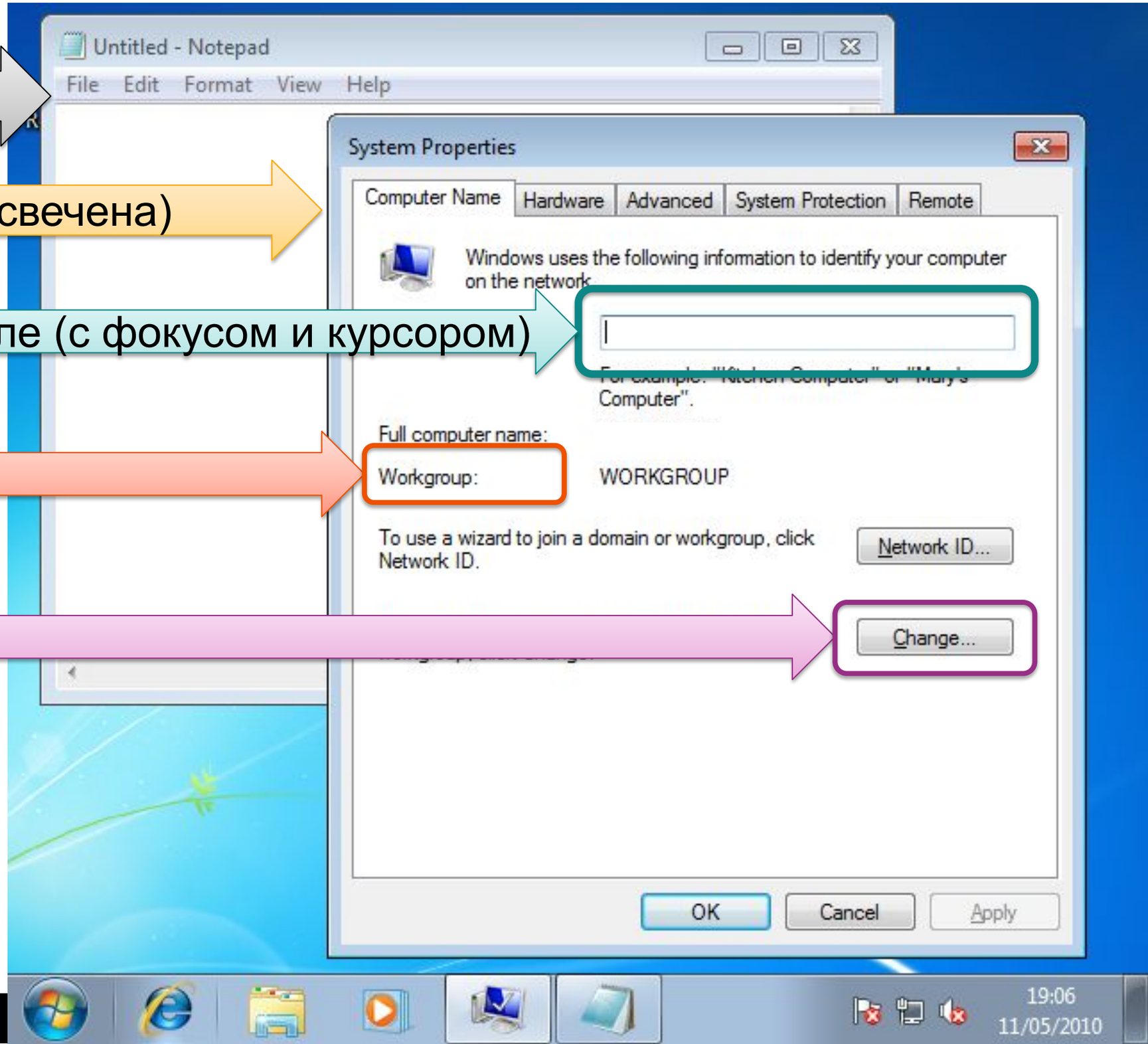
Форма без фокуса

Форма с фокусом (подсвечена)

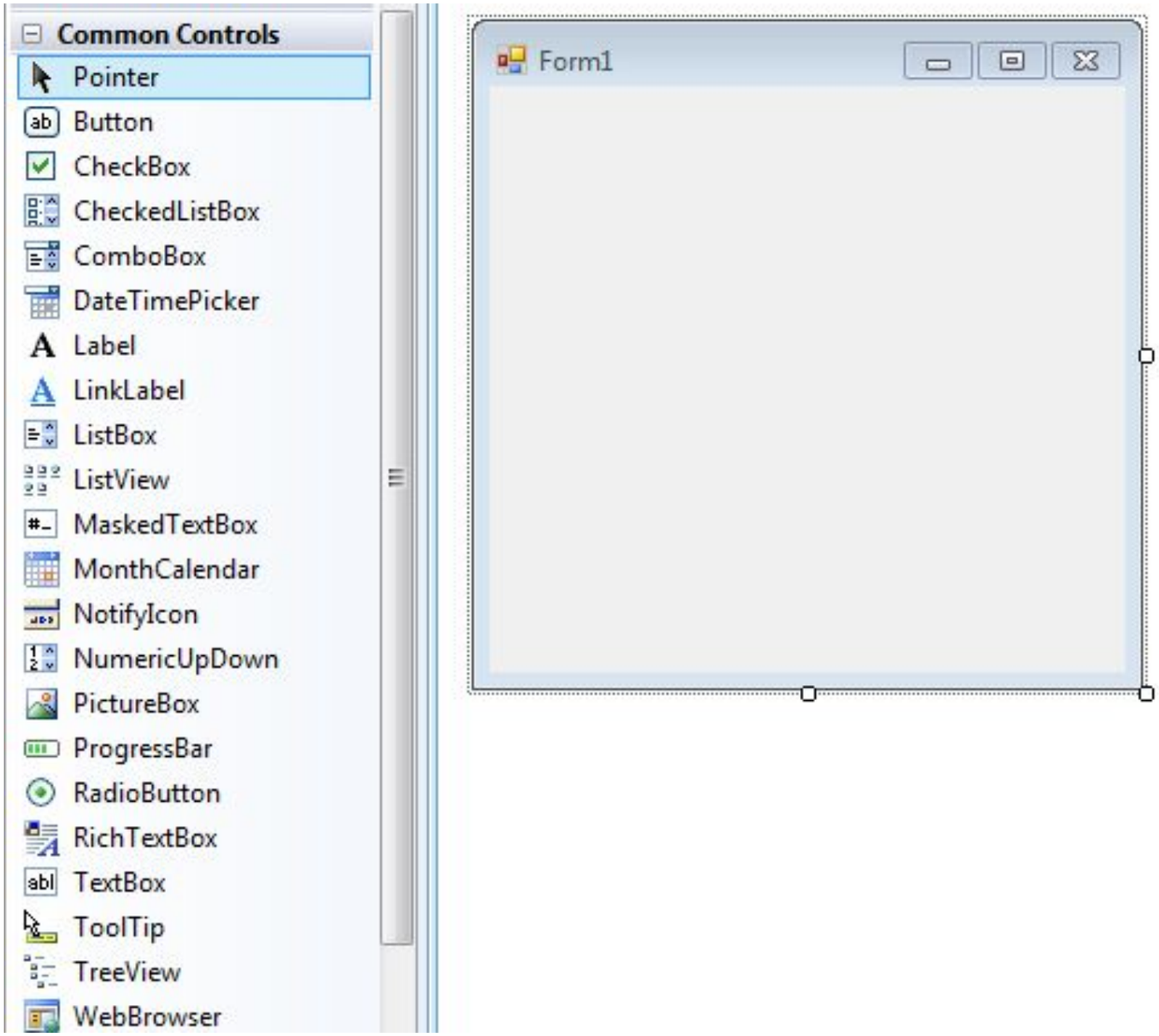
Контрол: Текстовое поле (с фокусом и курсором)

Контрол: Метка

Контрол: Кнопка

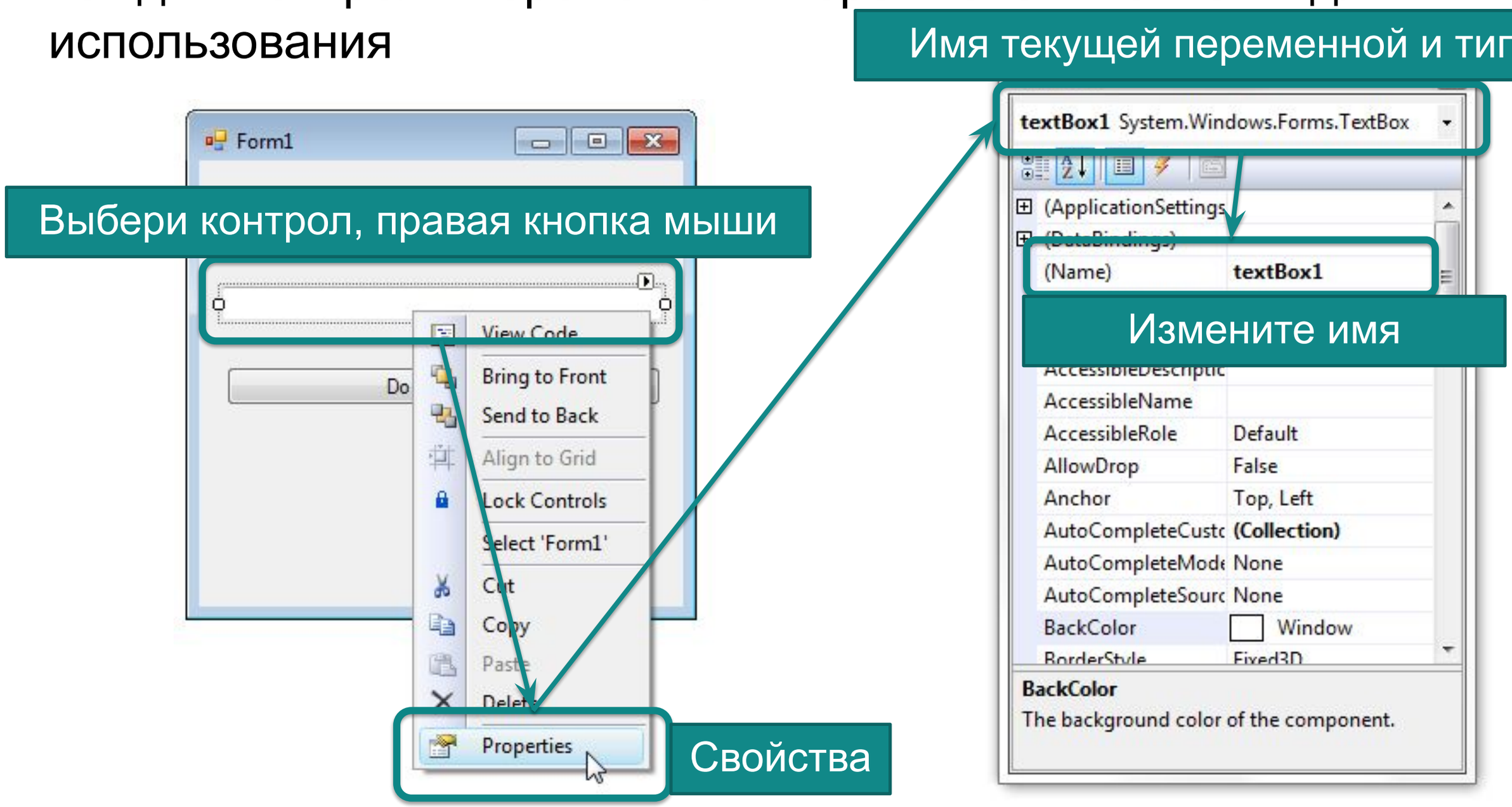


Создадим первую форму



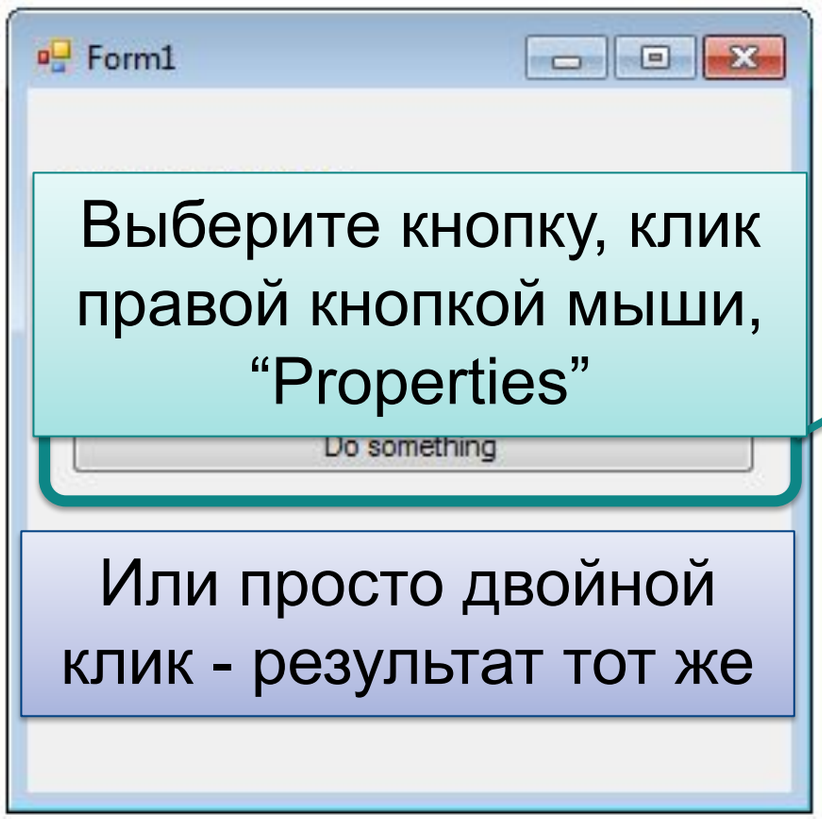
Контролы - они же и переменные

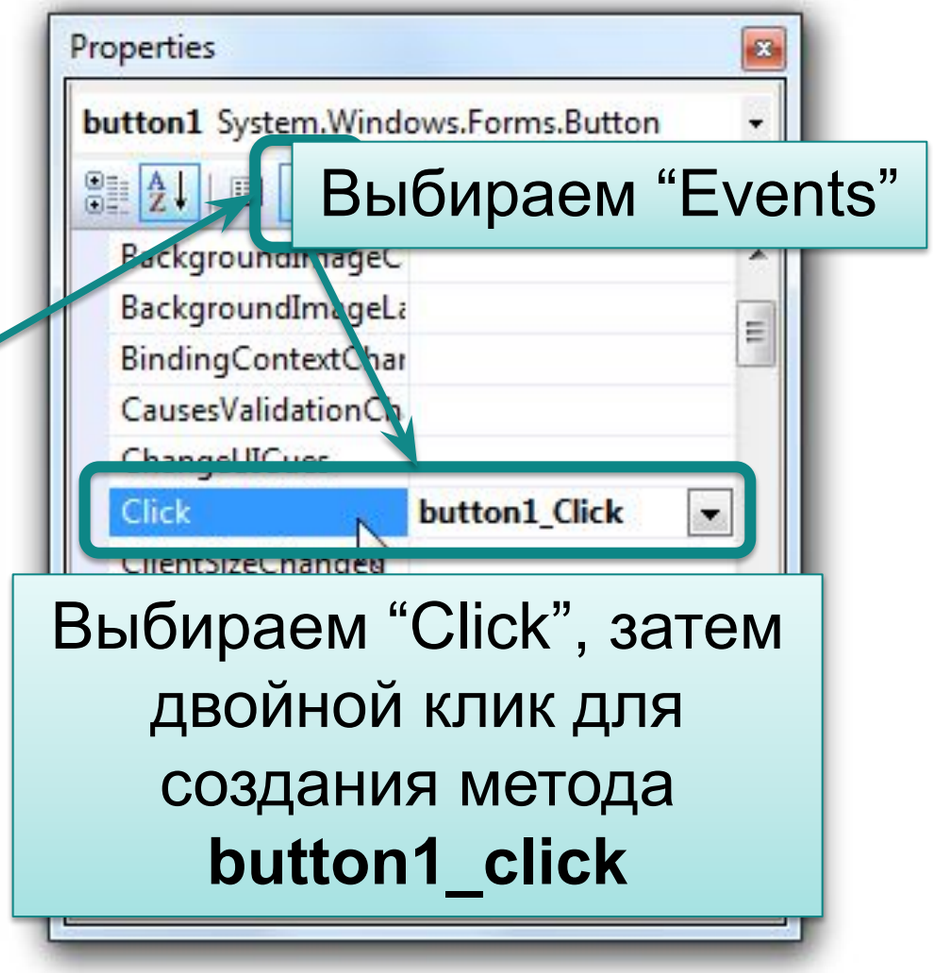
- Каждый контрол - переменная. Переименовывайте их для использования



Делайте что-то, когда пользователь выбрал!

- Для некоторых контролов нужно что-то делать, когда пользователь с ними взаимодействует
  - Пример: пользователь нажал на кнопку

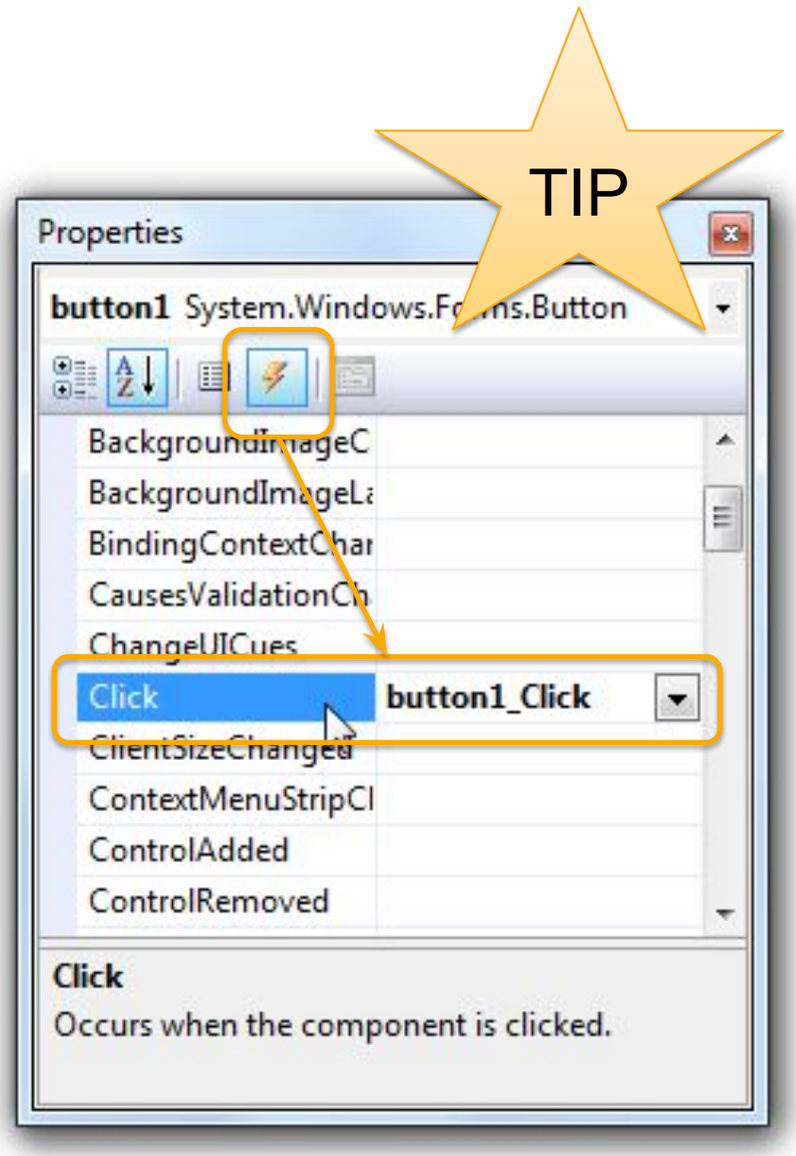




Ключевые моменты

- Создайте новую форму из меню *Project*, *Add Windows Form*
- Основные контролы: метка, текстовое поле, кнопка
- При взаимодействии с пользователем используются события
- Как использовать метод ShowDialog для формы

Создание события



Использование форм внутри AutoCAD

- Модальные формы
  - Application.ShowModalDialog
- Немодальные формы (рассмотрим вместо них палитры)
  - Application.ShowModelessDialog
  - Автоматически сохраняется размер и положение

Контекстное меню

- Уровень приложения
  - Application.AddDefaultContextMenuExtension
- Уровень объекта
  - Application. AddObjectContextMenuExtension – для RXClass

Вкладки расширения диалога

- Создание новой вкладки в диалоге настройки (Options)

Повестка дня

- Обзор .NET
- Основы написания приложений AutoCAD .NET API
- Взаимодействие с пользователем
- Основы базы данных
- Словари
- Пользовательский интерфейс
- События
- Input PointMonitor
- Jigs

Обработка событий - Пример

- Создание обработчика событий (callback)

```
Sub objAppended(ByVal o As Object, ByVal e As ObjectEventArgs)
    'Do something here
    'Do something else, etc.
End Sub
```

- Сопоставление обработчика события и события

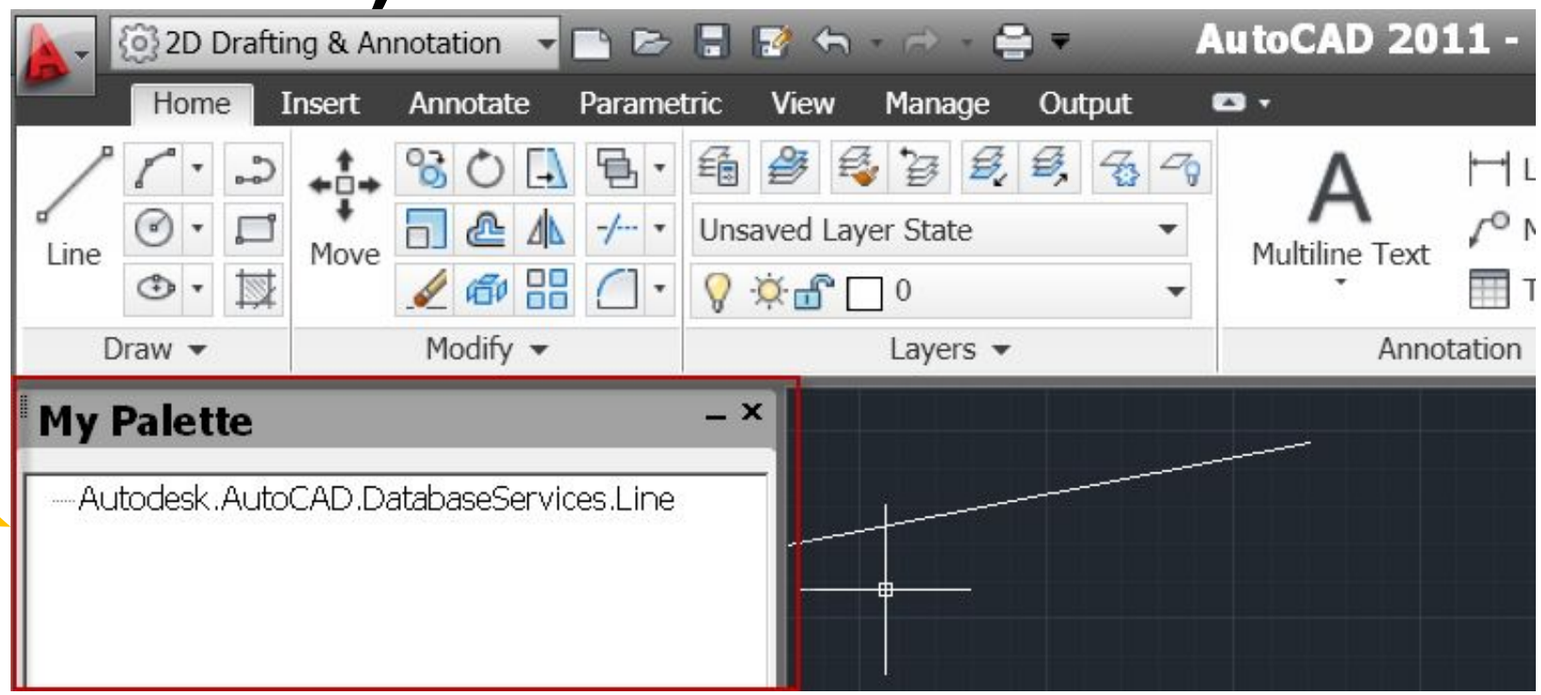
```
Dim db As Database = Application.DocumentManager.MdiActiveDocument.Database
AddHandler db.ObjectAppended, New ObjectEventHandler(AddressOf objAppended)
```

- Удаление обработчика события

```
RemoveHandler db.ObjectAppended, AddressOf objAppended
```

Набор палитр (PaletteSet)

- Пользовательские прикрепляемые окна внутри AutoCAD

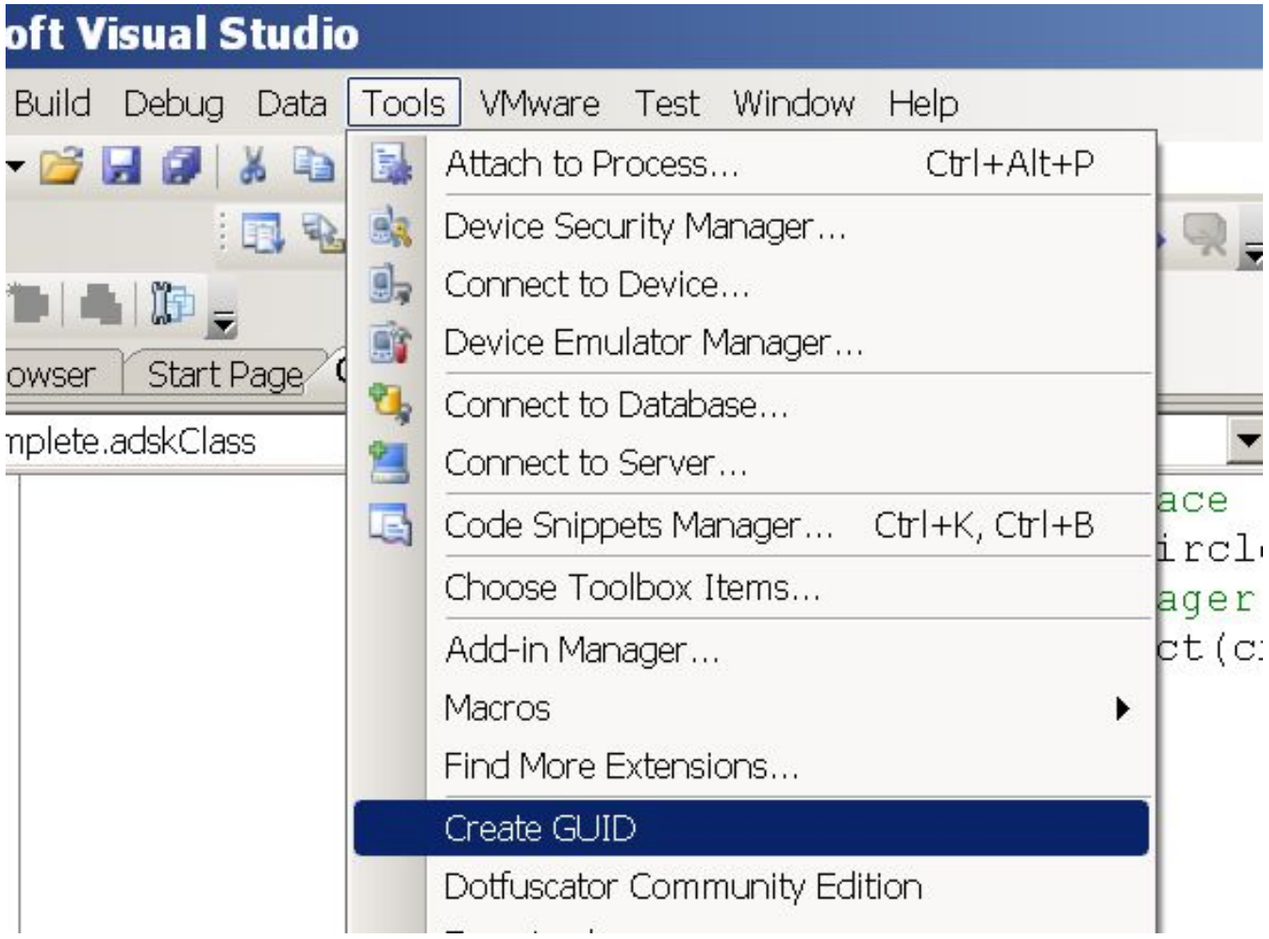


- В пространстве имён Autodesk.AutoCAD.Windows

- PaletteSets содержит контролы, такие как UserControl
  - Используйте Visual Studio wizards для создания контролов
  - Используйте метод Add класса PaletteSet для добавления UserControl

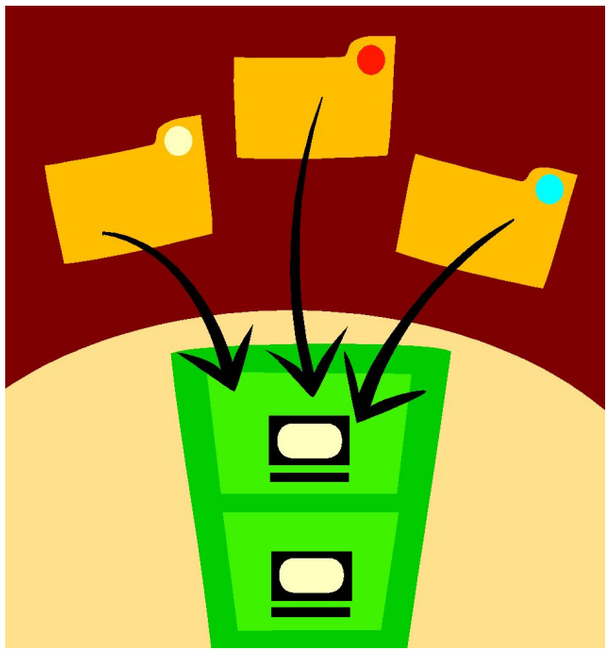
PaletteSet 2

- Конструктору PaletteSet нужен уникальный GUID
- В Visual Studio в меню Tools выберите "Create Guid".
- Перед созданием PaletteSet проверяйте не существует ли он уже.
- Используйте глобальную переменную для PaletteSet




```
If (myPaletteSet = Nothing) Then
    myPaletteSet = New PaletteSet("My Palette", New Guid("D61D0875-A507-4b73-8B5F-9266BEACD596"))
    myPalette = New UserControl1
    myPaletteSet.Add("Palette1", myPalette)
End If
```

Lab 4 - PaletteSet и события базы данных



Повестка дня

- Обзор .NET
- Основы написания приложений AutoCAD .NET API
- Взаимодействие с пользователем
- Основы базы данных
- Словари
- Пользовательский интерфейс
- События
- Input PointMonitor
- Jigs

InputPoint Monitor

- Позволяет следить за пользовательским вводом в AutoCAD, чрезвычайно мощное API.
- Обеспечивает соответствующие данные о вводе, такие как - OSNAP, контекст рисования, вычисляемы точки, примитивы в пределах прицела, и т.д.
- Так же позволяет рисовать временную графику и легко реализовывать подсказки
- Создаётся событием PointMonitor класса Editor
  - Делегат - PointMonitorEventHandler

```
<CommandMethod("addPointmonitor")> _
Public Sub startMonitor()
    Dim ed As Editor = Application.DocumentManager.MdiActiveDocument.Editor
    AddHandler ed.PointMonitor, New PointMonitorEventHandler(AddressOf MyPointMonitor)

End Sub
```

```
Public Sub MyPointMonitor(ByVal sender As Object, ByVal e As PointMonitorEventArgs)
```

Lab 6 - PointMonitor

Повестка дня

- Обзор .NET
- Основы написания приложений AutoCAD .NET API
- Взаимодействие с пользователем
- Основы базы данных
- Словари
- Пользовательский интерфейс
- События
- Input PointMonitor
- Jigs

Jigs (джиг)

- Позволяет графически манипулировать формой примитива в реальном времени
- Доступно два типа Jig
  - EntityJig – только для управления одним примитивом
  - DrawJig – для управления одним и более примитивами
- Необходимо создать класс, наследующий EntityJig или DrawJig
- Две функции должны быть переопределены: Sampler and Update
- Конструктор этого класса принимает параметр - примитив для манипуляции
  - Используйте метод Drag класса Editor Drag для начала манипуляций

```
Dim circle As Circle = New Circle(Point3d.Origin, Vector3d.ZAxis, 10)
Dim jig As New MyCircleJig(circle)
Dim ed As Autodesk.AutoCAD.EditorInput.Editor = _
                    Application.DocumentManager.MdiActiveDocument.Editor
Dim promptResult As PromptResult = ed.Drag(jig)
```

Lab 7 - Jig

Спасибо!

Autodesk