REVIT RTC TECHNOLOGY CONFERENCE NORTH AMERICA 2014

Learn to Write Add-ons No Coding Exp Required!

Adam Sheather

YTL Corporation

Who am I?

- BIM Manager for YTL Corporation Malaysia
- All in one company Property Developer, Design, Engineering, QS, Construction and FM
- Hotels, Resorts, Residential, Commercial, Retail, Power and Rail
- Projects from 10mill to 2.5billionUSD
- Programming for 4-5 years
- Self taught a long struggle, had lots of help from community and the Dev Days with Jeremy Tammik were invaluable
- Have written the Company API toolkits for ADG and GHD
- Now do a lot of things with opensource, javascipt, C/C++, PHP, F# (dabbling).
- Have developed custom addons for Dynamo

Overall Lesson Plan

- Lab 1 – Computing Essentials, Visual Studio and First Project
- Lab 2a – Revit API Pre-Starters, Setup and VSTA
- Lab 2b – Select Objects, get data, set data, export/import data
- Lab 3 – UI Setup, Project Templates, Views
- Lab 4 – Export/Rename, Place Families, Create Floors, Events

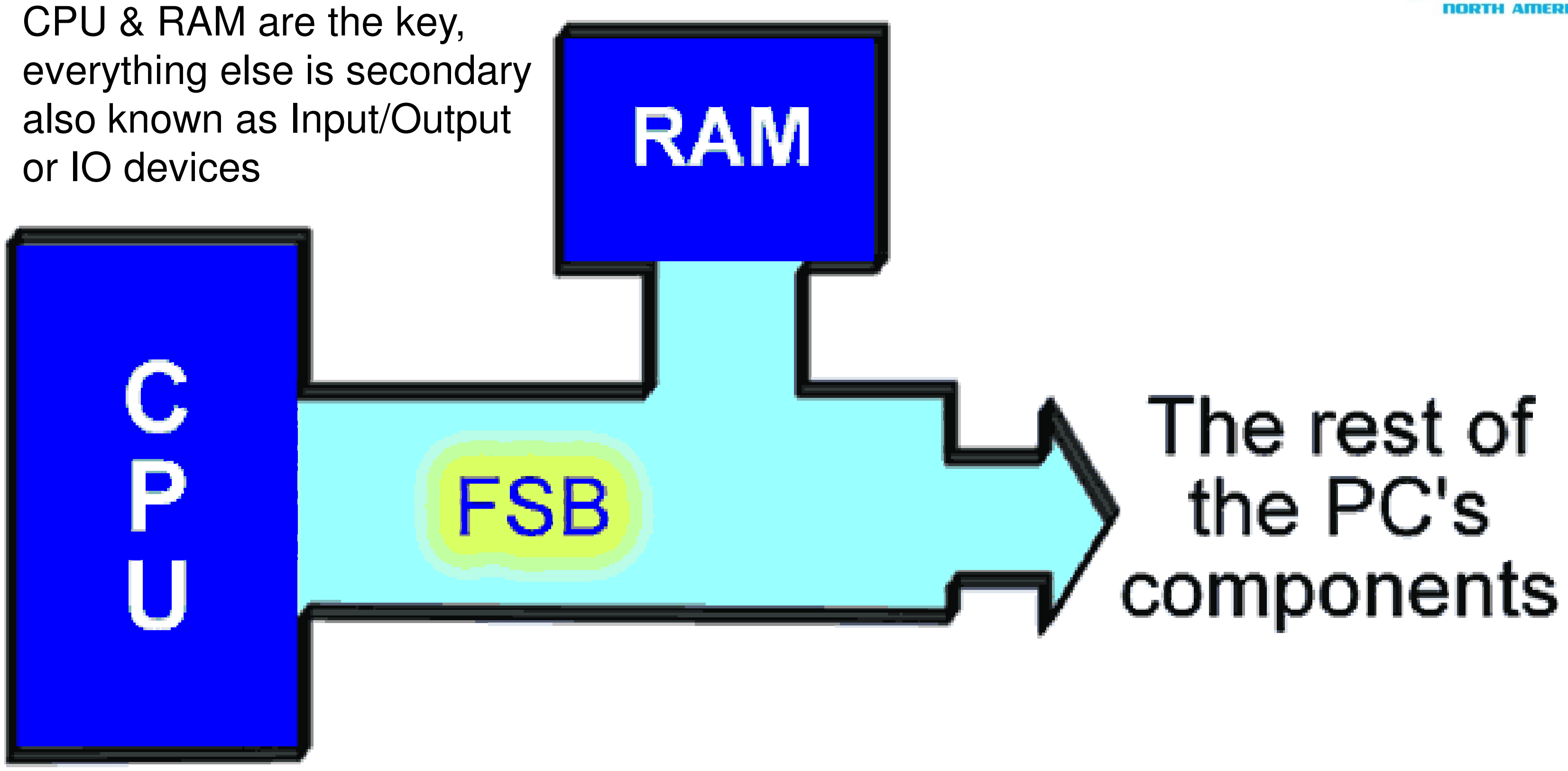
Lab 1

- Computing Essentials CPU, RAM and IO
- Bytes, Memory and ASCII
- Language History, C#, and API’s
- Object Oriented Programming and Classes
- Visual Studio Essentials
- Core Syntax Rules (Syntax = Programming grammar)
- First Project

Computing Essentials

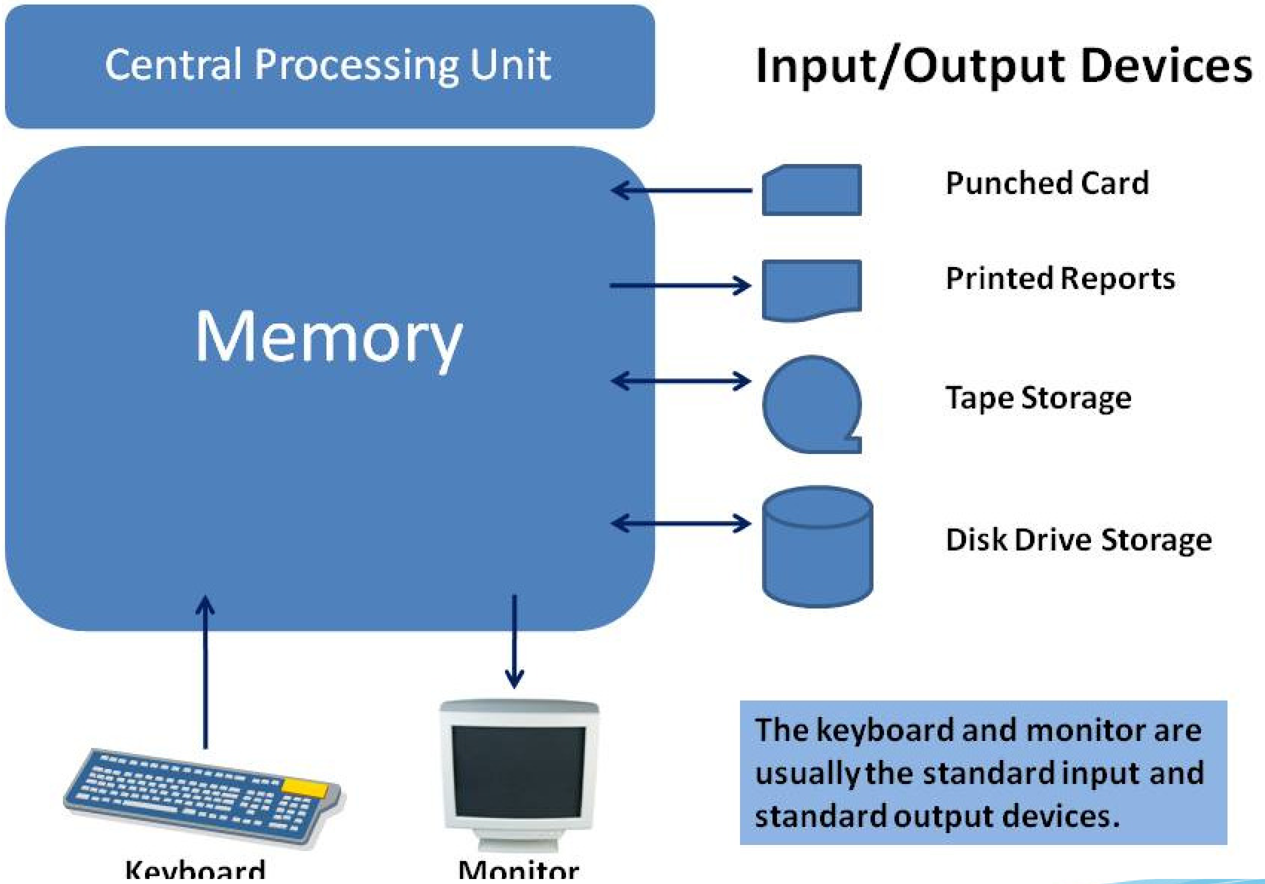
- CPU and RAM
- Input/Output Devices

Computer Essentials.
CPU & RAM are the key,
everything else is secondary
also known as Input/Output
or IO devices



The Key Pieces of a Computer System
http://www.karbosguide.com/books/pcarchitecture/chapter26.htm

Examples of IO Devices



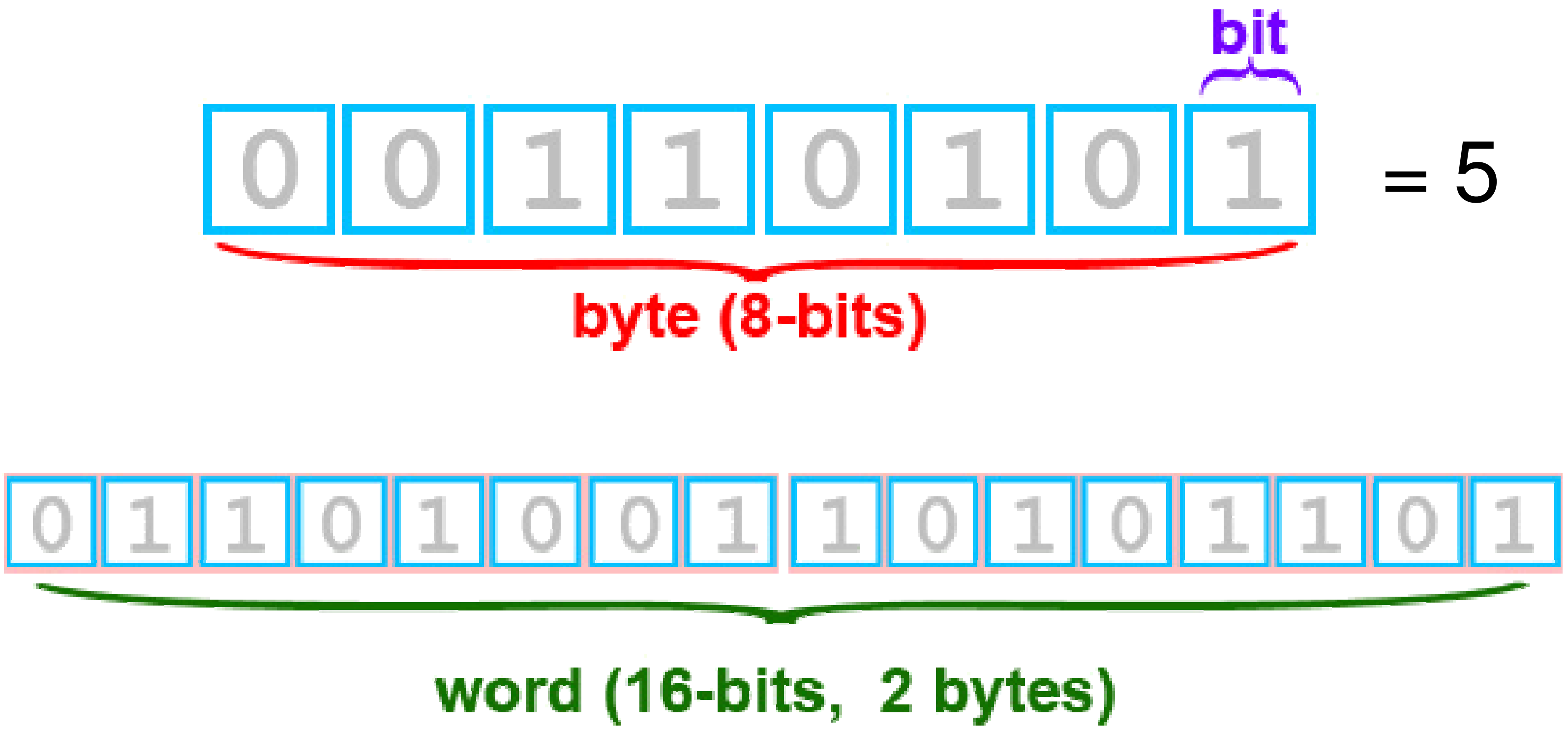
The keyboard and monitor are usually the standard input and standard output devices.

http://inst.eecs.berkeley.edu/~cs61cl/fa08/labs/lab25.html

Bytes, Memory and ASCII

- Bits, Bytes and so on
- Memory Systems and Addresses
- Hexadecimals
- ASCII

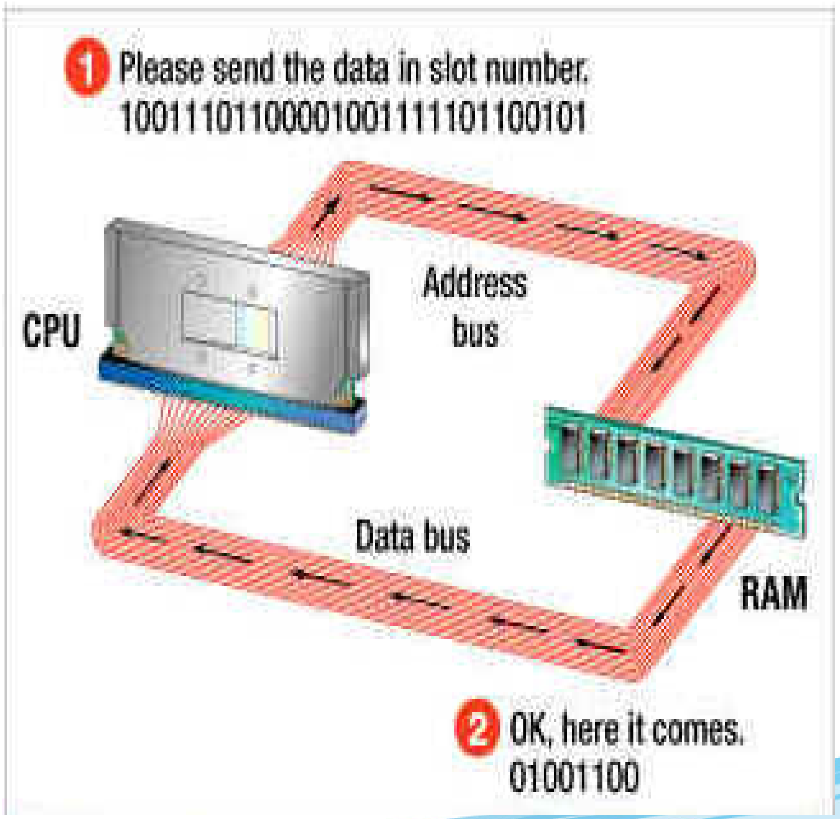
Bits & Bytes



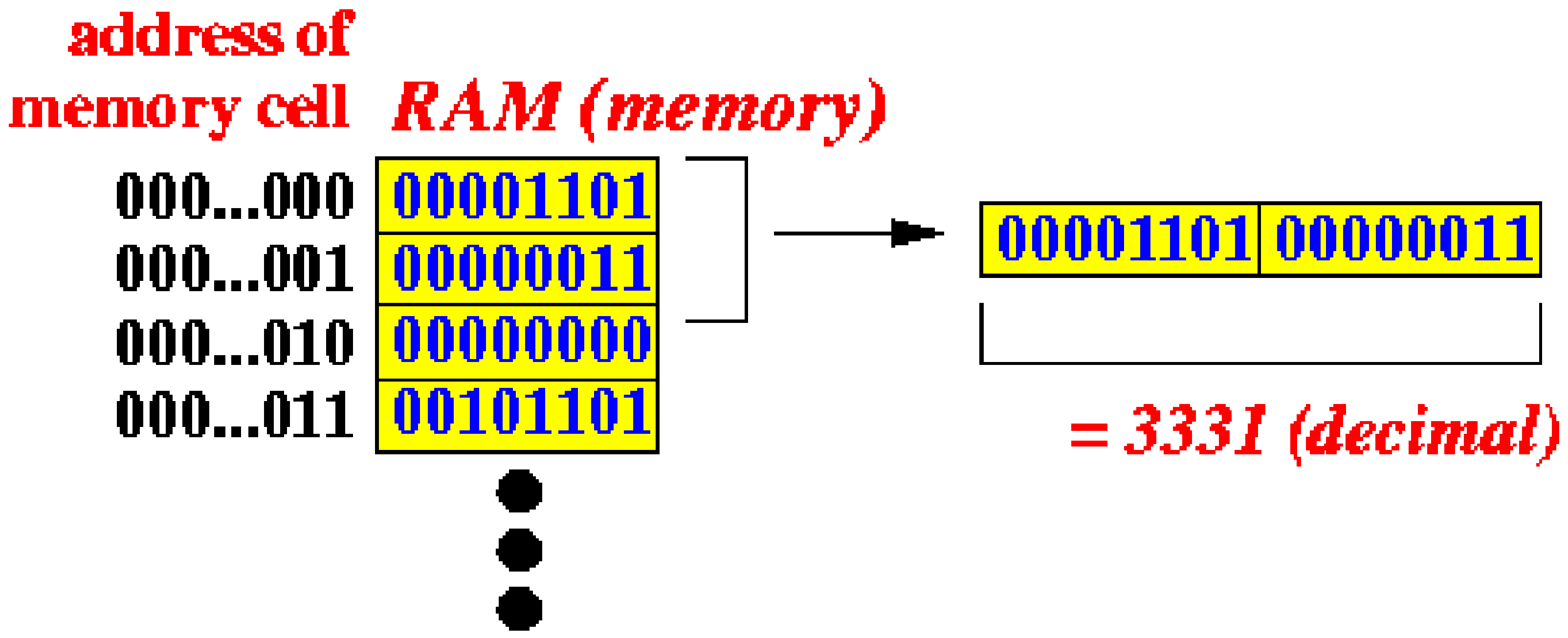
http://iesfmontseny.es/clil/bits_and_bytes.html

Bytes, Memory and ASCII

- Lowest Part of Memory Address is a Byte
- Bytes are made up Bits
- CPU's ask for Addresses of bytes manipulate them and send them back to the same or different RAM Addresses.


1 Please send the data in slot number. 10011101100001001111101100101
CPU
Address bus
Data bus
RAM
2 OK, here it comes. 01001100

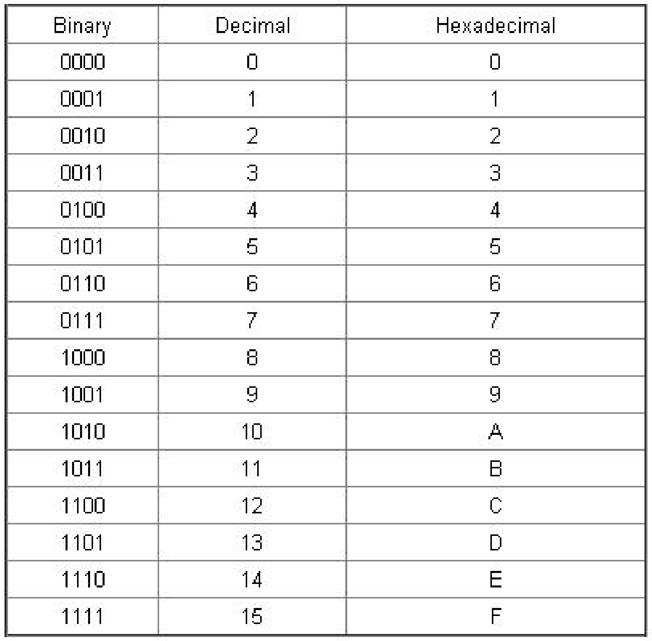
http://www.alf.sd83.bc.ca/courses/It12/using_it/processor_speed.htm


address of memory cell
RAM (memory)
000...000
000...001
000...010
000...011
00001101
00000011
00000000
00101101
00001101 00000011
= 3331 (decimal)

Hexadecimals

It uses sixteen distinct symbols, most often the symbols 0–9 to represent values zero to nine, and A, B, C, D, E, F (or alternatively a–f) to represent values ten to fifteen. For example, the hexadecimal number 2AF3 is equal, in decimal, to $(2 \times 16^3) + (10 \times 16^2) + (15 \times 16^1) + (3 \times 16^0)$, or 10995.

RAM Memory addresses are generally stored as hexadecimal locations

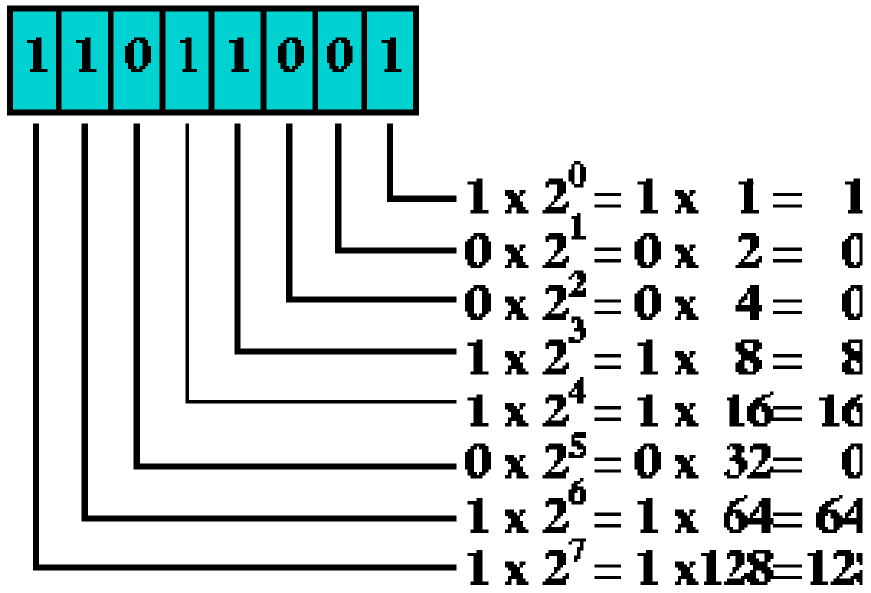


Binary	Decimal	Hexadecimal
0000	0	0
0001	1	1
0010	2	2
0011	3	3
0100	4	4
0101	5	5
0110	6	6
0111	7	7
1000	8	8
1001	9	9
1010	10	A
1011	11	B
1100	12	C
1101	13	D
1110	14	E
1111	15	F

http://www.learn44.com/binary-to-decimal-and-hexadecimal-conversion-chart/
Wikipedia

Calculating Hexadecimals to Binary

REVIT RTC TECHNOLOGY CONFERENCE NORTH AMERICA 2014



$1 \times 2^0 = 1 \times 1 = 1$

$0 \times 2^1 = 0 \times 2 = 0$

$0 \times 2^2 = 0 \times 4 = 0$

$1 \times 2^3 = 1 \times 8 = 8$

$1 \times 2^4 = 1 \times 16 = 16$

$0 \times 2^5 = 0 \times 32 = 0$

$1 \times 2^6 = 1 \times 64 = 64$

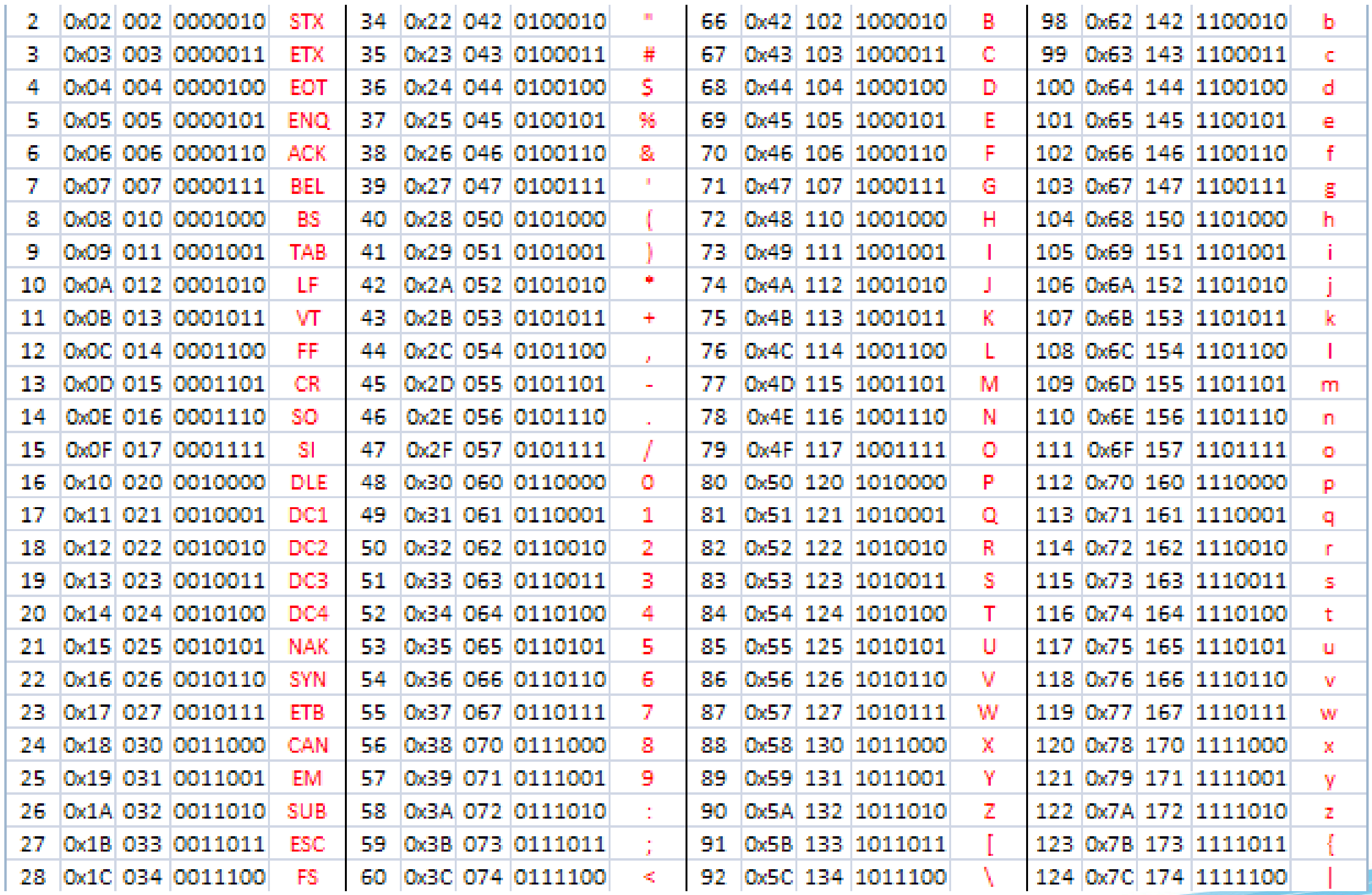
$1 \times 2^7 = 1 \times 128 = 12$

$$1 + 8 + 16 + 64 + 128 = 217$$

http://www.freesoft.org/CIE/Topics/19.htm

ASCII

- The American Standard Code for Information Interchange (ASCII /ˈæski/ *ASS-kee*)[1] is a character-encoding scheme originally based on the English alphabet that encodes 128 specified characters - the numbers 0-9, the letters a-z and A-Z, some basic punctuation symbols, some control codes that originated with Teletype machines, and a blank space - into the 7-bit binary integers.

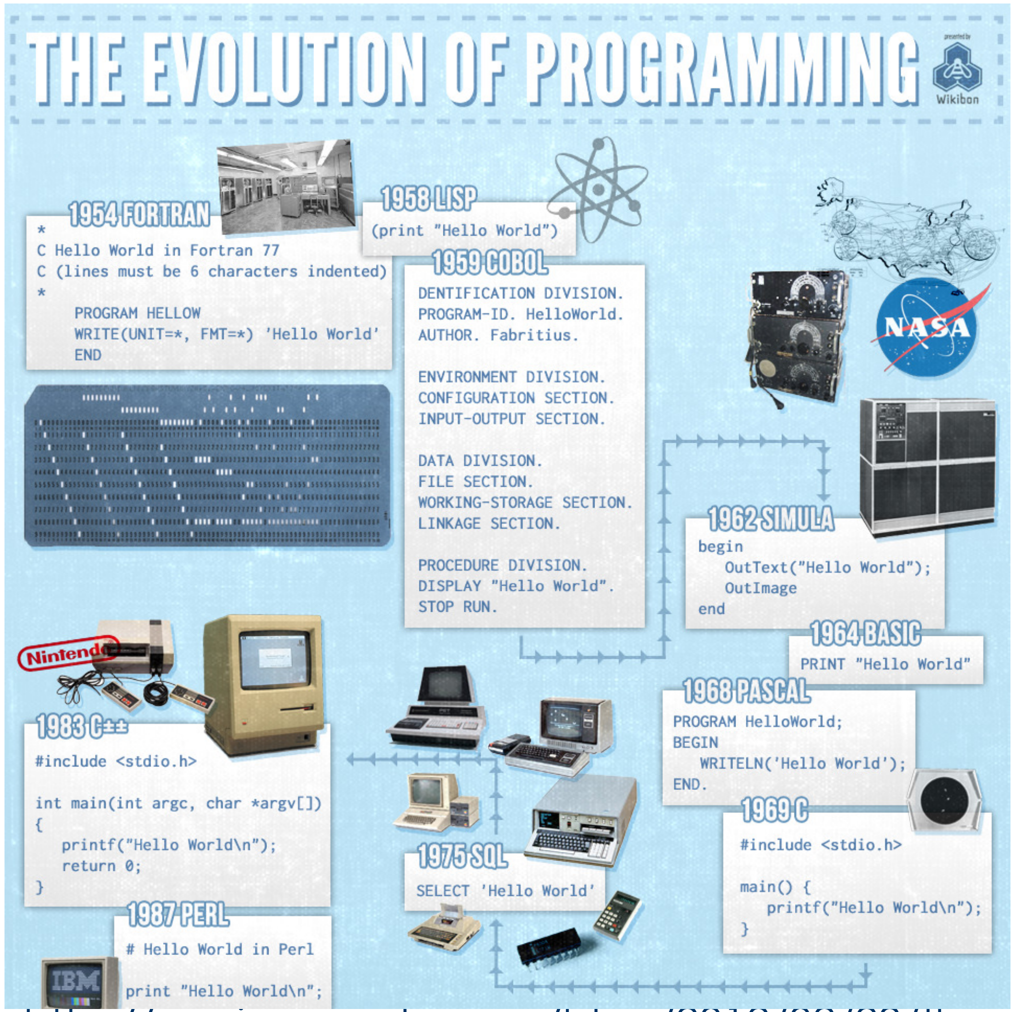


2	0x02	002	0000010	STX	34	0x22	042	0100010	"	66	0x42	102	1000010	B	98	0x62	142	1100010	b
3	0x03	003	0000011	ETX	35	0x23	043	0100011	#	67	0x43	103	1000011	C	99	0x63	143	1100011	c
4	0x04	004	0000100	EOT	36	0x24	044	0100100	$	68	0x44	104	1000100	D	100	0x64	144	1100100	d
5	0x05	005	0000101	ENQ	37	0x25	045	0100101	%	69	0x45	105	1000101	E	101	0x65	145	1100101	e
6	0x06	006	0000110	ACK	38	0x26	046	0100110	&	70	0x46	106	1000110	F	102	0x66	146	1100110	f
7	0x07	007	0000111	BEL	39	0x27	047	0100111	'	71	0x47	107	1000111	G	103	0x67	147	1100111	g
8	0x08	010	0001000	BS	40	0x28	050	0101000	(	72	0x48	110	1001000	H	104	0x68	150	1101000	h
9	0x09	011	0001001	TAB	41	0x29	051	0101001	)	73	0x49	111	1001001	I	105	0x69	151	1101001	i
10	0x0A	012	0001010	LF	42	0x2A	052	0101010	*	74	0x4A	112	1001010	J	106	0x6A	152	1101010	j
11	0x0B	013	0001011	VT	43	0x2B	053	0101011	+	75	0x4B	113	1001011	K	107	0x6B	153	1101011	k
12	0x0C	014	0001100	FF	44	0x2C	054	0101100	,	76	0x4C	114	1001100	L	108	0x6C	154	1101100	l
13	0x0D	015	0001101	CR	45	0x2D	055	0101101	-	77	0x4D	115	1001101	M	109	0x6D	155	1101101	m
14	0x0E	016	0001110	SO	46	0x2E	056	0101110	.	78	0x4E	116	1001110	N	110	0x6E	156	1101110	n
15	0x0F	017	0001111	SI	47	0x2F	057	0101111	/	79	0x4F	117	1001111	O	111	0x6F	157	1101111	o
16	0x10	020	0010000	DLE	48	0x30	060	0110000	0	80	0x50	120	1010000	P	112	0x70	160	1110000	p
17	0x11	021	0010001	DC1	49	0x31	061	0110001	1	81	0x51	121	1010001	Q	113	0x71	161	1110001	q
18	0x12	022	0010010	DC2	50	0x32	062	0110010	2	82	0x52	122	1010010	R	114	0x72	162	1110010	r
19	0x13	023	0010011	DC3	51	0x33	063	0110011	3	83	0x53	123	1010011	S	115	0x73	163	1110011	s
20	0x14	024	0010100	DC4	52	0x34	064	0110100	4	84	0x54	124	1010100	T	116	0x74	164	1110100	t
21	0x15	025	0010101	NAK	53	0x35	065	0110101	5	85	0x55	125	1010101	U	117	0x75	165	1110101	u
22	0x16	026	0010110	SYN	54	0x36	066	0110110	6	86	0x56	126	1010110	V	118	0x76	166	1110110	v
23	0x17	027	0010111	ETB	55	0x37	067	0110111	7	87	0x57	127	1010111	W	119	0x77	167	1110111	w
24	0x18	030	0011000	CAN	56	0x38	070	0111000	8	88	0x58	130	1011000	X	120	0x78	170	1111000	x
25	0x19	031	0011001	EM	57	0x39	071	0111001	9	89	0x59	131	1011001	Y	121	0x79	171	1111001	y
26	0x1A	032	0011010	SUB	58	0x3A	072	0111010	:	90	0x5A	132	1011010	Z	122	0x7A	172	1111010	z
27	0x1B	033	0011011	ESC	59	0x3B	073	0111011	;	91	0x5B	133	1011011	[	123	0x7B	173	1111011	{
28	0x1C	034	0011100	FS	60	0x3C	074	0111100	<	92	0x5C	134	1011100	\	124	0x7C	174	1111100	|

http://www.catonmat.net/blog/ascii-cheat-sheet/

Language History, API's and dotnet

- History of Languages
- Why C#
- What is an API
- Dotnet



1954 FORTRAN
```
*
C Hello World in Fortran 77
C (lines must be 6 characters indented)
*
    PROGRAM HELLOW
    WRITE(UNIT=*, FMT=*) 'Hello World'
    END
```

1958 LISP
```
(print "Hello World")
```

1959 COBOL
```
DENTIFICATION DIVISION.
PROGRAM-ID. HelloWorld.
AUTHOR. Fabritius.

ENVIRONMENT DIVISION.
CONFIGURATION SECTION.
INPUT-OUTPUT SECTION.

DATA DIVISION.
FILE SECTION.
WORKING-STORAGE SECTION.
LINKAGE SECTION.

PROCEDURE DIVISION.
DISPLAY "Hello World".
STOP RUN.
```

1962 SIMULA
```
begin
   OutText("Hello World");
   OutImage
end
```

1964 BASIC
```
PRINT "Hello World"
```

1968 PASCAL
```
PROGRAM HelloWorld;
BEGIN
   WRITELN('Hello World');
END.
```

1969 C
```
#include <stdio.h>

main() {
   printf("Hello World\n");
}
```

1975 SQL
```
SELECT 'Hello World'
```

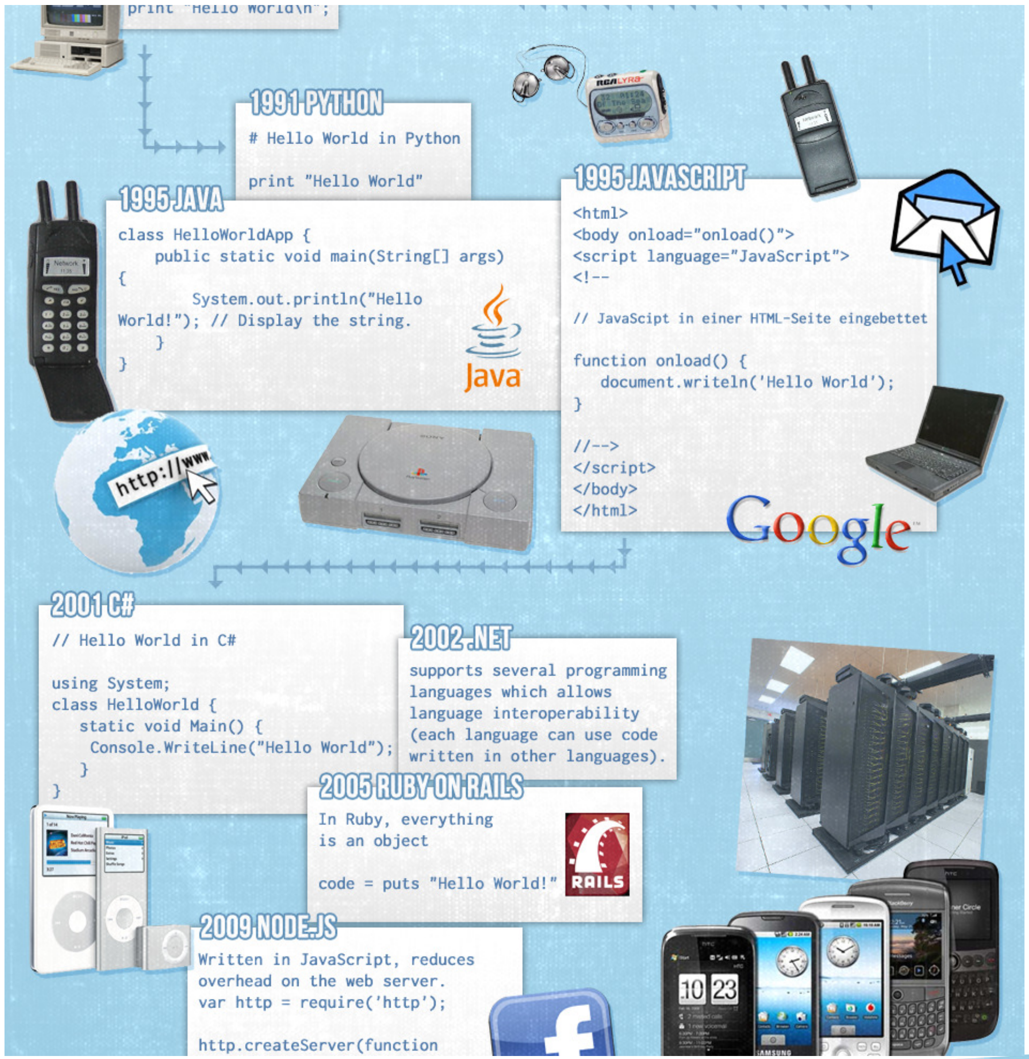
1983 C++
```
#include <stdio.h>

int main(int argc, char *argv[])
{
   printf("Hello World\n");
   return 0;
}
```

1987 PERL
```
# Hello World in Perl

print "Hello World\n";
```

http://servicesangle.com/blog/2012/02/29/the-evolution-of-programming-infographic/



```
print "Hello World\n";
```

1991 PYTHON

```
# Hello World in Python

print "Hello World"
```

1995 JAVA

```
class HelloWorldApp {
    public static void main(String[] args)
{
        System.out.println("Hello
World!"); // Display the string.
    }
}
```

1995 JAVASCRIPT

```
<html>
<body onload="onload()">
<script language="JavaScript">
<!--

// JavaScipt in einer HTML-Seite eingebettet

function onload() {
   document.writeln('Hello World');
}

//-->
</script>
</body>
</html>
```

2001 C#

```
// Hello World in C#

using System;
class HelloWorld {
   static void Main() {
    Console.WriteLine("Hello World");
   }
}
```

2002 .NET

```
supports several programming
languages which allows
language interoperability
(each language can use code
written in other languages).
```

2005 RUBY ON RAILS

```
In Ruby, everything
is an object

code = puts "Hello World!"
```

2009 NODE.JS

```
Written in JavaScript, reduces
overhead on the web server.
var http = require('http');

http.createServer(function
```

http://servicesangle.com/blog/2012/02/29/the-evolution-of-programming-infographic/

Why C#

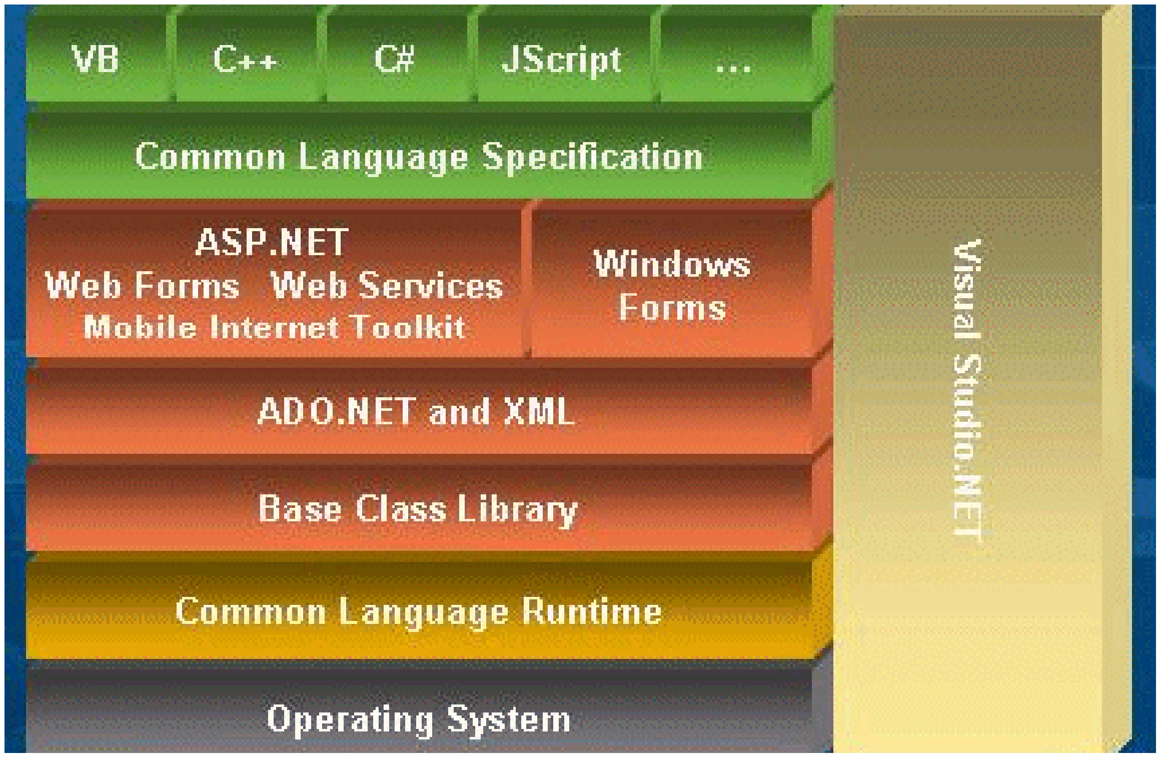
- Part of the C language family more easily transferred skills
- VB syntax very one-off
- A lot more Samples around for Revit
- Personal preference ☺

Ultimately the two can do the same things as they both run on the dotnet framework or the same system they just go about it different ways.

What is an API? (Application Programming Interface)

- specifies how some software components should interact with each other.
- accessing databases or computer hardware, such as hard disk drives or video cards,
- I comes in the form of a library that includes specifications for routines, data structures, object classes, and variables
- An API specification can take many forms, such as the Microsoft Windows API,
- the libraries of a programming language, e.g., C++ or Java API.
- Web APIs are also a vital component of today's web fabric.

What is dotnet? It's the Windows API



http://www.wilsonmar.com/msdotnet.htm

Object Oriented Programming

- C# is an OOP language
- Everything in C# is an object
- All Objects contain fields which are like Revit Parameters
- All Objects can contain methods which are like Revit Object Manipulations (say using the "trim" method on a wall vs dragging with a mouse.
- An Object is known programmatically as a "Class"

Object-oriented programming (OOP) is a programming paradigm that represents the concept of "objects" that have data fields (attributes that describe the object) and associated procedures known as methods. Objects, which are usually instances of classes, are used to interact with one another to design applications and computer programs.

Object Templates are called Classes

There are some key components that make up a class. A class is very similar to a Revit family in that the class itself is merely a template, it’s not until you create an instance of the class in a project that it uses the template and becomes an object. This is similar to a beam family loaded into a project but not used. Every time you create an instance it creates a version of the family. This creation process is called using a “Constructor Method” or Constructor.

Three Major Components make up Classes.

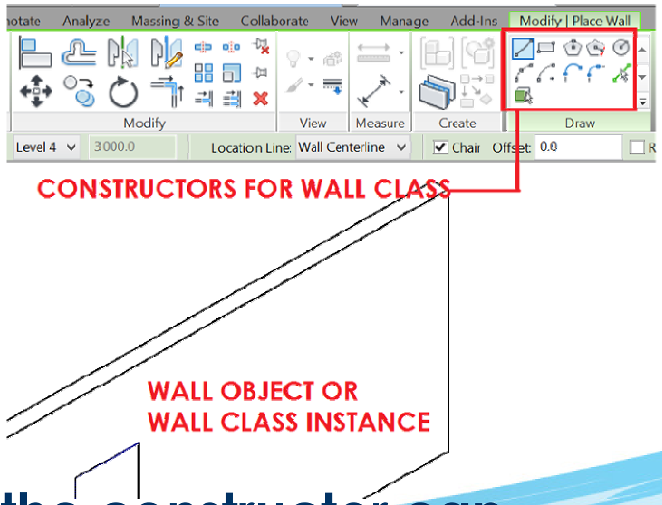
- Constructors
- Fields
- Methods

C# Classes Constructors

Constructors are the instructions to create an instance of an object.

You can have multiple Constructors for a Class.

An Example is with Revit walls.



Wall by Line/Arc

Wall by Face

Wall by Picked Line

Create Similar

In some cases byFace and Create similar the constructor can specify or constrain the object in certain ways.

C# Classes Fields

Fields are the Parameters of a class. They can contain numbers, decimals, text and Yes/No Parameters.

You create many of these, and programmatically they can also be known as variables.

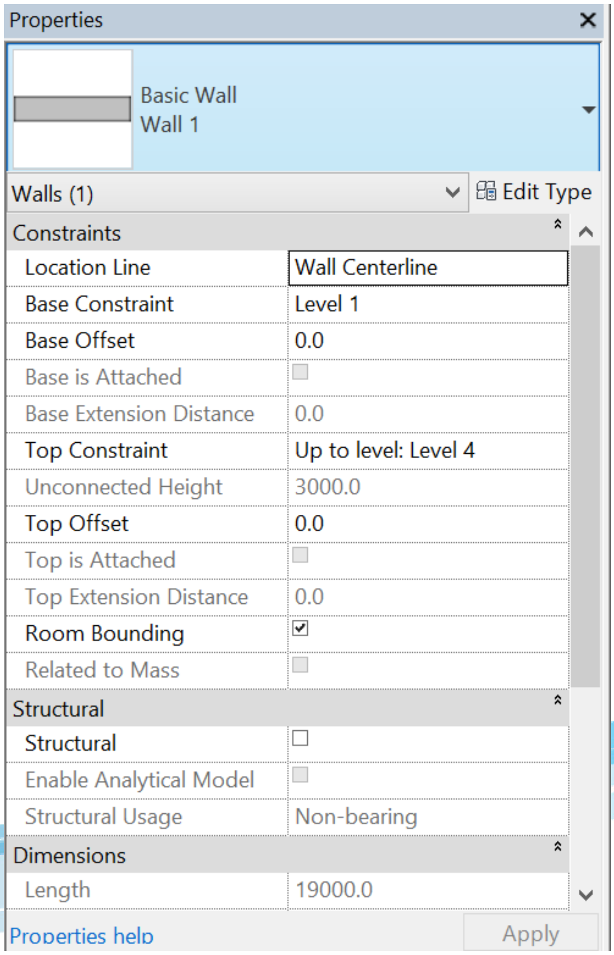
The coding terminology for these items is below

Number – Integer (int)

Text – String (string)

Decimals – double (double)

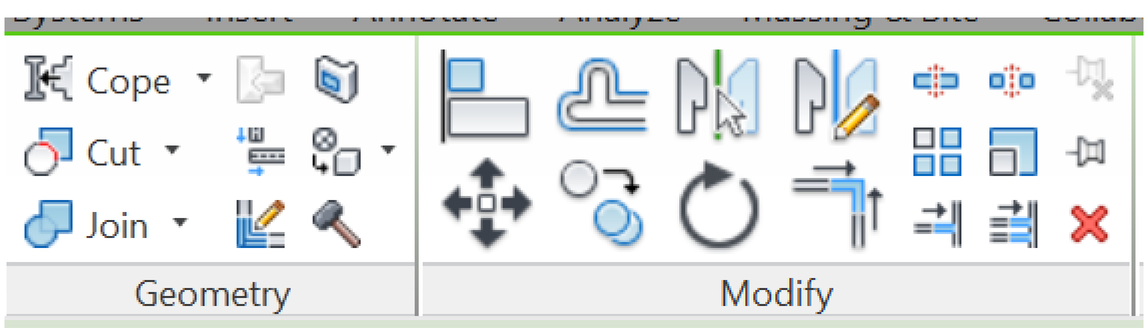
Yes/No – Boolean (bool)



Constraints	
Location Line	Wall Centerline
Base Constraint	Level 1
Base Offset	0.0
Base is Attached	☐
Base Extension Distance	0.0
Top Constraint	Up to level: Level 4
Unconnected Height	3000.0
Top Offset	0.0
Top is Attached	☐
Top Extension Distance	0.0
Room Bounding	✓
Related to Mass	☐
Structural	
Structural	☐
Enable Analytical Model	☐
Structural Usage	Non-bearing
Dimensions	
Length	19000.0

C# Classes Methods

Methods are the instructions to change or modify an object.



Again this is something that you will use many times to manipulate variables, other methods and other objects in your code.

Methods are very powerful features.

OOP Principles

OOP methods are very hard to explain and outside the main realm of this conversation. The idea behind them is to make code as reusable as possible so you make less code for whole program and if you make a change you only have to make it once and it update everywhere! (sound familiar).

Unlike Revit this is entirely possible with OOP practices however just like Revit, very hard to achieve in some cases without a lot of forethought and planning, which is not our goal in this class ☺

OOP Principles

Major Principals are as follows.

- Encapsulation: Processing functions Data – an example could be ensuring only numbers are every inputted into a calculator so errors cannot occur, this involves binding a method to a Field
- Polymorphism: A horrible word developed by programmers. This means creating methods that can work with many datatypes. Say you have a method that sorts a list from highest to lowest, the use of polymorphism would be to make the method generic so it could access numbers, or decimals or Alphanumerical's and perform the same function.

OOP Principles

Major Principals are as follows.

- Inheritance: The easiest to understand, this is creating base classes of which fields and methods can be passed to child classes. An example would be an instance of a wall who's base class would be a wall type, A wall types base class would be the Wall element. The characteristics from each are passed down and added so if functionality changes at a higher level it passes to all sub-classes.

Using all 3 together is key to having less code and easily modifiable API's for upgrades and bug fixing.

Visual Studio Express – Integrated Development Environment (IDE)

Microsoft has Visual Studio Express (VS) which is a free tool we can download and use to create our API's.

We will also be exploring and using the VTSA in Revit as a great tool to debug portions of code as debugging from VS can take a long time to fire up and waste time especially when you are creating a single method or for R&D.

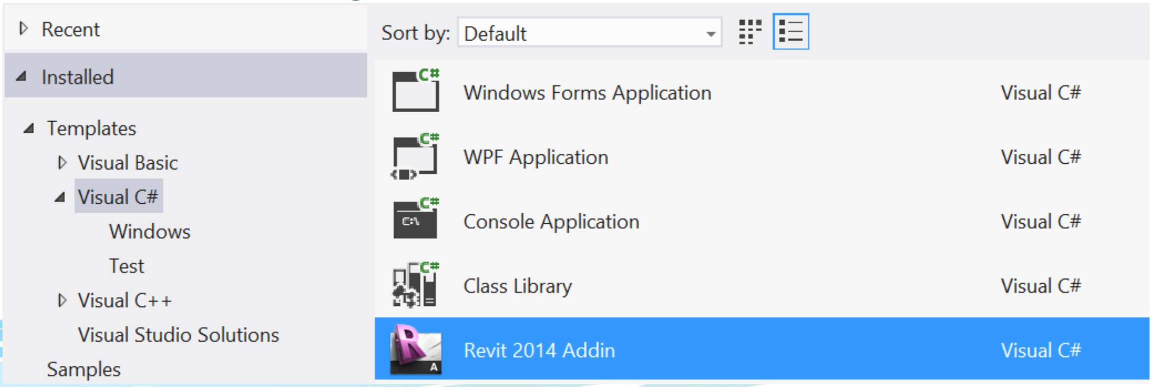
This tool contains a user interface, project management, properties, colour coding, built in compiler (extremely practical compared to other IDE's), debugging features, error highlights and many other things.

Visual Studio will be more then capable of development most Revit Addons. Key features missing are Team Tools, easy data-connectivity to other then MSSQL-Server, ability to create an installer for your API and compile F#(Which is what Dynamo is built from). Otherwise it can and will fulfil all your major needs and I've never needed Pro.

Visual Studio Project Types

Like there are different project types you can create. Similar to Revit you can open a family "project" or a massing "project" as well as a normal project, VS has a number of Projects and each one is important.

- Windows Forms –build interface applications, supports older OS versions
- WPF – New way to Build interfaces, responsive and contextual feedback
- Console – Applications that run, exe files etc…
- Class Library – Creates .dll or library files programs use, Revit API always requires the use of .dll to work
- Revit 2014 Addin – Jeremy Tammiks Template



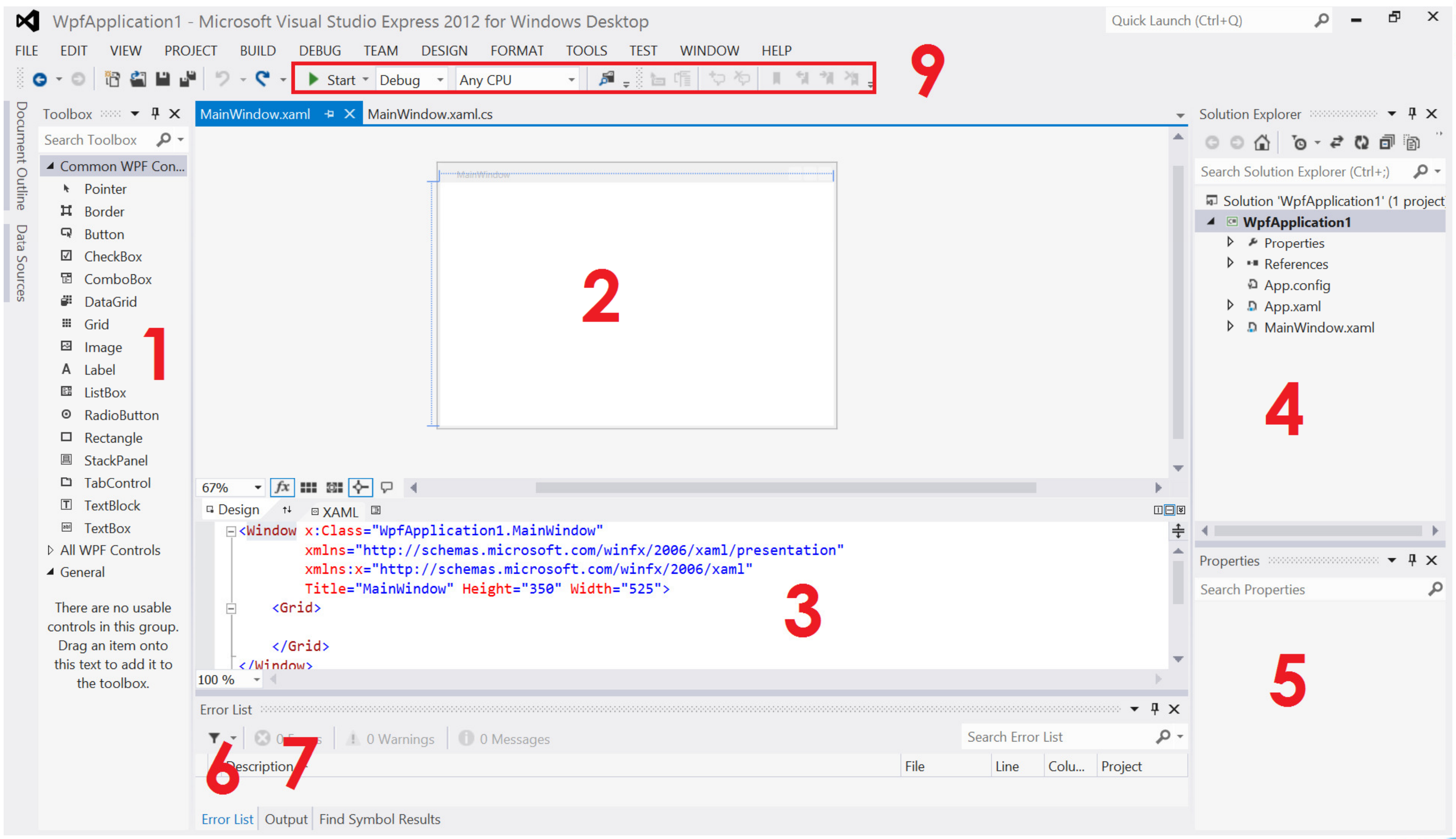
Windows Forms Application	Visual C#
WPF Application	Visual C#
Console Application	Visual C#
Class Library	Visual C#
Revit 2014 Addin	Visual C#

First Project – WPF

For our first project we are going to create a WPF project.

Select the WPF project and call it a new name, if you prefer you can save the project to your USB.

When you open the project you should see something like the following.



There are no usable controls in this group. Drag an item onto this text to add it to the toolbox.

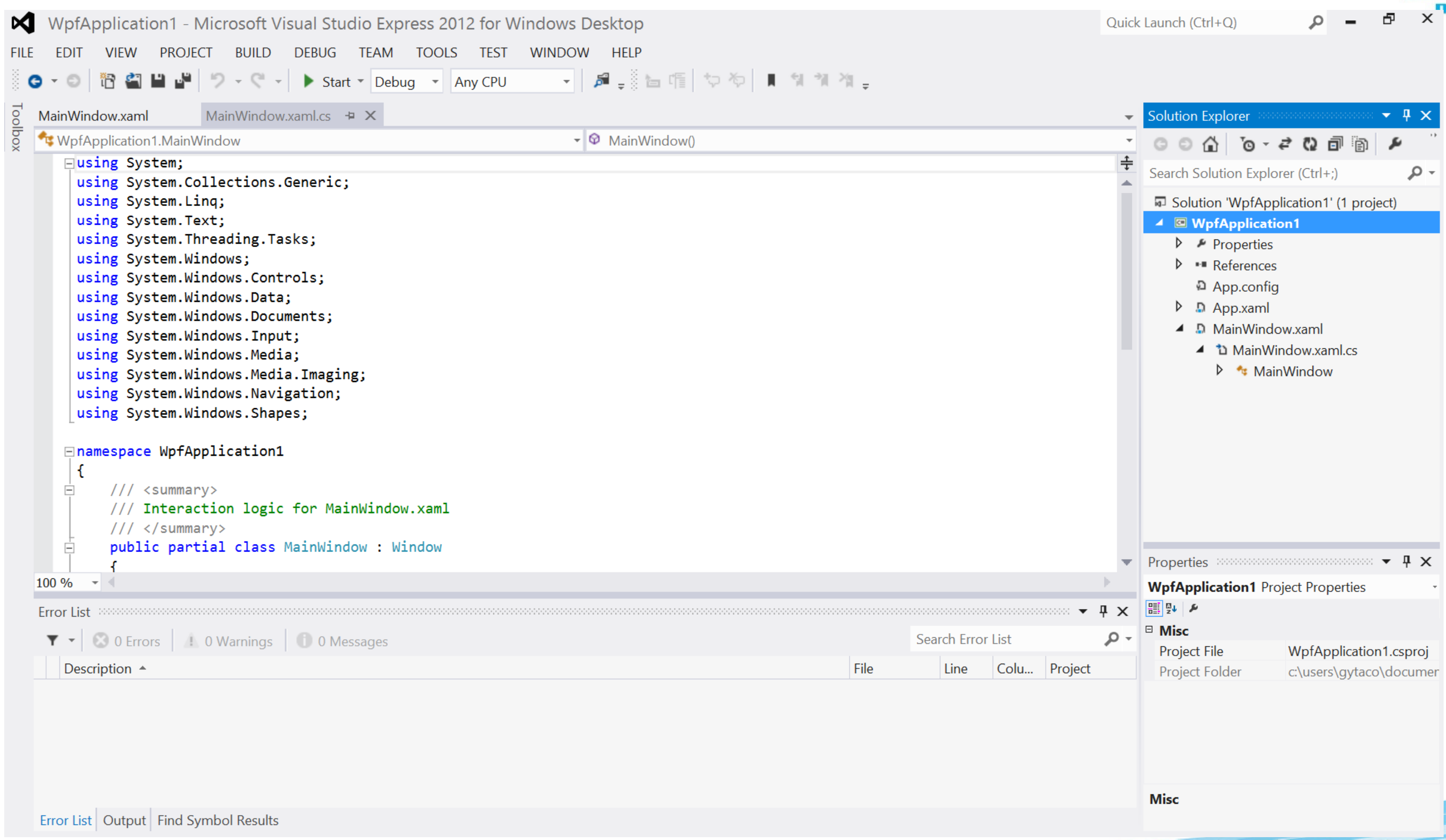
```
<Window x:Class="WpfApplication1.MainWindow"
        xmlns="http://schemas.microsoft.com/winfx/2006/xaml/presentation"
        xmlns:x="http://schemas.microsoft.com/winfx/2006/xaml"
        Title="MainWindow" Height="350" Width="525">
    <Grid>

    </Grid>
</Window>
```

Visual Studio Interface

Visual Studio Interface – Cheat Sheet

1. Toolbox – This is all the features we can add to our interface
2. This is our Main interaction UI either as a rendered interface as shown or as native C# code
3. XAML This is the code used to make the above rendered UI. This is only available in WPF Projects, and is a special version of XML.
4. This is our Solution Explorer or "Project Browser" like Revit this holds all our files, allows you to add new files, data connections, references and all things to do with our coding Project.
5. This is our Properties window, similar to "Property Palettes" in Revit if you click on certain objects contextual modifiers and options will be shown.
6. Error List A very important tool that will automatically highlight when there are errors or problems with your code before you attempt to compile them.
7. Output lists all the output our code makes on runtime, can be very useful for debugging or checking correct outputs.
8. Debugging tools allow us to run the code in debug mode and play it one line at a time and check data to ensure our tool is working correctly.



```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using System.Threading.Tasks;
using System.Windows;
using System.Windows.Controls;
using System.Windows.Data;
using System.Windows.Documents;
using System.Windows.Input;
using System.Windows.Media;
using System.Windows.Media.Imaging;
using System.Windows.Navigation;
using System.Windows.Shapes;

namespace WpfApplication1
{
    /// <summary>
    /// Interaction logic for MainWindow.xaml
    /// </summary>
    public partial class MainWindow : Window
    {
```

Project File	WpfApplication1.csproj
Project Folder	c:\users\gytaco\documer

More typical environment you will be working in.

Compiling & Debugging

Once we have our project running like now, we can compile it. This means the code is fed into the dotnet framework and turned into binary code in the form of a application .exe.

VS comes with the compiler built in which saves us a ton of trouble and messing around compared to other IDE's and OS's like Linux.

For the moment if you hit the start button VS will start to compile the program and if there are no errors you should a blank interface appear on your screen. You will notice that VS is still running in the background this is because we are running in debug mode and later we will see how it helps fix our applications.

When you close the application the debug mode will cancel and go back into editing mode.

Where’s my file??

So you compiled a project but where is your .exe??

Well we can find it by right clicking on the solution and going to the “Open Folder Location” option.

This will open a windows folder where you can see all your project files.

If you then go into the \bin\Debug are of the file you will see a number of files, however only the <projectname>.exe is needed and you can copy and share that with anyone and it will run on a windows PC.

SYNTAX RULES – Cheat Sheet

- The Language is Case sensitive
- Lines Terminate with Semi Colons
- Code is put into code blocks { }
- Comments start with //

- Declaring Variables
  DataType variableName = value;

- Variables must start with a letter
- Variables cannot contain Spaces
- Variables can contain Numbers
- Cannot contain symbols (except underscore)

References and Namespaces

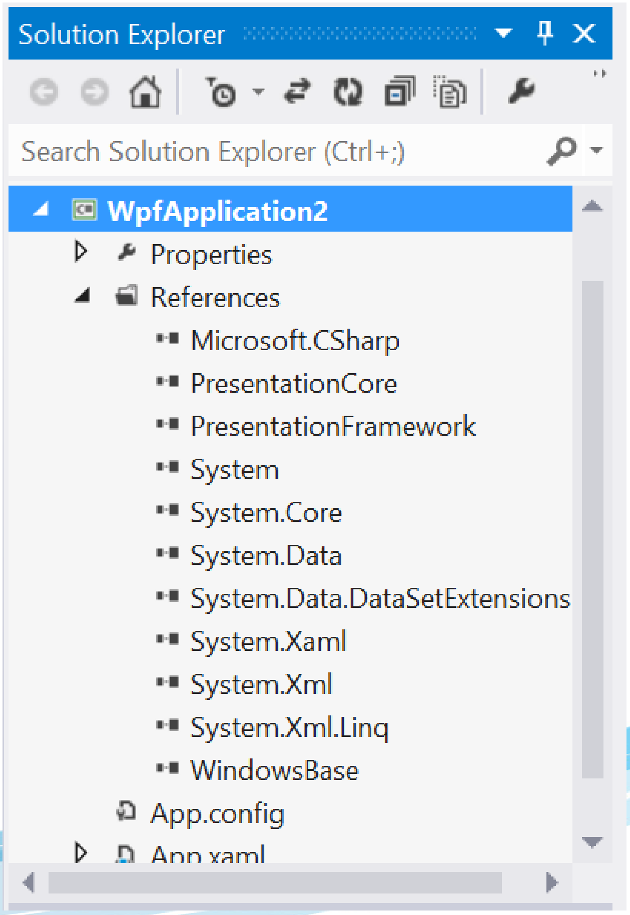
References are an important part of writing code.

References contain API commands or classes, methods and fields to create many windows objects. On our list from the left handside we can see there are multiple references each one containing some code that might control creating our interface or the C# language.

A Namespace is like an address for our code. Normally it is separated by a "."

For example my class Wall might be hidden in the address Autodesk.Revit.DB.Wall

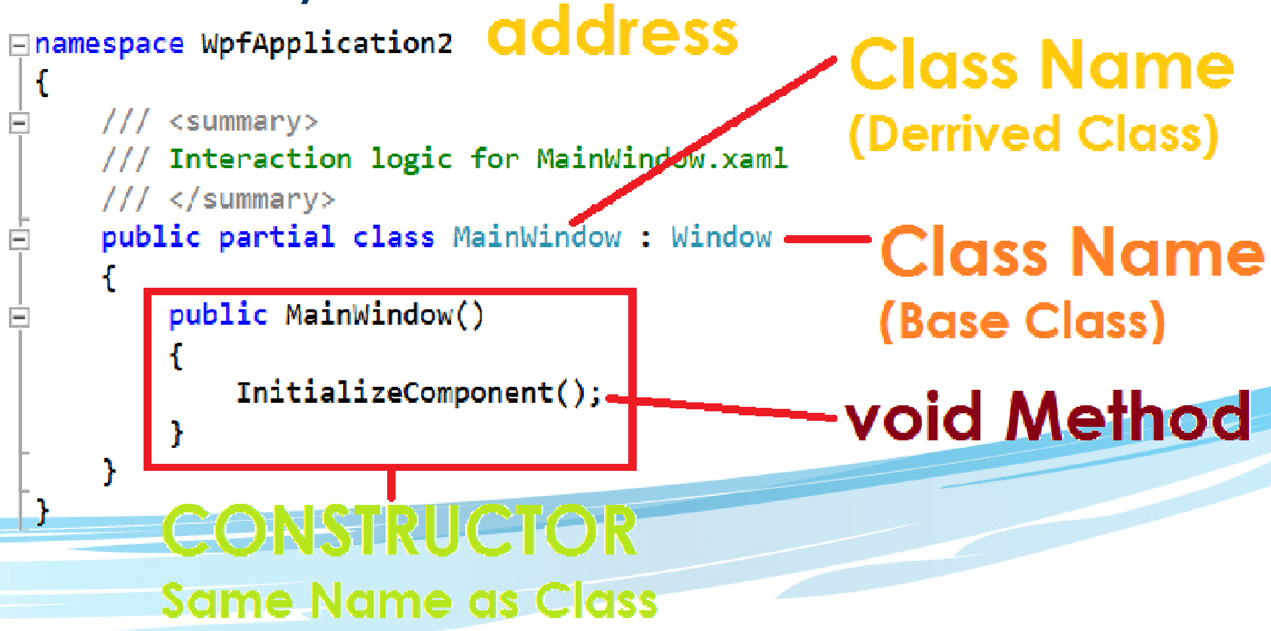
To save us writing out the address over and over again we can use the keyword "Using" to make an address shortcut in the start of our code.



WpfApplication2
- Properties
- References
  - Microsoft.CSharp
  - PresentationCore
  - PresentationFramework
  - System
  - System.Core
  - System.Data
  - System.Data.DataSetExtensions
  - System.Xaml
  - System.Xml
  - System.Xml.Linq
  - WindowsBase
- App.config
- App.xaml

Namespace, Classes, Constructor and Method

- Namespaces: are the start of your code, as you can see it's all code blocked { }
- Class Name (Derrived): This Class gets it's settings from the Base Class, plus you can make your own.
- Class Name (Base): This Class was created by Microsoft and comes with plenty of Constructors, Methods and Fields so we don't have to make our own.
- The name of the Constructor always matches the name of the Class
- Void Method we will discuss soon just be content to know it creates the UI in windows.



```
namespace WpfApplication2
{
    /// <summary>
    /// Interaction logic for MainWindow.xaml
    /// </summary>
    public partial class MainWindow : Window
    {
        public MainWindow()
        {
            InitializeComponent();
        }
    }
}
```

Public vs Private??

All this means is whether other classes can access your class, or only the class itself can access the methods contained within it.

As many objects in Revit or code are very similar therefore some classes need to be private as they may have the same names but affect different objects.

Calling a generic move comment might be bad, compared to calling the object Wall.Move();

This makes much more sense.

By default C# makes all fields, methods and classes private so if you need to you can specify public or private. It is good practice to specify private to others coders know you mean to keep that object private.

Methods

Methods are actions or manipulations in coding terminology.
They can edit, change, delete add information to fields.

There are three main Types;

void methodname() – Doesn't return a Value

static void methodname() – Class does not need to be initialised to be used.

void DataType methodname() – Requires the DataType value be returned

DataTypes

DataTypes are the fields or properties used in your code.

C# Built in ones include Integers(int), Text(string), Decimals(doubles) and True/False(bool).

string name = “Adam”; - Assigns the test “Adam” to the variable name.

int number = 1; - Assigns the number 1 to the variable name.

Double decimal; - Creates space in the memory for the variable decimal but assigns no value.

Bool yes = null; - Assigns a special value called null to the variable, this is simple an empty value.

From these users can create Custom values, an example would be an ElementId in Revit. One would assume this would be an int, but infact it’s a special DataType.

DataTypes can have their own methods and special features to be used correctly.

Sample Files & Lab Files

If you become lost, or the code doesn't work don't be discouraged.

Many of our Key steps will be saved as Ex1.0, Ex1.1, Ex1.2 and so on. Each heading will display the Exercise and Step we are up to.

You can find all these in a Folder in the C# Projects.

REMEMBER USE DEBUGGING AND STEP THROUGH THE CODE SLOWLY!!!

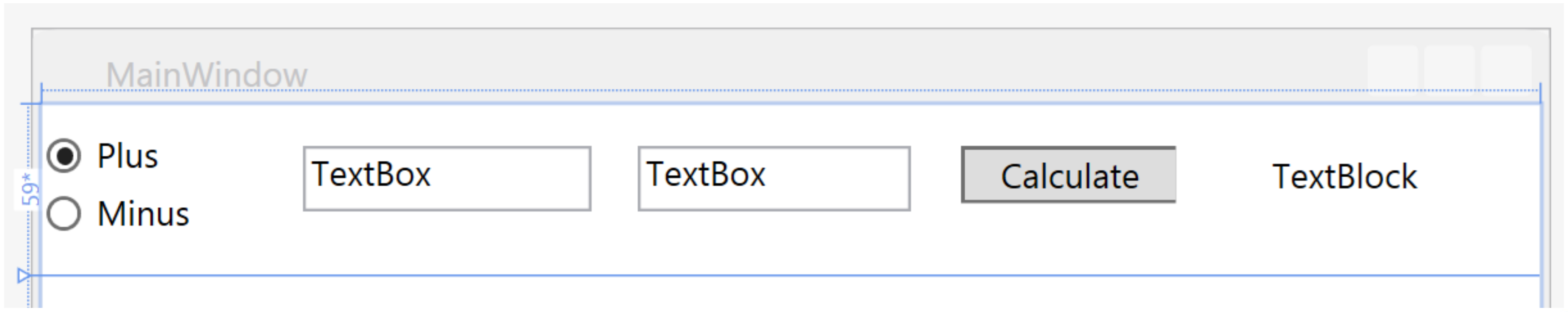
SHUT UP and lets code!! – Ex 1.0

Lets start by going back to the UI Screen.

We are going to create a simple addition/subtraction calculator inside our interface.

The Interface will take 2 entered values and output a fixed value when the user presses the “Calculate” Button.

To start setup the following items, you can find all these in the Toolbox Dialog.

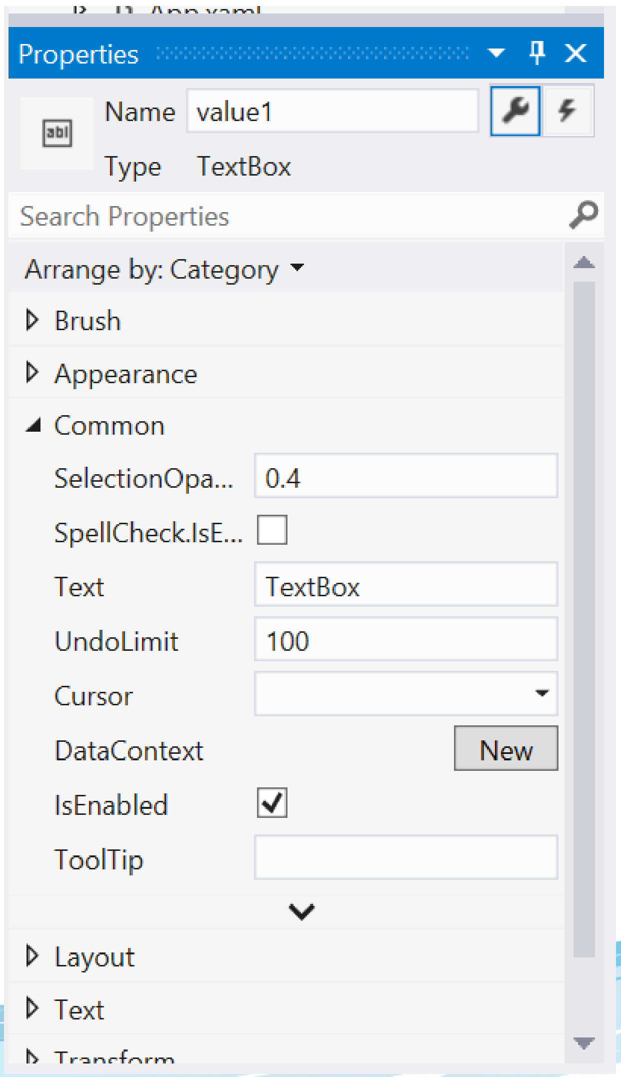


Naming Objects

When place an Object on your UI, when you select it you can change various attributes similar to accessing Contextual Tools in Revit.

The main thing we need to do here is name our Textbox, so in our code we know where our value will be stored.

In my code I call this value1 and I suggest you do to for consistency.



Properties

Name value1

Type TextBox

Search Properties

Arrange by: Category

Brush

Appearance

Common

SelectionOpa...	0.4
SpellCheck.IsE...	☐
Text	TextBox
UndoLimit	100
Cursor	
DataContext	New
IsEnabled	✓
ToolTip	

Layout

Text

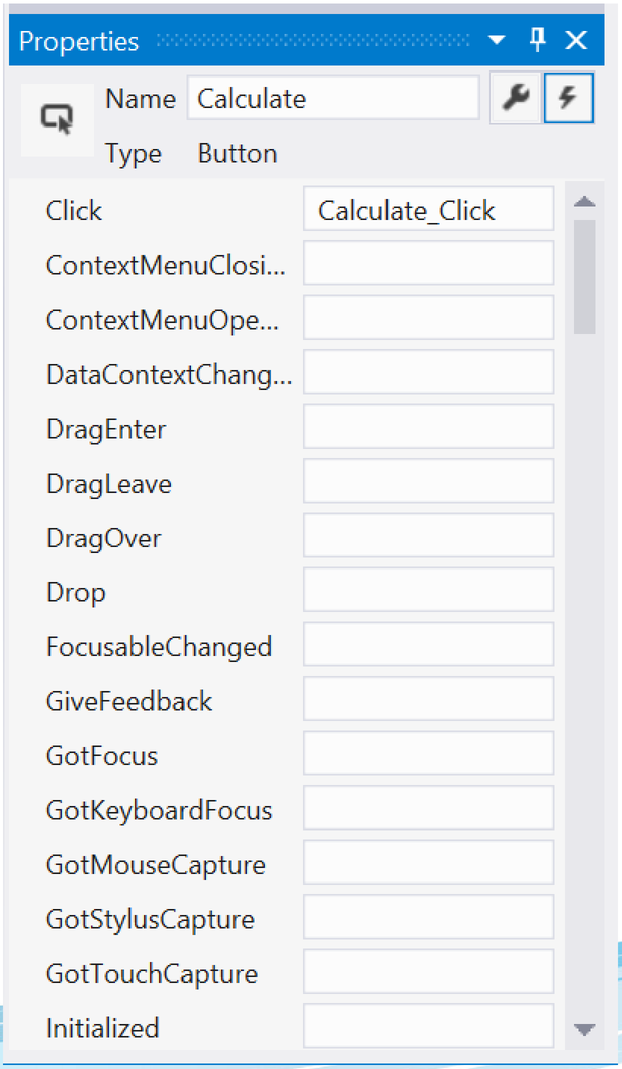
Transform

Naming Events

A Button is a special option as it doesn't store values, but it has a key feature we need!!

Many UI items have properties such as colours, sizes etc.. But if you click on the lightning bolt they also have what is called Events!!!!

Events are basically triggers for methods. In this case we need to name the button AND we need to name the event so we can code the instructions to your PC when a user clicks on a Button.



Properties

Name Calculate

Type Button

Click	Calculate_Click
ContextMenuClosi...	
ContextMenuOpe...	
DataContextChang...	
DragEnter	
DragLeave	
DragOver	
Drop	
FocusableChanged	
GiveFeedback	
GotFocus	
GotKeyboardFocus	
GotMouseCapture	
GotStylusCapture	
GotTouchCapture	
Initialized	

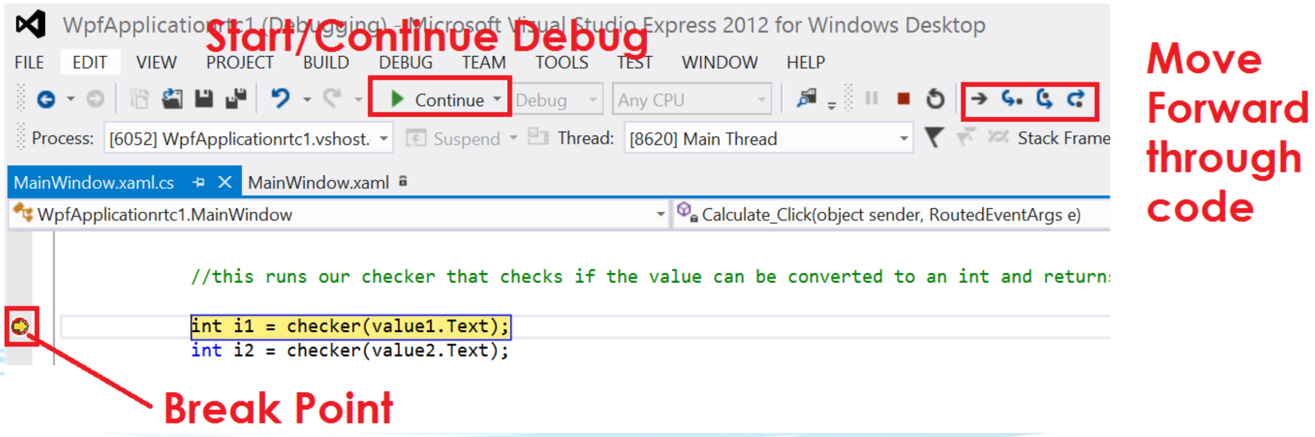
Debugging!!

A Button is a special option as it doesn't store values, but it has a key feature we need!!

Start/Continue:

Move through Code:

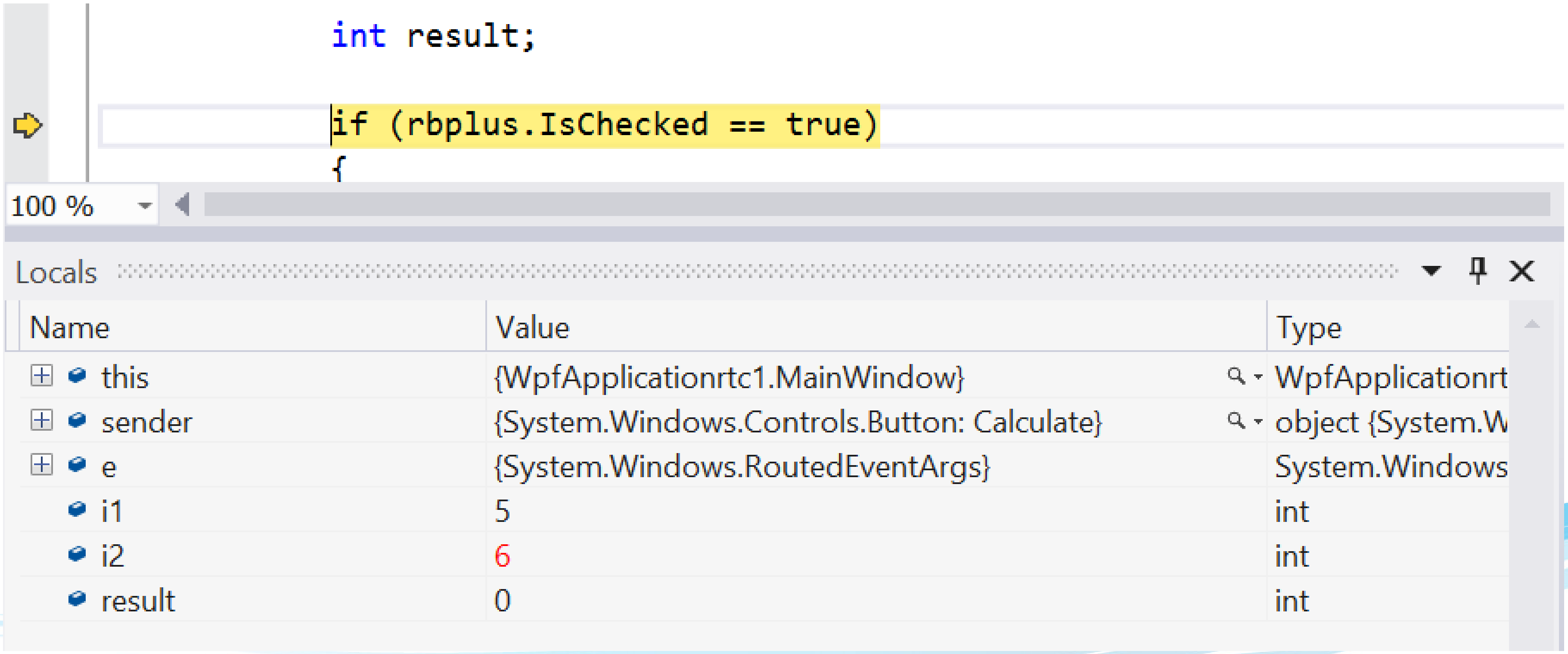
Breakpoints:


WpfApplicationrtc1 (Debugging) - Microsoft Visual Studio Express 2012 for Windows Desktop
Start/Continue Debug
FILE EDIT VIEW PROJECT BUILD DEBUG TEAM TOOLS TEST WINDOW HELP
Continue
Debug
Any CPU
Process: [6052] WpfApplicationrtc1.vshost.
Suspend
Thread: [8620] Main Thread
Stack Frame
MainWindow.xaml.cs
MainWindow.xaml
WpfApplicationrtc1.MainWindow
Calculate_Click(object sender, RoutedEventArgs e)
Move Forward through code
Break Point

```
            //this runs our checker that checks if the value can be converted to an int and return
            int i1 = checker(value1.Text);
            int i2 = checker(value2.Text);
```

Debugging!!

As you work through your code, you can track values, fields, classes and members your code is creating for debugging purposes.



```
            int result;

            if (rbplus.IsChecked == true)
            {
```

100 %

Locals

Name	Value	Type
this	{WpfApplicationrtc1.MainWindow}	WpfApplicationrt
sender	{System.Windows.Controls.Button: Calculate}	object {System.W
e	{System.Windows.RoutedEventArgs}	System.Windows
i1	5	int
i2	6	int
result	0	int

IntelliSense

Intellisense is one of the best features for VS Studio, it is very similar to predictor text on your phone, but rather the and guess words it shows the possible methods, statements and other things you can access with the Revit API.

To start you just need to type then use the arrow keys, everytime you are happy with a choice press TAB to move to the next word.

TIP: If you are unsure what options a method might have, press F12, this will take to a screen where you can see all the methods, Fields and options you can access for that class.

IF Statement

The IF statement is one of the most use options in both families and Coding with the Revit API.

It works exactly the same in Revit and API it’s just written differently.

Revit Example IF(Height <= 10, Height, 10)

API Way

```
if (Height <= 10)
{
 Height;
}
Else
{
Height = 10;
}
```

Method with Arguments

Methods are the most useful tool in an API methods are the actions items in a class which change or update or use information rather then simply storing it.

In Exercise 1 we used the Following Method.

int checker(string text)

DataType the Method returns, remember Void methods do not return values

Name of the my variable can be anything refer to naming conventions in your handout

Name(Arguments) Anything in these brackets are arguments that allow users to load parameters into the method, this allows methods to be used over and over in different situations

DataType of the argument for the first parameter in this case it requires a string

Name of the parameter this name is only used INSIDE the code blocks (In Scope)

{

//Code block that use the data in string text and returns an int

Return int; (ALL METHODS THAT ARE NOT VOID MUST RETURN THEIR DATATYPE VALUE

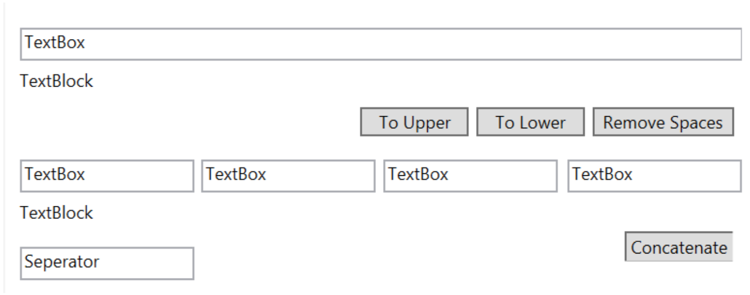
}

String Manipulation Tools

We are now going to go through the string manipulation tools and review a couple of examples, these can be used in any situations that require strings

Here is our interface it will do a few things

- Change input to upper case
- Change input to Lower case
- Remove the Spaces in our Text



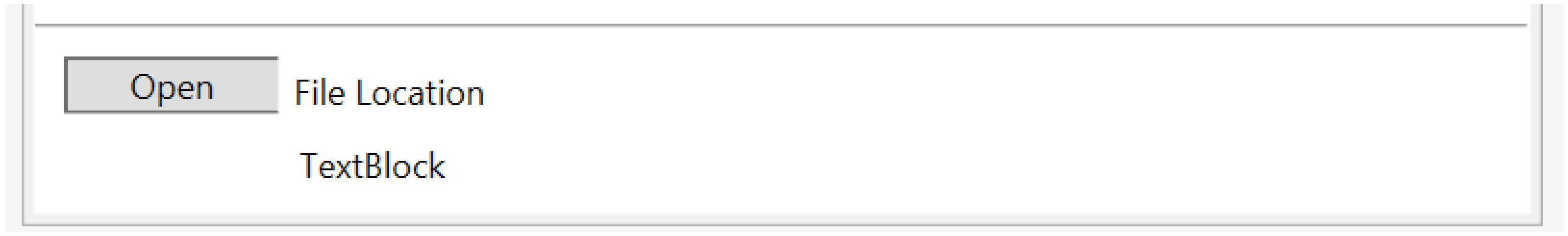
- The second option will allow us to enter multiple strings an concatenate them into a single string with a custom seperator

File Reader

Our last tool for our hello world is a file selection tool.

This will show users how to use the windows default selection dialog box, add filters and format requirements, then read information from the file.

Obviously this is extremely useful for allowing users to select files from within Revit such as excel, families or any other files that require use input.



REVIT RTC TECHNOLOGY CONFERENCE NORTH AMERICA 2014

Questions?