Visual Programming

CS411

Virtual University of Pakistan

Table Of Contents

Chapter 1

Lecture 1

This course is about graphical user interfaces and the event-driven model as applied to desktop, web, and mobile applications. It uses a ground up approach from what you already know.

The pre-requisites are C++ programming and data structures. I will introduce any new languages and concepts we are going to use.

This is a hands-on course. We will use in-class examples and you should focus on completing programming assignments to understand the material.

We will primarily use 2 books in this course. "Event processing in action" and "Windows resentation foundationunleashed". The first one is authored by Opher Etzion and Peter Niblett and is one of the very fewbooks on event processing. Event processing is often taught as a side-concept, but we will focus on it in this course. The other book is by Adam Nathan. It is about Windows presentation foundation or WPF in short. We use it as our gui example with C#. We will introduce C# concepts and WPF concepts. They are not a pre-requisite for this course.

Later in the course, we will touch event-driven programming in the browser, in particular made possible by techniques called AJAX. Again not a pre-requisite. I will introduce all the concepts in class. Towards the very end, we will discuss event driven programming on mobiles. Again no pre-requisite. I will introduce all the tools involved.

So what is Visual Programming and what you will learn in this course. Its primarily event driven concepts and application of these concepts on GUI programs on desktop, web, and mobile.

Here is our first example:

```
Using namespace std;

#include <iostream>

Int main()
{
  Char a;
  Do {
    A = cin.get();
    Cout << a;
  } while (a!='x');
  Return 0;
}
```

Let's compile and run using the commands

G++ -o example1 example1.cc ./example1

Our second example

```
#include <iostream>
#include <fstream>
```

```
Int main()
{
  Char a = '-';
  Do {
    Std::ifstream f("test.txt", std::ifstream::in);
    If (f.good() && a != f.peek()) {
      A = f.get();
      Std::cout << a << std::endl;
    }
    F.close();
  } while (a!='x');
  Return 0;
}
```

Let's compile and run it. There are a few issues like busy wait etc. But we'll leave them on a side for now.

So why we saw these two very basic examples. Well, they were both very easy to write. With an OR.

```
#include <iostream>
#include <fstream>

Int main()
{
  Char a = '-', b;
  Do {
    If (iskeyboardpressed()) {
      B = std::cin.get();
      Std::cout << b << std::endl;
    }
    Std::ifstream f("test.txt", std::ifstream::in);
    If (f.good() && a != f.peek()) {
      A = f.get();
      Std::cout << a << std::endl;
    }
    F.close();
  } while (a!='x' && b!='x');
  Return 0;
}
```

I.e. We react to either of the two actions. Here is how we can do it.

```
#include  <iostream>
#include<fstream>
 Int main()
 {
 Char a  =’-’, b;
 Do{
 If(iskeyboardpressed())  {
 B = std::cin.get();
 Std::cout  << b  << std::endl;
 }
 Std::ifstream f("test.txt", std::ifstream::in);
 If(f.good() && a  != f.peek())  {
 A = f.get();
 Std::cout  << a  << std::endl;
 }
```

```
F.close();
} while  (a!=’x’ && b!=’x’);
Return  0;
}
```

Or we could re-factor it into a cleaner version.

```
#include <iostream>
#include <fstream>

Int onkeypress()
{
  Char b = std::cin.get();
  Std::cout << b << std::endl;
  Return b == 'x';
}

Int onfilechanged()
{
  Char a = f.get();
  Std::cout << a << std::endl;
  Return a == 'x';
}

Int main()
{
  Char a = '-', b;
  Do {
    If (iskeyboardpressed()) {
      If (onkeypress())
Return 0;
    }
    Std::ifstream f("test.txt", std::ifstream::in);
    If (f.good() && a != f.peek()) {
      If (onfilechanged())
Return 0;
    }
    F.close();
  } while (true);
  Return 0;
}
```

So what are the pros and cons of the re-factored approach. We do one thing at a time (more focused approach). Blocking is a huge issue. Nothing is parallel. The key idea is that once you understand the approach, it scales really well to a large number of things you want to respond to.

Next time we’ll define events, event loops, and different terms in the event driven model.

Chapter 2

Lecture 2

Last time, we saw some examples and demonstrated the idea of event driven design. We can re-factor the last example even one step further like this:

```
#include <iostream>
#include <fstream>

Int onkeypress()
{
  Char b = std::cin.get();
  Std::cout << b << std::endl;
  Return b == 'x';
}

Int onfilechanged()
{
  Char a = f.get();
  Std::cout << a << std::endl;
  Return a == 'x';
}

Int main()
{
  Char a = '-', b;
  Do {
    If (iskeyboardpressed()) {
      If (onkeypress())
 Return 0;
    }
    Std::ifstream f("test.txt", std::ifstream::in);
    If (f.good() && a != f.peek()) {
      If (onfilechanged())
 Return 0;
    }
    F.close();
  } while (true);
  Return 0;
}
```

So what is an event? It's an occurrence within a particular system or domain. There are two meanings: something that happened and the corresponding detection in computer world. An event captures "some" things from the actual occurrence and multiple events may capture "one" occurrence.

Probabilistic events may or may not relate to an actual occurrence e.g. A fraud detection event on a banking transaction. Every event is represented by an event-object. There are various types of events and information in the event describes details of the particular event type. E.g. Key press, file event etc.

Let's take a coffee shop analogy. Events are taking place all the time. Let's understand synchronous and asynchronous behavior with this example. So if an order is completed, coffee ready, pastery heated before the next order that's synchronous behavior. There are observed events and deduced events. Situation is

an event ocurrence that requires a reaction.

Let's take example of events in computing systems. Interrupts and exceptions on a computer e.g. Divide by zero. Patient monitored by sensors. Car sensors alerting to oil or pressure situations. Banking alerts. Road tolling. Etc. Event processing is computing that performs operations on events. Common event processing operations include reading, creating, transforming, and deleting events. The design, coding and operation of applications that use events, either directly or indirectly is called event-based programming or applications based on event-driven architecture. So what if we don't use event-driven programming. We will poll for events. Even if you make your application non-event-driven, you still wait for events. Wait for a single event is blocking operation.

Synchronous operations are completed before the next operation can be started. Asynchronous operations can be started and we can do something else before they are completed.

Why do we want our applications to be event based. They are easier to scale. Well suited to Visual programming where multiple GUI elements and many sources of events exist. It has a direct mapping to real world. Its deterministic-there is an exact mapping between a situation in the real world and its representation in the event processing system.

At the same time its approximate-the event processing system provides an approximation to real world events. So what kind of events are there in non-real time applications. Mouse and keyboard, secondary events of GUI elements, file events, message based parallel and local communication, network events etc.
Here is flower delivery example from the book.

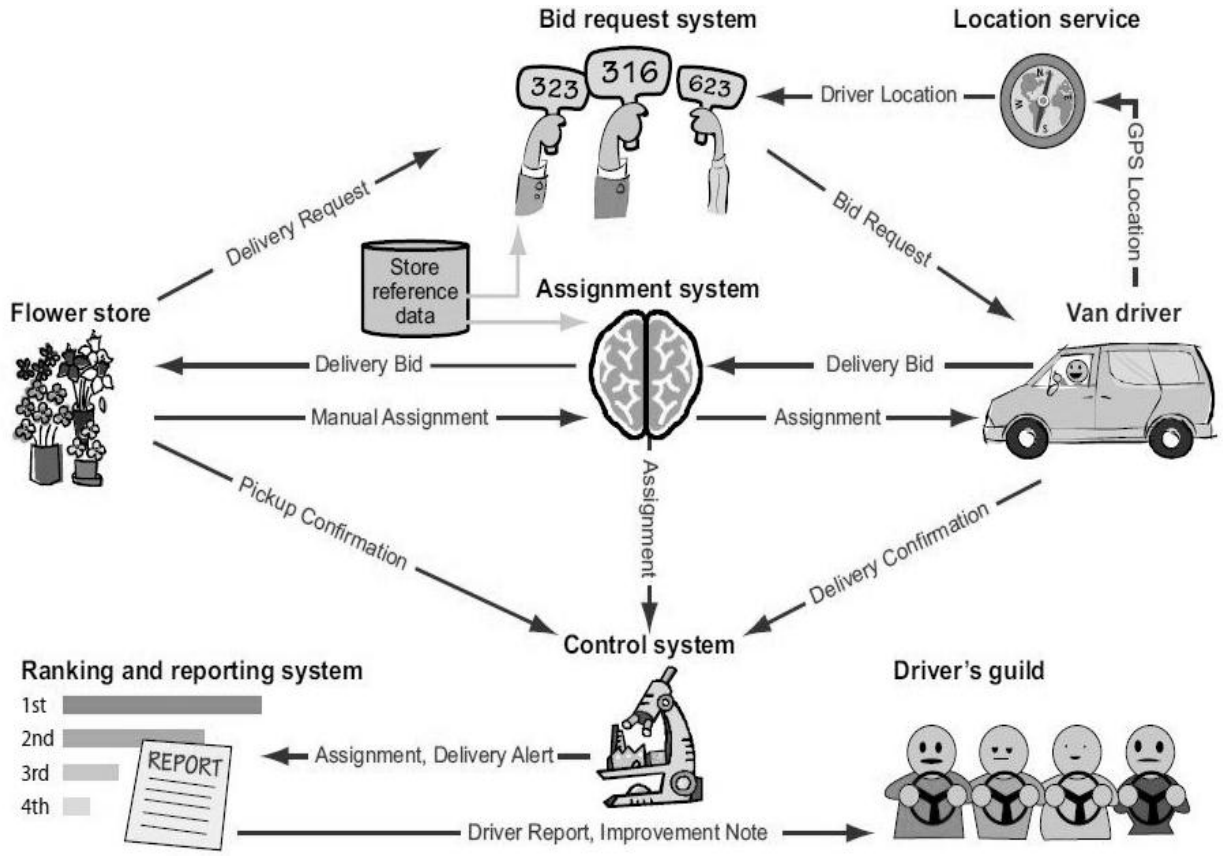


Figure 1.6 Parts of the Fast Flower Delivery application. The arrows represent the flow of events, and the pictures represent the various entities involved in the network.

Here are some excerpts of its description from the book. A consortium of flower stores in a large city has established an agreement with local independent van drivers to deliver flowers from the stores to their destinations. When a store gets a flower delivery order, it creates a request which is broadcast to relevant drivers within a certain distance from the store, with the time for pickup (typically now) and the required delivery time. A driver is then assigned and the customer is notified that a delivery has been scheduled. The driver makes the pickup and delivery, and the person receiving the flowers confirms the delivery time by signing for it on the drivers mobile device.

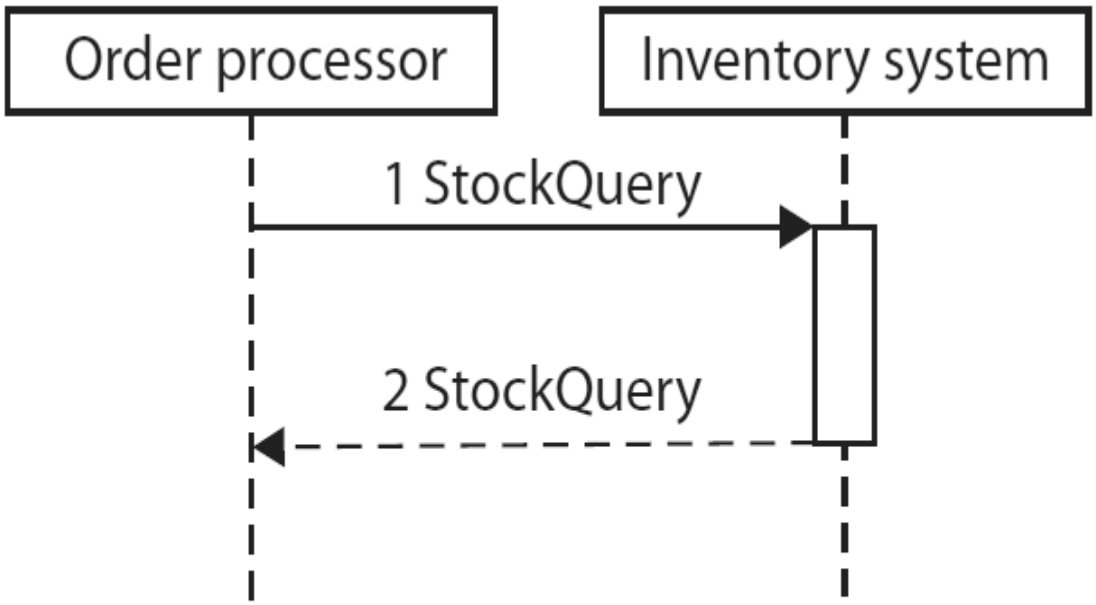
The system maintains a ranking of each individual driver based on his or her ability to deliver flowers on time. Each store has a profile that can include a constraint on the ranking of its drivers; for example, a store can require its drivers to have a ranking greater than 10. The profile also indicates whether the store wants the system to assign drivers automatically,or whether it wants to receive several applications and then make its own choice. A permanently weak driver is a driver with fewer than five assignments on all the days on which the driver has been active.

An idle driver is a driver with at least one day of activity that had no assignments. A consistently weak driver is a driver whose assignments, when active, are at least two standard deviations lower than the average assignments per driver on that day. A consistently strong driver is a driver whose daily assignments are at least two standard deviations higher than the average number of assignments per driver on each day in question. An improving driver is a driver whose assignments increase or stay the same day by day.

Chapter 3

Lecture 3

Let's discuss request and response based applications. Example is a web browser which sends queries and updates. It performs synchronous interactions. It looks like this:



Events are based on the principle of decoupling. Let's compare events and request-response based architecture. Events have already happened whereas requests are asking something to happen. Service requester or client contacts the service provider or the server. In event-driven architecture event producer sends event to event consumer. A customer order can be represented as an event or a request: what are the benefits of each approach?

Similarly, should we send a request to query the state or use "change events". Event has meaning independent of its producers and consumers whereas the same cannot be said about requests. Events are often one way (push events). Events decouple producers and consumers. Events can be processed asynchronously. There can be more than one consumer, each processing it differently possibly processing a sequence of events that happened over time. Latency is reduced in comparison to the "pull" style.

Events can also be compared to messages. Messages may contain events. Events may live outside events like in event logs.

An event channel is a subscription mechanism for events. It further decouples consumers and producers. Can even be an intermediate XML file to store events.

Event-driven architecture and service-driven architecture can also be compared. Event-based programming, also called event-driven architecture (EDA) is an architectural style in which one or more components in a software system execute in response to receiving one or more event notifications. Service-oriented architecture (SOA) is built from request-response. It moves away from monolithic applications. There are

many similarities and often a component can provide both modes of contacting it.
Here are some definitions from the book.

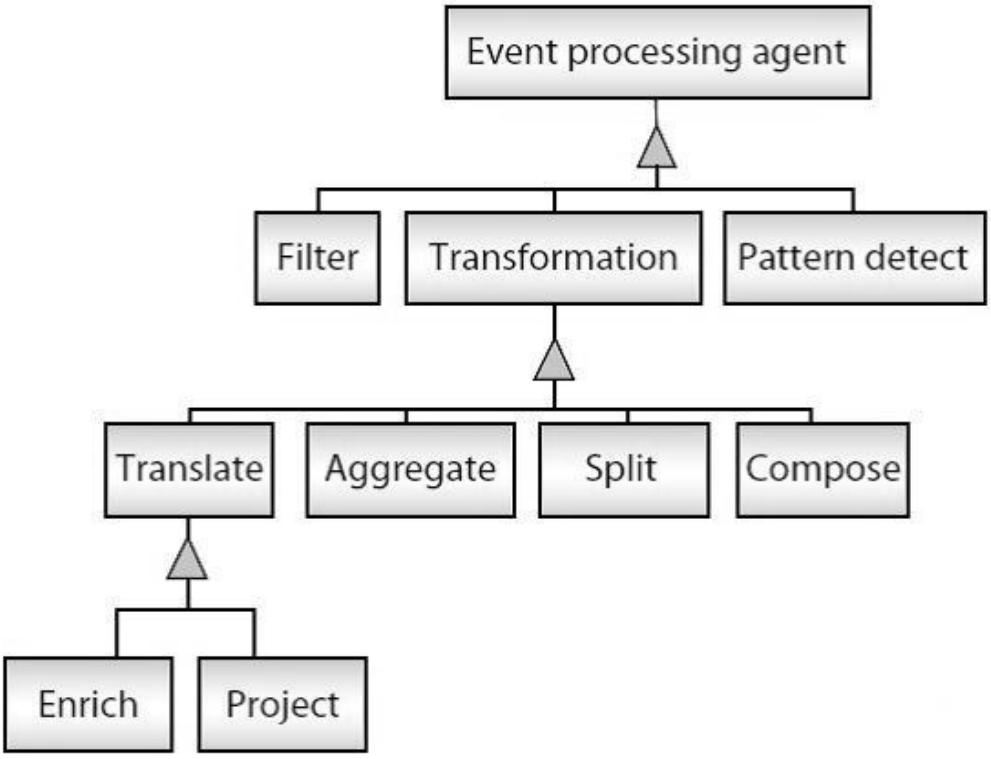
EVENT PRODUCER: An event producer is an entity at the edge of an event process- ing system that introduces events into the system.

EVENT CONSUMER: An event consumer is an entity at the edge of an event pro- cessing system that receives events from the system.

RAW EVENT: A raw event is an event that is introduced into an event processing system by an event producer. DERIVED EVENT: A derived event is an event that is generated as a result of event processing that takes place inside an event processing system.

STATELESS EVENT PROCESSING An event processing agent is said to be stateless if the way it processes one event does not influence the way it processes any subsequent events.

EVENT STREAM An event stream (or stream) is a set of associated events. It is often a temporally totally ordered set (that is to say, there is a well-defined timestamp-based order to the events in the stream). A stream in which all the events must be of the same type is called a homogeneous event stream; a stream in which the events may be of different types is referred to as a heterogeneous event stream.
Here are types of event agents.



Looking back at the program discussed before, can you map the event definitions to the code below:

```
#include <iostream>
#include <fstream>

Int filechanged = 0;
```

```
Char lastchar = '-';

Int onkeypress()
{
  Char b = std::cin.get();
  Std::cout << b << std::endl;
  Return b == 'x';
}

Int onfileread()
{
  Char a = f.get();
  If (a != lastchar) {
    Filechanged = 1;
    Lastchar = a;
  }
  Return 0;
}
Int onfilechanged(char a)
{
  Std::cout << a << std::endl;
  Return a == 'x';
}
Int main()
{
  Do {
    If (kbhit()) {
      If (onkeypress())
 Return 0;
    }
    Std::ifstream f("test.txt", std::ifstream::in);
    If (f.good()) {
      If (onfileread(f.get()))
 Return 0;
    }
    F.close();
    If (filechanged) {
      If (onfilechanged())
 Return 0;
    }
  } while (true);
  Return 0;
}
```

What aspects of real occurrence do event objects capture. An event type is a specification for a set of event objects that have the same semantic intent and same structure; every event object is considered to be an instance of an event type. An event attribute is a component of the structure of an event. Each attribute has a name and a data type. It can also have occurrence/detection time, certainty, source, location. Events can be composed, generalized, and specialized. Let's experiment with flower delivery example. Can you find events and their attributes.

Event producers can produce events that are hardware generated, software generated, or from human interaction. Event consumers have similar types. Examples are locking or unlocking a door, raising or lowering a barrier, applying the brakes on a vehicle, opening or closing a valve, controlling a railroad switch, and turning a piece of equipment on or off. Try to find producers and consumers in flower delivery application.

Chapter 4

Lecture 4

We will get introduced to C# in this lecture. Before that, Let's start with some history of Java and J++. Microsoft wanted to extend Java to communicate it with COM. Sun did not want that as it would make Java platform dependent. Microsoft wanted a clean-room implementation of Java. What is a clean-room implementation?

Clean room design (also known as the Chinese wall technique) is the method of copying a design by reverse engineering and then recreating it without infringing any of the copyrights and trade secrets associated with the original design. Clean room design is useful as a defense against copyright and trade secret infringement because it relies on independent invention. However, because independent invention is not a defense against patents, clean room designs typically cannot be used to circumvent patent restrictions.

Where do the name for C# comes from. The initial name was "Cool", which stood for "C-like Object Oriented Language". Whats the difference between .NET and C#. C# design most directly reflects .NET (CLR) design but C# is a language and .NET is the platform.

As a first step, you should get Visual Studio from http://www.microsoft.com/visualstudio. Click Downloads and choose "Visual Studio Express 2012 for Windows Desktop". Choose "Download now" or "Install now". You can save iso file to disk (right click to mount in win8) otherwise there are free utilities to mount it or burn it on a cd.

Install and register online for continued use. To create a new project choose File, New Project, Visual C#, Console Application. Then click the Start key.
Here are some notable features of C#.

- No global variables or functions.
- Locals cannot shadow global variables.
- There is a strict boolean type.
- Memory address pointers can only be used in specifically marked "unsafe" blocks and require permissions
- No instruction to "free" memory. Only garbage collection.
- Try-finally block.
- No "multiple inheritance" but interfaces supported.
- Operator overloading allowed.
- More type-safe (only integer widening allowed).
- Enumeration members are scoped.
- Property syntax for getters and setters.
- No checked exceptions.
- some functional programming features like function objects and lambda expressions.

The common type system of C# has value types and reference types. Instances of value types do not have referential identity nor referential comparison semantics. Value types are derived from System.valuetype and can always be created, copied and have a default value (int, float, char, System.datetime, enum, struct). In contrast, reference types have the notion of referential identity. Default equality and inequality comparisons test for referential rather than structural equality unless overloaded e.g System.String. Not "always" possible to create an instance of a reference type, copy an existing instance, perform a value comparison on two instances, though specific types "can" provide such services by exposing a public constructor or implementing a corresponding interface e.g. Object, System.String, System.Array.

Here is how to box and unbox variables in C#. Boxing stores a value type in a reference type.

```
Int foo = 42;// Value type.
Object bar = foo;// foo is boxed to bar.

Int foo2 = (int)bar;// Unboxed back to value type.
```

Here is an example of Generics in C# which are like templates in C++.

```
Public class genericlist<T>
{
 Void Add(T input) { }
}
Class  testgenericlist
{
 Private class exampleclass{ }
 Static void Main()
 {
 // Declare a list of type int.
 Genericlist<int> list1= new genericlist<int>();
 // Declare a list of type string.
 Genericlist<string> list2= new genericlist<string>();
 // Declare a list of type exampleclass.
 Genericlist<exampleclass> list3 = new genericlist<exampleclass>();
 }

   }
```

Pre-processor directives like #if, #else, #endif, #region, #endregion are supported. Comments are written using // and /* */.

It also provides an XML documentation system that is used like this.

```
Public class Foo{
/**<summary>A summary of the method.</summary>
* <param name="firstparam">A description of the parameter.</param>
* <remarks>Remarks about the method.</remarks> */
Public static void Bar(int firstparam){}

   }
```

The C# specification details a minimum set of types and class libraries that the compiler expects to have available. In practice, C# is most often used with some implementation of the Common Language Infrastructure (CLI), which is standardized as ECMA-335 Common Language Infrastructure (CLI).

Here is a hello world program in C#.

```
Using System;
Class Program {
```

```
    Static void Main() {
        Console.writeline("Hello world!");
    }
}
```

Writing the using clause uses System as a candidate prefix for types used in the source code. When the compiler sees use of the Console type later in the source code. It allows the programmer to state all candidate prefixes to use during compilation instead of always using full type names. Console is a static class in the System namespace.

If you run the program from within Visual Studio and the console window disappears quickly you can use the following code.

```
1 // keep screen from going  away
2 // when run from VS. NET
3 Console.readline();
```

To test use csc.exe Welcome.cs or run from within Visual Studio. Now Let's write a GUI based hello world.

```
Using System.Windows.Forms;

Class Program
{
    Static void Main()
    {
        Messagebox.Show("Hello world!");
    }

  }
```

Here is a hello world with a call to Console.Write with an argument.

```
// Namespace Declaration
Using System;

// Program start class
Class interactivewelcome
{
 // Main begins program execution.
 Public static void Main()
 {
 // Write to console/get input
Console.Write("What is your name?: ");
Console.Write("Hello, {0}! ", Console.readline());
Console.writeline("Welcome to the C\# Station Tutorial!");
 }
}
```

Chapter 5

Lecture 5

Here is an example with boolean types.

```
Using System;

Class Booleans
{
Public static void Main()
 {
Bool content = true;
Bool nocontent = false;

Console.writeline("It is {0} that C\# Station provides C\# programming language
content.", content);
Console.writeline("The statement above is not {0}.", nocontent);
 }
}
```

Here are the integer types in C#.

Type	Size (in bits)	Range
sbyte	8	-128 to 127
byte	8	0 to 255
short	16	-32768 to 32767
ushort	16	0 to 65535
int	32	-2147483648 to 2147483647
uint	32	0 to 4294967295
long	64	-9223372036854775808 to 9223372036854775807
ulong	64	0 to 18446744073709551615
char	16	0 to 65535

And here are the floating point and decimal types in C#.

Type	Size (in bits)	precision	Range
float	32	7 digits	1.5×10^{-45} to 3.4×10^{38}
double	64	15-16 digits	5.0×10^{-324} to 1.7×10^{308}
decimal	128	28-29 decimal places	1.0×10^{-28} to 7.9×10^{28}

The System.String type supports the following sequences.

Char	Meaning	Value
\'	Single quote	0x0027
\"	Double quote	0x0022
\\	Backslash	0x005C
\0	Null	0x0000
\a	Alert	0x0007
\b	Backspace	0x0008
\f	Form feed	0x000C
\n	New line	0x000A
\r	Carriage return	0x000D
\t	Horizontal tab	0x0009
\v	Vertical tab	0x000B

The escape character is while @ is the verbatim character. The following operators are supported by C#.

Category (by precedence)	Operator(s)	Associativity
Primary	x.y f(x) a[x] x++ x-- new typeof default checked unchecked delegate	left
Unary	+ - ! ~ ++x --x (T)x	right
Multiplicative	* / %	left
Additive	+ -	left
Shift	<< >>	left
Relational	< > <= >= is as	left
Equality	== !=	right
Logical AND	&	left
Logical XOR	^	left
Logical OR	|	left
Conditional AND	&&	left
Conditional OR	||	left
Null Coalescing	??	left
Ternary	?:	right
Assignment	= *= /= %= += -= <<= >>= &= ^= |= =>	right

Here is an example of using the Array type in C#.

```
Using System;
Class Array{
 Public static void Main(){
    Int[] myints = { 5, 10, 15 };
    Bool[][] mybools = new bool[2][];
    Mybools[0] = new bool[2];
    Mybools[1] = new bool[1];
    Double[,] mydoubles = new double[2, 2];
    String[] mystrings = new string[3];
    Console.writeline("myints[0]: {0}, myints[1]: {1}, myints[2]: {2}", myints[0],
myints[1], myints[2]);
    Mybools[0][0] = true;
    Mybools[0][1] = false;
    Mybools[1][0] = true;
    Console.writeline("mybools[0][0]: {0}, mybools[1][0]: {1}", mybools[0][0],
mybools[1][0]);
    Mydoubles[0, 0] = 3.147;
    Mydoubles[0, 1] = 7.157;
    Mydoubles[1, 1] = 2.117;
    Mydoubles[1, 0] = 56.00138917;
    Console.writeline("mydoubles[0, 0]: {0}, mydoubles[1, 0]: {1}", mydoubles[0, 0],
mydoubles[1, 0]);
    Mystrings[0] = "Joe";
    Mystrings[1] = "Matt";
    Mystrings[2] = "Robert";
    Console.writeline("mystrings[0]: {0}, mystrings[1]: {1}, mystrings[2]: {2}",
mystrings[0], mystrings[1], mystrings[2]);
 }
}
```

You can make jagged arrays or multi-dimensional arrays. Jagged is basically array of arrays. Array size is any integer type value. It uses a zero-based index.

Control statements like if and else work much like C++.

```
If (myint < 0 || myint == 0) {

Console.writeline("Your number {0} is less than or equal to zero.", myint);

}  else  if (myint > 0 && myint <= 10){

Console.writeline("Your number {0} is in the range from 1 to 10.", myint);

}  else  if (myint > 10 && myint <= 20){

Console.writeline("Your number {0} is in the range from 11 to 20.", myint);

} else if (myint > 20 && myint <= 30){

Console.writeline("Your number {0} is in the range from 21 to 30.", myint);

} else{

Console.writeline("Your number {0} is greater than 30.", myint); }
```

Switch statement can work for booleans, enums, integral types, and strings. You break the switch statement

Branching statement	Description
break	Leaves the switch block
continue	Leaves the switch block, skips remaining logic in enclosing loop, and goes back to loop condition to determine if loop should be executed again from the beginning. Works only if switch statement is in a loop as described in Lesson 04: Control Statements - Loops.
goto	Leaves the switch block and jumps directly to a label of the form "<labelname>:"
return	Leaves the current method. Methods are described in more detail in Lesson 05: Methods.
throw	Throws an exception, as discussed in Lesson 15: Introduction to Exception Handling.

with one of the following branching statements:
The following four kinds of loops are available.

While (<boolean expression>) { <statements> }
Do { <statements> } while (<boolean expression>);
For (<initializer list>; <boolean expression>; <iterator list>) { <statements> }
foreach (<type> <iteration variable> in <list>) { <statements> }

Here is an example of a while loop:

```
Using System;
Class whileloop
{
    Public static void Main()
    {
        Int myint = 0;

        While (myint < 10)
        {
            Console.Write("{0} ", myint);
            Myint++;
        }
        Console.writeline();
    }
}
```

Another example with a for loop.

```
Using System;
Class forloop
{
    Public static void Main()
    {
        For (int i=0; i < 20; i++)
        {
            If (i == 10)
                Break;

            If (i % 2 == 0)
```

```
                Continue;
            Console.Write("{0} ", i);
        }
        Console.writeline();
    }

}
```

And one of foreach loop.

```
Using System;
Class foreachloop
{
    Public static void Main()
    {
        String[] names = {"Cheryl", "Joe", "Matt", "Robert"};

        Foreach (string person in names)
        {
            Console.writeline("{0} ", person);
        }
    }

}
```

Methods are of the form.

Attributes modifiers return-type method-name(parameters) { statements }

Elements are accessed using the dot operator. Object variables are references to the original object not the object themselves. Here is an example of a method.

```
Using System;
Class onemethod
{
    Public static void Main()
    {
        String mychoice;
        Onemethod om = new onemethod();
            Mychoice = om.getchoice();
    }
    String getchoice()
    {
        Return “example”;
    }

}
```

The “this” pointer in methods refers to the object on which the method is called. Parameters can be “ref” parameter which are passed by reference, “out” parameters which are used for return values or the “params” argument for variable arguments.

Here is an example of the last:

```
// show  addresses
Void viewaddresses(params string[] names)
{
 Foreach (string name in names){
 Console.writeline("Name: {0}", name);
 }
}
```

Chapter 6

Lecture 6

Namespaces allow name reuse. For example, a Console class can reside in multiple libraries. Here is an example of namespace declaration:

```
// Namespace Declaration

Using System;

// The C\# Station Namespace

Namespace csharp_station
{
// Program start class
Class namespacecss  {
 // Main begins program execution.

Public static void Main()
 {

// Write to console

  Console.writeline("This is the new C\# Station Namespace.");

}

 }

}
```

Nested namespaces can be used like this:

Namespace testnamespace {

Namespace tutorial { OR

Namespace testnamespace.tutorial {

Calling this namespace from within testnamespace we use "tutorial.". The using directive can be used to rename a long namespace in the current file like:

Using theexample = testnamespace.tutorial.myexample

Class use the word class followed by name, curly braces and the class body. It has constructor (ctor) to create objects. Ctor does not return any values and it initializes class members.

```
// Namespace Declaration
Using System;
// helper class
Class outputclass
{
String mystring;

// Constructor
```

```
Public outputclass(string inputstring)
    {

        Mystring = inputstring;
    }

// Instance Method

Public void printstring()
    {
        Console.writeline("{0}", mystring);

    }

// Destructor

~outputclass()
    {

        // Some resource cleanup routines

    }

}

// Program start class

Class exampleclass
{
    // Main begins program execution.

Public static void Main()
    {

    // Instance of outputclass
    Outputclass outcl = new
    Outputclass("This is printed by the output class.");
    // Call Output class' method
    Outcl.printstring();
    }

}
```

Multiple classes are conventionally stored in multiple files. To create a new class in IDE, use the new class wizard. Multiple .cs files can be compiled as csc a.cs b.cs c.cs. Default ctors are written with no arguments when no ctor is written. An initializer list can be used to use an alternate constructor.

Public outputclass() : this("Default Constructor String") { }

Multiple ctors can be written. Types of class members in C# are instance and static. For every new occurrence there are new instance vars.

Outputclass oc1 = new outputclass("outputclass1");

Outputclass oc2 = new outputclass("outputclass2");

Now oc1.printstring or oc2.printstring will access different instance variables. On the other hand static members have only one copy.

```
Public static void staticprinter(){
 Console.writeline("There is only one of me.");

}
```

Then you could call that function from Main() like this: outputclass.staticprinter();

Static ctor exists to initialize class static members. Its called only once. It has no parameters. Destructors (dtors) are called by garbage collector. "public" methods accessed outside class. "public" classes accessed outside assembly. A class can contain Constructors, Destructors, Fields, Methods, Properties, Indexers, Delegates, Events, and Nested Classes.

Inheritance introduces the concept of base classes and derived classes. Declaration of inheritance is done as Derived : Base. Only single inheritance is supported. Multiple interface inheritance discussed later.

```
Using System;

Public class parentclass
{

  Public parentclass()
 {

    Console.writeline("Parent Constructor.");

  }

  Public void print() {

    Console.writeline("I'm a Parent Class.");

  }

}

Public class childclass : parentclass
{

  Public childclass()
 {

    Console.writeline("Child Constructor.");

  }

  Public static void Main()
 {

    Childclass child = new childclass();

    Child.print();
```

```
  }

}
```

The output is:

Parent Constructor. Child Constructor. I'm a Parent Class

.

Derived class is exactly the same as base. Child class "is a" parent class except that derived is more special. Base are automatically initialized before derived. How can derived communicate with the base i.e. The specialized part communicate with the generic part. By using ": base()" at ctor time or "base.x()" later. "new" keyword used to override methods. Cast back to base type to call an overridden method of base.

Here is another example:

```
Using System;
Public class Parent {
  String parentstring;
  Public Parent() {
    Console.writeline("Parent Constructor.");
  }
  Public Parent(string mystring) {
    Parentstring = mystring;
    Console.writeline(parentstring);
  }

  Public void print() {
    Console.writeline("I'm a Parent Class.");
  }
}
  Public class Child : Parent {
    Public Child() : base("From Derived") {
      Console.writeline("Child Constructor.");
    }

    Public new void print() {
      Base.print(); Console.writeline("I'm a Child Class.");
    }

    Public static void Main() {
      Child child = new Child();
      Child.print(); ((Parent)child).print();
    }
  }
```

Output is:

From Derived
Child Constructor. I'm a Parent Class. I'm a Child Class. I'm a Parent Class.

Polymorphism means to invoke derived class methods through base class reference during run-time. Normally (as we saw) base class casting means base class method is called. Its handy when a group of objects is stored in array/list and we invoke some method on all of them. They dont have to be the same type but if related by inheritance, they can be cast.

```
Using System;
```

```
Public class drawingobject
{
 Public virtual void Draw()
 {
     Console.writeline("I'm just a generic drawing object.");
   }

}

Public class Line : drawingobject
{
 Public override void Draw()
 {
     Console.writeline("I'm a Line.");
 }
}

Public class Circle : drawingobject
{
 Public override void Draw()
 {
     Console.writeline("I'm a Circle.");
  }
}

Public class Square : drawingobject
{
 Public override void Draw()
 {
    Console.writeline("I'm a Square.");
 }
}

Public class drawdemo
{
  Public static int Main( )
    {

    Drawingobject[] dobj = new drawingobject[4];
    Dobj[0] = new Line();
    Dobj[1] = new Circle();
    Dobj[2] = new Square();
    Dobj[3] = new drawingobject();
    Foreach (drawingobject drawobj in dobj)
    {
      Drawobj.Draw();
    }
    Return 0;
  }
}
```

Polymorphic methods are declared with “virtual” keyword and the “override” keyword is used to provide an implementation. Polymorphism needs the signatures to be the same. Because of inheritance, the three derived classes can be treated as the base class. Because of polymorphism, the methods from drived can still be called.

Output is:

I'm a Line.
I'm a Circle. I'm a Square.
I'm just a generic drawing object.

Properties allow protected reads and writes to a field of a class. Other languages require custom getters and setters.

C# allows accessing properties like fields. Properties allows changing representation, public fields dont. Other languages use custom encapsulation to protect data.

```
Using System;

Public class Customer
{
    Private int m_id = -1;

    Public int getid()
    {
        Return m_id;
    }

    Public void setid(int id)
    {
        M_id = id;
    }

    Private string m_name = string.Empty;

    Public string getname()
    {
        Return m_name;
    }

    Public void setname(string name)
    {
        M_name = name;
    }
}

Public class customermanagerwithaccessormethods
{
    Public static void Main()
    {
        Customer cust = new Customer();

        Cust.setid(1);
        Cust.setname("Amelio Rosales");

        Console.writeline(
            "ID: {0}, Name: {1}",
            Cust.getid(),
            Cust.getname());

        Console.readkey();
    }
}
```

Its such a common pattern that C# has a dedicated language feature for it.

```
Using System;
Public class Customer
{
    Private int m_id = -1;

    Public int ID
    {
        Get
        {
            Return m_id;
        }
        Set
        {
            M_id = value;
        }
    }

    Private string m_name = string.Empty;

    Public string Name
    {
        Get
        {
            Return m_name;
        }
        Set
        {
            M_name = value;
        }
    }
}

Public class customermanagerwithproperties
{
    Public static void Main()
    {
        Customer cust = new Customer();

        Cust.ID = 1;
        Cust.Name = "Amelio Rosales";

 Console.writeline(
            "ID: {0}, Name: {1}",
            Cust.ID,
            Cust.Name);

        Console.readkey();
    }
}
```

Getters “return”, setters use “value”. Next we discuss read-only peroperties. Here is an example that uses read-only properties.

```
Using System;

Public class Customer
{
```

```
    Private int m_id = -1;
    Private string m_name = string.Empty;

    Public Customer(int id, string name)
    {
        M_id = id;
        M_name = name;
    }

    Public int ID
    {
        Get
        {
            Return m_id;
        }
    }

    Public string Name
    {
        Get
        {
            Return m_name;
        }
    }
}

Public class readonlycustomermanager
{
    Public static void Main()
    {
        Customer cust = new Customer(1, "Amelio Rosales");

        Console.writeline(
            "ID: {0}, Name: {1}",
            Cust.ID,
            Cust.Name);

        Console.readkey();
    }
}
```

This example has only get and no implementation of set. So what about write-only properties. Yes, they exist and have an implementation of set but not get.

```
  Using System;

Public class Customer
{
    Private int m_id = -1;

    Public int ID
    {
        Set
        {
            M_id = value;
        }
    }
```

```
    Private string m_name = string.Empty;

    Public string Name
    {
        Set
        {
            M_name = value;
        }
    }

    Public void displaycustomerdata()
    {
        Console.writeline("ID: {0}, Name:
            {1}", m_id, m_name);
    }
}

Public class writeonlycustomermanager
{
    Public static void Main()
    {
        Customer cust = new Customer();

        Cust.ID = 1;
        Cust.Name = "Amelio Rosales";

        Cust.displaycustomerdata();

        Console.readkey();
    }
}
```

Chapter 7

Lecture 7

Auto-implemented properties improve the common-case we saw. Here is the same example using auot-implemented properties. Properties have the same idea as getters and setters. They just allow simplified syntax. Same idea of backing store and a simplified syntax for the general case is enabled by auto-implemented properties.

```
  Using System;


Public class Customer
{
    Public int ID { get; set; }
    Public string Name { get; set; }
}

Public class autoimplementedcustomermanager
{
    Static void Main()
    {
        Customer cust = new Customer();

        Cust.ID = 1;
        Cust.Name = "Amelio Rosales";

        Console.writeline(
            "ID: {0}, Name: {1}",
            Cust.ID,
            Cust.Name);

        Console.readkey();
    }
}
```

An indexer enables your class to be treated like an array. However you have your internal data representation not an actual array. Its implemented using "this" keyword and square brackets syntax. It looks like a property implementation. Here is an example.

```
Class intindexer
{
    Private string[] mydata;

    Public intindexer(int size)
    {
        Mydata = new string[size];

        For (int i=0; i < size; i++)
        {
            Mydata[i] = "empty";
        }
```

```
    }

    Public string this[int pos]
    {
        Get
        {
            Return mydata[pos];
        }
        Set
        {
            Mydata[pos] = value;
        }
    }

    Static void Main(string[] args)
    {
        Int size = 10;

        Intindexer myind = new intindexer(size);

        Myind[9] = "Some Value";
        Myind[3] = "Another Value";
        Myind[5] = "Any Value";

        Console.writeline("\nindexer Output\n");

        For (int i=0; i < size; i++)
        {
            Console.writeline("myind[{0}]: {1}", i, myind[i]);
        }
    }
}
```

The output of this program looks like this:

Indexer Output

Myind[0]: empty
Myind[1]: empty
Myind[2]: empty
Myind[3]: Another Value myind[4]: empty
Myind[5]: Any Value
Myind[6]: empty
Myind[7]: empty
Myind[8]: empty
Myind[9]: Some Value

Indexers can take any number of parameters. The parameters can be integers, strings, or enums. Additionally they can be overloaded with different type and number of parameters. The following example has overloaded indexers.

```
Using System;

/// <summary>
///     Implements overloaded indexers.
/// </summary>
Class ovrindexer
```

```
{
    Private string[] mydata;
    Private int         arrsize;

    Public ovrindexer(int size)
    {
        Arrsize = size;
        Mydata = new string[size];

        For (int i=0; i < size; i++)
        {
            Mydata[i] = "empty";
        }
    }

    Public string this[int pos]
    {
        Get
       {
            Return mydata[pos];
        }
        Set
       {
            Mydata[pos] = value;
        }
    }

     Public string this[string data]
    {
        Get
       {
            Int count = 0;

            For (int i=0; i < arrsize; i++)
            {
                If (mydata[i] == data)
                    Count++;
            }
            Return count.tostring();
        }
        Set
       {
            For (int i=0; i < arrsize; i++)
            {
                If (mydata[i] == data)
                    Mydata[i] = value;
            }
        }
    }
    Static void Main(string[] args)
    {
        Int size = 10;
        Ovrindexer myind = new ovrindexer(size);

        Myind[9] = "Some Value";
        Myind[3] = "Another Value";
        Myind[5] = "Any Value";
```

```
        Myind["empty"] = "no value";

        Console.writeline("\nindexer Output\n");

        For (int i=0; i < size; i++)
        {
            Console.writeline("myind[{0}]: {1}", i, myind[i]);
        }

        Console.writeline("\nnumber of \"no value\" entries: {0}", myind["no value"]);
    }
}
```

Indexers with multiple parameters look like this.

```
Public object this[int param1, ..., int paramn] {
Get {
 // process and return some class data
 }
Set {
 // process and assign some class data
 }

   }
```

Our next topic is a "struct". Its a value-type, whereas class is a reference-type. Value types hold their value in memory where they are declared, but reference types hold a reference to an object in memory. If you copy a class, C# creates a new copy of the reference to the object and assigns the copy of the reference to the separate class instance. Structs cant have destructors, cant have implementation inheritance (still can have interface inheritance). Many built-in types are structs like System.Int32 is a C# int. C# built-in types are aliases for .NET Framework types. Syntax of struct and class are very similar. Here is an example of using a struct.

```
/// <summary>
/// Custom struct type, representing
    A rectangular shape
/// </summary>
Struct Rectangle
{
    /// <summary>
    /// Backing Store for Width
    /// </summary>
    Private int m_width;

    /// <summary>
    /// Width of rectangle
    /// </summary>
    Public int Width
    {
        Get
        {
            Return m_width;
        }
        Set
        {
            M_width = value;
        }
    }
```

```
    /// <summary>
    /// Backing store for Height
    /// </summary>
    Private int m_height;

    /// <summary>
    /// Height of rectangle
    /// </summary>
    Public int Height
    {
        Get
        {
            Return m_height;
        }
        Set
        {
            M_height = value;
        }
    }
}
```

Its instantiated and used just like a class.

```
Using System;

/// <summary>
/// Example of declaring and using
///    a struct
/// </summary>
Class structexample
{
    /// <summary>
    /// Entry point: execution starts
///        here
    /// </summary>
    Static void Main()
    {
        // instantiate a new Rectangle struct
        // where Width is set to 1 and Height is set to 3
       Rectangle rect1 = new Rectangle();
        Rect1.Width = 1;
        Rect1.Height = 3;

        // show the value of Width and Height for rect1
        Console.writeline("rect1: {0}:{1}", rect1.Width, rect1.Height);

        Console.readkey();
    }
}
```

The output of this program is: Output:

Rect1: 1:3

We can add constructors to a struct but cant overlaod the default ctor which initializes everything to default values. We can add methods like classes and they are called in exactly the same way.

An object initializer is a syntax for initializing a struct without using a ctor. It looks like this:

```
// you can also use object initialization syntax
        Rectangle rect11 = new Rectangle
        {
            Width = 1,
            Height = 3
        };
```

Next we discuss "Interfaces". They are declared like a class but have no implementation. There is only declarations of events, indexers, methods and/or properties. They are inherited by classes which provide the real implementation. So, what are interfaces good for if they don't implement functionality? They are great for putting together plug-n-play like architectures where components can be interchanged at will. The interface forces each component to expose specific public members that will be used in a certain way.

Interfaces define a contract. For instance, if class foo implements the idisposable interface, it is making a statement that it guarantees it has the Dispose() method, which is the only member of the idisposable interface. Any code that wishes to use class foo may check to see if class foo implements idisposable. When the answer is true, then the code knows that it can call foo.Dispose()

To define an interface, we use the following syntax.

```
Interface imyinterface
{
Void methodtoimplement();
}
```

The "I" prefix is a convention. There is no implementation, only semi-colon. There are only method signatures.
Let's now see how its used.

```
Class  interfaceimplementer : imyinterface
{
Static void Main()
{
 Interfaceimplementer iimp = new interfaceimplementer();
 Iimp.methodtoimplement();
}
Public void methodtoimplement()
{
 Console.writeline("methodtoimplement() called.");
}
}
```

The syntax used is same as for inheritance. Any signature difference or not implementing some method in the interface is an error. Interfaces can also be inherited. Here is an example.

```
Using System;

Interface iparentinterface
{
    Void parentinterfacemethod();
}

Interface imyinterface : iparentinterface
{
    Void methodtoimplement();
}
```

```
Class interfaceimplementer : imyinterface
{
    Static void Main()
    {
        Interfaceimplementer iimp = new interfaceimplementer();
        Iimp.methodtoimplement();
        Iimp.parentinterfacemethod();
    }

    Public void methodtoimplement()
    {
        Console.writeline("methodtoimplement() called.");
    }

    Public void parentinterfacemethod()
    {
        Console.writeline("parentinterfacemethod() called.");
    }
}
```

We can also do polymorphism using interfaces.

Chapter 8

Lecture 8

A delegate is a reference to a method. Its like function pointers in some other languages. Methods are algorithms that operate on data. Sometimes the data needs a special operation e.g. A comparator in a sorting routine. We can use a bad solution which would an if then else for all types we want our sorting routine to work for. There are two good solutions. One is to implement a comparator interface in all types we want and then pass an instance of a type that provides that interface. The other solution is using delegates i.e. References to functions. Now we can pass a comparator delegate to the sorting algorithm.

```
Using System;

// this is the delegate declaration
Public delegate int Comparer(object obj1, object obj2);

Public class Name
{
    Public string firstname = null;
    Public string lastname = null;

    Public Name(string first, string last)
    {
        Firstname = first;
        Lastname = last;
    }

    // this is the delegate method handler
    Public static int comparefirstnames(object name1, object name2)
    {
        String n1 = ((Name)name1).firstname;
        String n2 = ((Name)name2).firstname;

        If (String.Compare(n1, n2) > 0)
        {
            Return 1;
        }
        Else if (String.Compare(n1, n2) < 0)
        {
            Return -1;
        }
        Else
        {
            Return 0;
        }
    }

    Public override string tostring()
    {
        Return firstname + " " + lastname;
    }
}
```

```
Class simpledelegate
{
    Name[] names = new Name[5];

    Public simpledelegate()
    {
      Names[0] = new Name("Joe", "Mayo");
        Names[1] = new Name("John", "Hancock");
        Names[2] = new Name("Jane", "Doe");
        Names[3] = new Name("John", "Doe");
        Names[4] = new Name("Jack", "Smith");
    }
    Static void Main(string[] args)
    {
        Simpledelegate sd = new simpledelegate();

        // this is the delegate instantiation
        Comparer cmp = new Comparer(Name.comparefirstnames);
        Console.writeline("\nbefore Sort: \n");
        Sd.printnames();
        // observe the delegate argument
        Sd.Sort(cmp);
        Console.writeline("\nafter Sort: \n");
        Sd.printnames();
    }
    // observe  the delegate parameter
    Public void Sort(Comparer compare)
    {
        Object temp;

        For (int i=0; i < names.Length; i++)
        {
            For (int j=i; j < names.Length; j++)
            {
                // using delegate "compare" just like
                // a normal method
                If ( compare(names[i], names[j]) > 0 )
                {
                    Temp = names[i];
                    Names[i] = names[j];
                    Names[j] = (Name)temp;
                }
            }
        }
    }
    Public void printnames()
    {
        Console.writeline("Names: \n");

        Foreach (Name name in names)
        {
            Console.writeline(name.tostring());
        }
    }
}
```

A C# event is a class member that is activated whenever the event it was designed for occurs (fires).

Anyone interested in the event can register and be notified as soon as the event fires. At the time an event fires, methods registered with the event will be invoked.

Events and delegates work hand in hand. Any class, including the same class that the event is declared in, may register one of its methods with the event. This occurs through a delegate, which specifies the signature of the method that is registered for the event. The delegate may be one of the pre-defined .NET delegates or one that you declare yourself. Then, you assign the delegate to the event, which effectively registers the method that will be called when the event fires.

Here is an example:

```
Using System;
Using System.Drawing;
Using System.Windows.Forms;

// custom delegate
Public delegate void Startdelegate();

Class Eventdemo : Form
{
    // custom event
    Public event Startdelegate startevent;
    Public Eventdemo()
    {
        Button clickme = new Button();
        Clickme.Parent = this;
        Clickme.Text = "Click Me";
        Clickme.Location = new Point(
            (clientsize.Width - clickme.Width) /2,
            (clientsize.Height - clickme.Height)/2);
        // an eventhandler delegate is assigned
        // to the button's Click event
        Clickme.Click += new eventhandler(onclickmeclicked);
        // our custom "Startdelegate" delegate is assigned
        // to our custom "startevent" event.
        Startevent += new Startdelegate(onstartevent);
        // fire our custom event
        Startevent();
    }
    // this method is called when the "clickme" button is pressed
    Public void onclickmeclicked(object sender, eventargs ea)
    {
        Messagebox.Show("You Clicked My Button!");
    }
    // this method is called when the "startevent" Event is fired
    Public void onstartevent()
    {
        Messagebox.Show("I Just Started!");
    }
    Static void Main(string[] args)
    {
        Application.Run(new Eventdemo());
    }
}
```

To run this example in Visual Studio you will either have to add a reference to System.Drawing.dll and

Sys-tem.Windows.Forms.dll or you can make a new Windows Forms Project. This class inherits from Form, which essen-tially makes it a Windows Form. This automatically gives you all the functionality of a Windows Form, including Title.Bar, Minimize/Maximize/Close buttons, System Menu, and Borders. It is started by calling the Run() method of the static Application object with a reference to the form object as its parameter. The += syntax registers a delegate with the event. Its fired like a method call. The Click event is pre-defined, eventhandler delegate is also already in .NET. All you do is define your callback method (delegate handler method) that is invoked when someone presses the clickme button. The onclickmeclicked() method, conforms to the signature of the eventhandler delegate.

Next, we discuss exception handling. There can be unforeseen errors in the program. There are some normal errors and some exceptional errors e.g. File i/o error, system out of memory, null pointer exception. Exceptions are "thrown". They are derived from System.Exception class. They have a message property, contain a stacktrace, a tostring method. Identifying the exceptions you'll need to handle depends on the routine you're writing e.g. System.IO.File.openread() could throw securityexception, argumentexception, argumentnullexception, pathtoolongexception, directorynotfoundexception, unauthorizedaccessexception, filenotfoundexception, or notsupportedexception.

Here is an example of exception handling:

```
Using System;
Using System.IO;

Class trycatchdemo
{
    Static void Main(string[] args)
    {
        Try
        {
            File.openread("nonexistentfile");
        }
        Catch(Exception ex)
        {
            Console.writeline(ex.tostring());
        }
    }
}
```

A single exception can be handled differently as well:

```
1           catch (filenotfoundexception fnfex)
2           {
3               Console.writeline(fnfex.tostring());
4           }
5           catch (Exception ex)

6           {
7               Console.writeline(ex.tostring());
8           }
```

First handler that handles an exception catches it. Otherwise the exception bubbles up the caller stack until someone handles it.

Chapter 9

Lecture 9

Exception can leave your program in an inconsistent state by not releasing resources or doing some other type of cleanup. Sometimes you need to perform clean up actions whether or not your program succeeds. These are good candidates for using a finally block e.g. A filestream must be closed.

```
Using System;
Using System.IO;
Class finallydemo
{
    Static void Main(string[] args)
    {
        Filestream outstream = null;
        Filestream instream = null;
        Try
        {
            Outstream = File.openwrite("destinationfile.txt");
            Instream = File.openread("bogusinputfile.txt");
        }
        Catch (Exception ex)
        {
            Console.writeline(ex.tostring());
        }
        Finally
        {
            If (outstream != null)
            {
                Outstream.Close();
                Console.writeline("outstream closed.");
            }
            If (instream != null)
            {
                Instream.Close();
                Console.writeline("instream closed.");
            }
        }
    }
}
```

The alternate would be to duplicate code after catch and in catch. But finally is a neat cleanup solution.

Next, we discuss attributes. Attributes add declarative information to your programs. They are used for various purposes during runtime. They can be used at design time by application development tools. For example, dllimpor-tattribute allows a program to communicate with Win32 libraries. Obsoleteattribute causes a compile-time warning to appear. Such things would be difficult to accomplish with normal code. Attributes add metadata to your programs.

When your C# program is compiled, it creates a file called an assembly, which is normally an executable or DLL library. Assemblies are self-describing because they have metadata. Via reflection, a program's

attributes can be retrieved from its assembly metadata. Attributes are classes that can be written in C# and are used to decorate your code with declarative information. Its a powerful concept. It Let's you extend your language by creating customized declarative syntax with attributes.

Attributes are generally applied physically in front of type and type member declarations. They are declared with square brackets, "[" and "]" surrounding the attribute such as " [obsoleteattribute]". The "Attribute" part of the attribute name is optional i.e. "[Obsolete]" is correct as well. Parameter lists are also possible.

```
Using System;
Class basicattributedemo
{
    [Obsolete]
    Public void myfirstdeprecatedmethod()
    {
        Console.writeline("Called myfirstdeprecatedmethod().");
    }
    [obsoleteattribute]
    Public void myseconddeprecatedmethod()
    {
        Console.writeline("Called myseconddeprecatedmethod().");
    }
    [Obsolete("You shouldn't use this method anymore.")]
    Public void mythirddeprecatedmethod()
    {
        Console.writeline("Called mythirddeprecatedmethod().");
    }
    // make the program thread safe for COM
    [stathread]
    Static void Main(string[] args)
    {
        Basicattributedemo attrdemo = new basicattributedemo();
        Attrdemo.myfirstdeprecatedmethod();
        Attrdemo.myseconddeprecatedmethod();
        Attrdemo.mythirddeprecatedmethod();
    }
}
```

When you compile this program you'll see.

```
>csc basicattributedemo.cs
Microsoft (R) Visual C\# .NET Compiler version 7.10.2292.4 for Microsoft (R) .NET Framework version 1.1.4322
Copyright (C) Microsoft Corporation 2001-2002. All rights reserved.

Basicattributedemo.cs(29,3): warning CS0612:
'Basicattributedemo.myfirstdeprecatedmethod()'is obsolete basicattributedemo.cs(30,3): warning CS0612:
'Basicattributedemo.myseconddeprecatedmethod()' is obsolete basicattributedemo.cs(31,3):
warning CS0618:
'Basicattributedemo.mythirddeprecatedmethod()' is obsolete: 'You shouldn't use this method anymore.'
```

Stathread is a common attribute you will see later. It stands for Single Threaded Apartment model which is used for communicating with unmanaged COM.

Attribute parameters can be either positional parameters or named parameters. Usually named parameters with optional stuff, but positional can be optional as well e.g. "[Obsolete("You shouldn't use this method anymore.", true)] public void mythirddeprecatedmethod()" will give an error instead of warning. The attribute dllimportattribute has

Both positional and named parameters "[dllimport("User32.dll", entrypoint="messagebox")]". Positional parameters come before named parameters. There is no order requirement on named parameters.

```
Using System;
[assembly: clscompliant(true)]
Public class attributetargetdemo
{
    Public void nonclscompliantmethod(uint nclsparam)
    {
        Console.writeline("Called nonclscompliantmethod().");
    }
    [stathread]
    Static void Main(string[] args)
    {
        Uint myuint = 0;
        Attributetargetdemo tgtdemo = new attributetargetdemo();
        Tgtdemo.nonclscompliantmethod(myuint);
    }
}
```

We rarely write new attributes but we use them extensively.

Enums (or enumerations) are strongly typed constants. They are unique types that allow you to assign symbolic names to integral values. Enum of one type may not be implicitly assigned to an enum of another type (even though the underlying value of their members is the same). All assignments between different enum types and integral types require an explicit cast. It allows you to work with integral values, but using a meaningful name. North, South, East, and West or the zet of integers 0, 1, 2, and 3. C# type, enum, inherits the Base Class Library (BCL) type, Enum.

```
Using System;
// declares the enum
Public enum Volume
{
    Low,_
    Medium,
    High
}
// demonstrates how to use the enum
Class enumswitch
{
    Static void Main()
    {
        // create and initialize
        // instance of enum type
        Volume myvolume = Volume.Medium;
        // make decision based
        // on enum value
        Switch (myvolume)
        {
            Case Volume.Low:
                Console.writeline("The volume has been turned Down.");
                Break;
```

```
            Case Volume.Medium:
                Console.writeline("The volume is in the middle.");
                Break;
            Case Volume.High:
                Console.writeline("The volume has been turned up.");
                Break;
        }
        Console.readline();
    }
}
```

Default underlying type of an enum is "int". It can be changed by specifying a "base". Valid base types include byte, sbyte, short, ushort, int, uint, long, and ulong. Default value of first member is 0. You can assign any member any value. If it is unassigned it gets +1 the value of its predecessor.

```
Public enum Volume  : byte
{
 Low= 1,
 Medium,
 High
}
```

To convert user input to enum:

```
// get value user provided
String volstring  = Console.readline();
Int volint  = Int32.Parse(volstring);
// perform  explicit cast from
// int to Volume enum type

Volume myvolume  = (Volume)volint
```

Enum inherits from System.Enum in base class library.

```
// get a list of member names from Volume enum,
// figure out the numeric value, and display
Foreach  (string volume in Enum.getnames(typeof(Volume)))
{
Console.writeline("Volume Member:  {0}\n Value:  {1}",
Volume,  (byte)Enum.Parse(typeof(Volume), volume));
}
```

To enumerate values.

```
// get all values  (numeric values) from the Volume
// enum type, figure out member name, and display
Foreach  (byte val in Enum.getvalues(typeof(Volume)))
{
Console.writeline("Volume Value:  {0}\n Member:  {1}",
Val, Enum.getname(typeof(Volume), val));
}
```

Next topic is operator overloading. You can add operators to your own types e.g. A matrix type can have an add and a dot product operator etc.

```
Matrix result  = mat1.Add(mat2); // instance
Matrix result  = Matrix.Add(mat1, mat2);
Matrix result  = mat1.dotproduct(mat2).dotproduct(mat3);
Matrix result  = mat1  + mat2;
Matrix result  = mat1  * mat2; 6  Matrix result = mat1 * mat2 * mat3 * mat4;
```

It should not be used when its not natural to be used. Its implementation syntax is:

```
Public static Matrix operator *(Matrix mat1, Matrix mat2)
```

```
{
 // dot product  implementation
}
```

Here is an example.

```
Using System;

Class Matrix
{
    Public const int dimsize = 3;
    Private double[,] m_matrix = new double[dimsize, dimsize];

    // allow callers to initialize
    Public double this[int x, int y]
    {
        Get { return m_matrix[x, y]; }
        Set { m_matrix[x, y] = value; }
    }

    // let user add matrices
    Public static Matrix operator +(Matrix mat1, Matrix mat2)
    {
        Matrix newmatrix = new Matrix();

        For (int x = 0; x < dimsize; x++)
            For (int y = 0; y < dimsize; y++)
                Newmatrix[x, y] = mat1[x, y] + mat2[x, y];

        Return newmatrix;
    }
}
```

Overloaded operators must be static. They must be declared in the class for which the operator is defined. It is required that matching operators are both defined, e.g. == and != so that the behavior is consistent. Access modifiers on types and assemblies provide encapsulation. They are: private, protected, internal, protected internal, and public.

Next, we see how generic collections are used: There is a very useful List collection. It is better than arraylist of objects in earlier .NET versions. Another useful collection is Dictionary¡tkey,tvalue¿ vs. A non-generic hashtable of objects. We write "using System.Collections.Generic" to include these collections.

```
List<int> myints  = new List<int>();

Myints.Add(1);
Myints.Add(2);
Myints.Add(3);

For(int i= 0; i < myints.Count; i++)
{
Console.writeline("myints:{0}", myints[i]);
}

Public class Customer
{
Public Customer(int id, string name)
{
ID= id;
Name = name;
```

```
}
Public int ID
{
Get;
Set;
}
Public string Name
{
Get;
Set;
}
}
Dictionary<int, Customer> customers = new Dictionary<int, Customer>();

Customer cust1 = new Customer(1, "Cust 1");
Customer cust2 = new Customer(2, "Cust 2");
Customer cust3 = new Customer(3, "Cust 3");

Customers.Add(cust1.ID, cust1);
Customers.Add(cust2.ID, cust2);
Customers.Add(cust3.ID, cust3);

Foreach (keyvaluepair<int, Customer> custkeyval in customers)
{
Console.writeline(
"Customer ID:{0}, Name: {1}",
Custkeyval.Key,
Custkeyval.Value.Name);
}
```

Chapter 10

Lecture 10

Let's discuss Anonymous methods. It is used with delegates and handlers and events. Anonymous methods result in much less code. Anonymous method is a method without a name. You don't declare anonymous methods, Instead they get hooked up directly to events. With delegates 1) you declare the delegate, 2) write a method with a signature defined by the delegate interface, 3) declare the event based on that delegate, and 4) write code to hook the handler method up to the delegate. To delcare an anonymous method, you just use keyword "delegate" followed by method body.

```
Public partial class Form1: Form
{
 Public Form1()
 {
  Button btnhello= new Button();
  Btnhello.Text= "Hello";
  Btnhello.Click +=
  Delegate
  {
   Messagebox.Show("Hello");
  };
  Controls.Add(btnhello);
 }
}
```

We can skip parameters when not using them. Here is an example with parameters.

```
Using System;
Using System.Windows.Forms;

Public partial class Form1 : Form
{
    Public Form1()
    {
        Button btnhello = new Button();
        Btnhello.Text = "Hello";

        Btnhello.Click +=
            Delegate
            {
                Messagebox.Show("Hello");
            };

        Button btngoodbye = new Button();
        Btngoodbye.Text = "Goodbye";
        Btngoodbye.Left = btnhello.Width + 5;
        Btngoodbye.Click +=
            Delegate(object sender, eventargs e)
            {
                String message = (sender as Button).Text;
                Messagebox.Show(message);
            };
```

```
        Controls.Add(btnhello);
        Controls.Add(btngoodbye);
    }
}
```

We can create nullable value types by appending a question mark to a type name.

```
Int? Unitsinstock = 5;
Datetime? Startdate;
Startdate= datetime.Now;
Startdate= null;
Int availableunits;

If(unitsinstock== null)
{
Availableunits = 0;
}
Else
{
Availableunits = (int)unitsinstock;
}
// OR just
Int availableunits = unitsinstock ?? 0;
```

We will now discuss debugging in Visual Studio.By printing output frequently we can debug. We can use Console.Write to print. Breakpoints allow stopping the program during execution. Press F5 and execution will stop at breakpoint. You can hover over variabeles for their current values. Locals, watch, call stack, immediate window give you different information about the program. Your changes will have effect on the current execution. Conditional breakpoints have a hit count and when hit condition.

We now discuss XML. XML or extensible Markup Language is widely used for exchanging data. Its readable for both humans and machines. Its a stricter version of HTML. Its made of tags, attributes, and values.

```
<users>
<user name="John Doe" age="42" />
<user name="Jane Doe" age="39" />
</users >
```

And here is a larger XML file.

```
<gesmes:Envelope xmlns:gesmes="http://www.gesmes.org/xml/2002-08-01"
xmlns="http://www.ecb.int/vocabulary/2002-08-01/eurofxref">
 <gesmes:subject>Reference rates</gesmes:subject>
 <gesmes:Sender>
  <gesmes:name>European Central Bank</gesmes:name>
 </gesmes:Sender>
 <Cube>
  <Cube time="2012-12-18">
   <Cube currency="USD" rate="1.3178"/>
   <Cube currency="JPY" rate="110.53"/>
   <Cube currency="BGN" rate="1.9558"/>
   <Cube currency="CZK" rate="25.200"/>
   <Cube currency="DKK" rate="7.4603"/>
   <Cube currency="GBP" rate="0.81280"/>
   <Cube currency="HUF" rate="288.40"/>
   <Cube currency="LTL" rate="3.4528"/>
   <Cube currency="LVL" rate="0.6961"/>
```

```
   <Cube currency="PLN" rate="4.0928"/>
   <Cube currency="RON" rate="4.4700"/>
   <Cube currency="SEK" rate="8.7378"/>
   <Cube currency="CHF" rate="1.2080"/>
   <Cube currency="NOK" rate="7.3850"/>
   <Cube currency="HRK" rate="7.5380"/>
   <Cube currency="RUB" rate="40.6850"/>
   <Cube currency="TRY" rate="2.3476"/>
   <Cube currency="AUD" rate="1.2512"/>
   <Cube currency="BRL" rate="2.7595"/>
   <Cube currency="CAD" rate="1.2972"/>
   <Cube currency="CNY" rate="8.2079"/>
   <Cube currency="HKD" rate="10.2131"/>
   <Cube currency="IDR" rate="12707.71"/>
   <Cube currency="ILS" rate="4.9615"/>
   <Cube currency="INR" rate="72.2880"/>
   <Cube currency="KRW" rate="1413.30"/>
   <Cube currency="MXN" rate="16.7795"/>
   <Cube currency="MYR" rate="4.0236"/>
   <Cube currency="NZD" rate="1.5660"/>
   <Cube currency="PHP" rate="54.068"/>
   <Cube currency="SGD" rate="1.6053"/>
   <Cube currency="THB" rate="40.285"/>
   <Cube currency="ZAR" rate="11.2733"/>
  </Cube>
 </Cube>
</gesmes:Envelope>
```

We will now learn to read XML with the xmlreader class. There are two methods to read XML document. Using xmldocument and xmlreader. Xmldocuments reads entire document in memory, Let's you go forward, backward, even apply xpath searches on it. Xmlreader is fast, uses less memory and provides one element at a time.

```
Using System;
Using System.Text;
Using System.Xml;

Namespace parsingxml
{
    Class Program
    {
        Static void Main(string[] args)
        {
            Xmlreader xmlreader =
xmlreader.Create("http://www.ecb.int/stats/eurofxref/eurofxref-daily.xml");
            While (xmlreader.Read())
            {
                If ((xmlreader.nodetype == xmlnodetype.Element) && (xmlreader.Name ==
"Cube"))
                {
                    If (xmlreader.hasattributes)
                        Console.writeline(xmlreader.getattribute("currency") + ": " +
xmlreader.getattribute("rate"));
                }
            }
            Console.readkey();
        }
```

```
    }
}
```

Using xmldocument is easier. No order is required. Documentelement is the root element and childnodes is the set of children of any node.

```
Namespace  parsingxml
{
Class Program
{
Static void Main(string[] args)
{
Xmldocument  xmldoc = new xmldocument();
Xmldoc.Load("http://www.ecb.int/stats/eurofxref/eurofxref-daily.xml");
Foreach (xmlnode xmlnode in xmldoc.documentelement.childnodes[2].childnodes[0].Child
Console.writeline(xmlnode.Attributes["currency"].Value + ": "+ xmlnode.Attribut
Console.readkey();
}
}
}
```

An xmlnode is derived from xmlelement and contains Name, innertext, innerxml, outerxml, and Attributes. Xpath is a cross-platform Xml Query language. We will look at basic examples only. We will see xmldocument methods that take xpath queries and return xmlnode(s) in particular selectsinglenode and selectnodes.

We will use the RSS feed from CNN, located at http://rss.cnn.com/rss/edition world.rss. It has a hierarchy of rss containing channel containing item.

```
Using System;
Using System.Text;
Using System.Xml;

Namespace parsingxml
{
    Class Program
    {
        Static void Main(string[] args)
        {
            Xmldocument xmldoc = new xmldocument();
            Xmldoc.Load("http://rss.cnn.com/rss/edition_world.rss");
            Xmlnode titlenode = xmldoc.selectsinglenode("//rss/channel/title");
            If (titlenode != null)
                Console.writeline(titlenode.innertext);
            Console.readkey();
        }
    }
}
```

Another example with selectnodes instead of selectsinglenode.

```
Using System;
Using System.Text;
Using System.Xml;

Namespace parsingxml
{
    Class Program
    {
```

```
        Static void Main(string[] args)
        {
            Xmldocument xmldoc = new xmldocument();
            Xmldoc.Load("http://rss.cnn.com/rss/edition_world.rss");
            Xmlnodelist itemnodes = xmldoc.selectnodes("//rss/channel/item");
            Foreach (xmlnode itemnode in itemnodes)
            {
                Xmlnode titlenode = itemnode.selectsinglenode("title");
                Xmlnode datenode = itemnode.selectsinglenode("pubdate");
                If ((titlenode != null) && (datenode != null))
                    Console.writeline(datenode.innertext + ": " + titlenode.innertext);
            }
            Console.readkey();
        }
    }

}
```

If we only need title:

```
Using System;
Using System.Text;
Using System.Xml; 4
Namespace  parsingxml
{
Class Program
{
Static void Main(string[] args)
{
Xmldocument  xmldoc = new xmldocument();
Xmldoc.Load("http://rss.cnn.com/rss/edition_world.rss");
Xmlnodelist  itemnodes= xmldoc.selectnodes("//rss/channel/item");
Foreach (xmlnode  itemnode in itemnodes)
{
Xmlnode  titlenode= itemnode.selectsinglenode("title");
Xmlnode  datenode = itemnode.selectsinglenode("pubdate");
If((titlenode != null) && (datenode != null))
Console.writeline(datenode.innertext + ": "+ titlenode.innertext);
}
Console.readkey();
}
}
}
```

Let's write some XML using xmlwriter.

```
Using System;
Using System.Text;
Using System.Xml;

Namespace writingxml
{
    Class Program
    {
        Static void Main(string[] args)
        {
            Xmlwriter xmlwriter = xmlwriter.Create("test.xml");

            Xmlwriter.writestartdocument();
```

```
            Xmlwriter.writestartelement("users");

            Xmlwriter.writestartelement("user");
            Xmlwriter.writeattributestring("age", "42");
            Xmlwriter.writestring("John Doe");
            Xmlwriter.writeendelement();

            Xmlwriter.writestartelement("user");
            Xmlwriter.writeattributestring("age", "39");
            Xmlwriter.writestring("Jane Doe");

            Xmlwriter.writeenddocument();
            Xmlwriter.Close();
        }
    }
}
```

The above code will create the following XML:

```
<users>
  <user age="42">John Doe</user>
  <user age="39">Jane Doe</user>
</users>
```

Let's now see writing with xmldocument (especially for updates).

```
Using System;
Using System.Text;
Using System.Xml;
Using System.Xml.Serialization;

Namespace writingxml
{
    Class Program
    {
        Static void Main(string[] args)
        {
            Xmldocument xmldoc = new xmldocument();
            Xmlnode rootnode = xmldoc.createelement("users");
            Xmldoc.appendchild(rootnode);

            Xmlnode usernode = xmldoc.createelement("user");
            Xmlattribute attribute = xmldoc.createattribute("age");
            Attribute.Value = "42";
            Usernode.Attributes.Append(attribute);
            Usernode.innertext = "John Doe";
            Rootnode.appendchild(usernode);

            Usernode = xmldoc.createelement("user");
            Attribute = xmldoc.createattribute("age");
            Attribute.Value = "39";
            Usernode.Attributes.Append(attribute);
            Usernode.innertext = "Jane Doe";
            Rootnode.appendchild(usernode);

            Xmldoc.Save("test-doc.xml");
        }
    }
}
```

Updating is hard with a combination of xmlreader and xmlwriter.

```
Using System;
Using System.Text;
Using System.Xml;
Using System.Xml.Serialization;

Namespace writingxml
{
    Class Program
    {
        Static void Main(string[] args)
        {
            Xmldocument xmldoc = new xmldocument();
            Xmldoc.Load("test-doc.xml");
            Xmlnodelist usernodes = xmldoc.selectnodes("//users/user");
            Foreach (xmlnode usernode in usernodes)
            {
                Int age = int.Parse(usernode.Attributes["age"].Value);
                Usernode.Attributes["age"].Value = (age + 1).tostring();
            }
            Xmldoc.Save("test-doc.xml");
        }
    }
}
```

Chapter 11

Lecture 11

Let's start discussing extension methods.

```
Public class myutils
{
Public static bool isnumeric(string s)
{
Float output;
Return float.tryparse(s, out output);
}
}
String test = "4";
If(myutils.isnumeric(test))
Console.writeline("Yes");
Else

Console.writeline("No");
```

We can simplify the above example with an extension method.

```
Public class myutils
{
Public static bool isnumeric(string s)
{
Float output;
Return float.tryparse(s, out output);
}
}

String test = "4";
If(myutils.isnumeric(test))
Console.writeline("Yes");
Else

Console.writeline("No");
```

Let's now see how we can handle files.

```
Using System;
Using System.IO;

Namespace filehandlingarticleapp
{
    Class Program
    {
        Static void Main(string[] args)
        {
            If(File.Exists("test.txt"))
            {
                String content = File.readalltext("test.txt");
                Console.writeline("Current content of file:");
                Console.writeline(content);
            }
            Console.writeline("Please enter new content for the file:");
```

```
            String newcontent = Console.readline();
            File.writealltext("test.txt", newcontent);
        }
    }
}
```

And another example.

```
Using System;
Using System.IO;

Namespace filehandlingarticleapp
{
    Class Program
    {
        Static void Main(string[] args)
        {
            If(File.Exists("test.txt"))
            {
                String content = File.readalltext("test.txt");
                Console.writeline("Current content of file:");
                Console.writeline(content);
            }
            Console.writeline("Please enter new content for the file - type exit and
press enter to finish editing:");
            String newcontent = Console.readline();
            While (newcontent != "exit")
            {
                File.appendalltext("test.txt", newcontent + Environment.newline);
                Newcontent = Console.readline();
            }
        }
    }
}
```

If using keyword is not used, manually call the Close method of the stream.

```
Console.writeline("Please enter new content for the file - type exit and press enter to
finish”);
Using (streamwriter sw= new streamwriter("test.txt"))
{
String  newcontent = Console.readline();
While (newcontent != "exit")
{
Sw.Write(newcontent+ Environment.newline);
Newcontent = Console.readline();
}
}
```

We can delete a file like this.

```
Console.writeline("Please enter new content for the file  - type exit and press enter to
finish”);
Using (streamwriter sw= new streamwriter("test.txt"))
{
String  newcontent = Console.readline();
While (newcontent != "exit")
{
Sw.Write(newcontent+ Environment.newline);
Newcontent = Console.readline();
}
```

```
}
```

We can delete a directory like this.

```
Console.writeline("Please enter new content for the file -type exit and press enter to
finish”);
Using (streamwriter sw= new streamwriter("test.txt"))
{
String  newcontent = Console.readline();
While (newcontent != "exit")
{
Sw.Write(newcontent+ Environment.newline);
Newcontent = Console.readline();
}
}
```

We can rename a file like this.

```
If (File.Exists("test.txt"))
{
Console.writeline("Please enter a new name for this file:");
String  newfilename= Console.readline();
If (newfilename != String.Empty)
{
File.Move("test.txt", newfilename);
If (File.Exists(newfilename))
{
Console.writeline("The file was renamed to " + newfilename);
Console.readkey();
}
}
}
```

We can rename a directory like this.

```
If (Directory.Exists("testdir"))
{
Console.writeline("Please enter a new name for this directory:");
String  newdirname = Console.readline();
If (newdirname != String.Empty)
{
Directory.Move("testdir", newdirname);
If (Directory.Exists(newdirname))
{
Console.writeline("The directory was renamed to "+ newdirname);
Console.readkey();
}
}
}
```

We can create a new directory like this.

```
Console.writeline("Please enter a name for the new directory:");
String  newdirname = Console.readline();
If (newdirname != String.Empty)
{
Directory.createdirectory(newdirname);
If (Directory.Exists(newdirname))
{
Console.writeline("The directory was created!");
```

```
Console.readkey();
}
}
```

A fileinfo object gives you information about a file.

```
Static void Main(string[] args)
{
Fileinfo fi = new fileinfo(System.Reflection.Assembly.getexecutingassembly().Location);
If(fi!= null)
Console.writeline(String.Format("Information about file: {0}, {1} bytes, last modified o
Console.readkey();
}
```

Directoryinfo gives you information about a directory.

```
Directoryinfo di= new
directoryinfo(Path.getdirectoryname(System.Reflection.Assembly.getexecuti
If(di!= null)
{
Fileinfo[] subfiles= di.getfiles();
If(subfiles.Length> 0)
{
Console.writeline("Files:");
Foreach (fileinfo  subfile in subfiles)
{
Console.writeline("" + subfile.Name + " (" + subfile.Length + " bytes)");
}
}
Console.readkey();
}
```

Finding directories as well is as easy.

```
Directoryinfo[] subdirs = di.getdirectories();
If(subdirs.Length > 0)
{
Console.writeline("Directories:");
Foreach (directoryinfo  subdir in subdirs)
{
Console.writeline("" + subdir.Name);
}
}
```

We can use reflection to get information about types in our program.

```
Using System;
Using System.Collections.Generic;
Using System.Text;
Using System.Reflection;

Namespace  reflectiontest
{
Class Program
{
Static void Main(string[] args)
{
String test = "test";
Console.writeline(test.gettype().fullname);
Console.writeline(typeof(Int32).fullname);
Console.readkey();
}
}
```

```
}
```

Here is another example.

```
Using System;
Using System.Collections.Generic;
Using System.Text;
Using System.Reflection;

Namespace reflectiontest
{
    Class Program
    {
        Static void Main(string[] args)
        {
            Assembly assembly = Assembly.getexecutingassembly();
            Type[] assemblytypes = assembly.gettypes();
            Foreach(Type t in assemblytypes)
                Console.writeline(t.Name);
            Console.readkey();
        }
    }

    Class dummyclass
    {
        //Just here to make the output a tad less boring :)
    }
}
```

Let's see how methods are shown in reflection.

```
Using System;
Using System.Collections.Generic;
Using System.Text;
Using System.Reflection;

Namespace reflectiontest
{
    Class Program
    {
        Static void Main(string[] args)
        {
            Type testtype = typeof(testclass);
            Constructorinfo ctor = testtype.getconstructor(System.Type.emptytypes);
            If (ctor != null)
            {
                Object instance = ctor.Invoke(null);
                Methodinfo methodinfo = testtype.getmethod("testmethod");
                Console.writeline(methodinfo.Invoke(instance, new object[] { 10 }));
            }
            Console.readkey();
        }
    }

    Public class testclass
    {
        Private int testvalue = 42;

        Public int testmethod(int numbertoadd)
        {
```

```
            Return this.testvalue + numbertoadd;
        }
    }
}
```

We now have a more detailed example. We will now build a complete profile save load system with reflection.

```
Using System;
Using System.IO;
Using System.Reflection;


Class Program
{
    Public class Person
    {
        Public void Load()
        {
            If (File.Exists("settings.dat"))
            {
                Type type = this.gettype();

                String propertyname, value;
                String[] temp;
                Char[] splitchars = new char[] { '|' };
                Propertyinfo propertyinfo;

                String[] settings = File.readalllines("settings.dat");
                Foreach (string s in settings)
                {
                    Temp = s.Split(splitchars);
                    If (temp.Length == 2)
                    {
                        Propertyname = temp[0];
                        Value = temp[1];
                        Propertyinfo = type.getproperty(propertyname);
                        If (propertyinfo != null)
                            This.setproperty(propertyinfo, value);
                    }
                }
            }
        }

        Public void Save()
        {
            Type type = this.gettype();
            Propertyinfo[] properties = type.getproperties();
            Textwriter tw = new streamwriter("settings.dat");
            Foreach (propertyinfo propertyinfo in properties)
                Tw.writeline(propertyinfo.Name + "|" + propertyinfo.getvalue(this,
null));
            Tw.Close();
        }

        Public void setproperty(propertyinfo propertyinfo, object value)
        {
```

```
            Switch (propertyinfo.propertytype.Name)
            {
                Case "Int32":
                    Propertyinfo.setvalue(this, Convert.toint32(value), null);
                    Break;
                Case "String":
                    Propertyinfo.setvalue(this, value.tostring(), null);
                    Break;
            }
        }

        Public int Age { get; set; }
        Public string Name { get; set; }
    }

    Static void Main(string[] args)
    {
        Person person = new Person();
        Person.Load();
        If ((person.Age > 0) && (person.Name != String.Empty))
        {
            Console.writeline("Hi " + person.Name + " - you are " + person.Age + " years
old!");
        }
        Else
        {
            Console.writeline("I don't seem to know much about you. Please enter the
following information:");
            Type type = typeof(Person);
            Propertyinfo[] properties = type.getproperties();
            Foreach (propertyinfo propertyinfo in properties)
            {
                Console.writeline(propertyinfo.Name + ":");
                Person.setproperty(propertyinfo, Console.readline());
            }
            Person.Save();
            Console.writeline("Thank you! I have saved your information for next time.");
        }
        Console.readkey();
    }
}
```

Chapter 12

Lecture 12

We will start discussing WPF (Windows Presentation Foundations) now. It was publicly announced in 2003 (codenamed Avalon). WPF 4 was released in April 2010. It has a steep learning curve. Code has to be written in many places. There are multiple ways to do a particular task.

WPF enables polished user interfaces which are getting a lot of attention. It enables rapid iterations and major interface changes throughout the development process. It allows to keep user interface description and implementation separate. Developers can create an "ugly" application which designers can re-theme. Win32 style of programming makes such re-theming difficult. The code to re-paint the user interface is mixed up with program logic. GDI was an earlier user interface library introduced in windows 1.0 in 1985. Opengl was a leap ahead introduced in the 90's with directx coming in 95 and directx 2 in 96. GDI+ is a newer user interface library based on directx. It is also used behind Xbox graphics. Next was Windows Forms which is the primary way of user interface design in C#. XNA comes with managed library for directx and is great for game development (.net/com interoperability not required). A simple example is drawing bitmaps on buttons which can be efficiently done using GDI.

The highlights of WPF are 1) broad integration (2D, 3D, video, speech libraries etc.) 2) Resolution Independence with WPF giving emphasis on vector graphics 3) Hardware accelration as it is based on Direct3D but it can work using software pipeline also if Direct3D hardware is not available. 4) Declrative progrmaming using extensible Application Markup Language (XAML; pronounced "Zammel"). Custom attribute and configuration files were always there but XAML is very rich. 5) Rich composition and customization e.g. You can create a combobox filled with animated

Buttons or a Menu filled with live video clips! And it is quite easy to skin applications.

In short, WPF aims to combine the best attributes of systems such as directx (3D and hardware acceleration), Windows Forms (developer productivity), Adobe Flash (powerful animation support) and HTML (declarative markup). The first release in November 2006 was WPF 3.0 because it shipped as part of the .NET Framework 3.0. WPF 3.5 came an year later. Next version as part of .NET 3.5 SP1 came in August 2008. WPF Toolkit released in Aug 2008 was experimental. The toolkit has quick releases. Regarding tool support, WPF extensions for Visual Studio 2005 came a few months after the first WPF release and a public release of expression blend. Now, Visual Studio 2012 is a first class WPF devevelopment environment. Its mostly re-written using WPF and expression blend is 100% WPF and is great for designing and prototyping WPF apps.

New things that came in WPF 3.5/3.5SP1 include Interactive3D with 2d elements in 3d scenes, first class interoperability with directx, Better data binding using XLINQ and better validation and debugging which reduces code, Better special effects, High performance custom drawing, Text improvements, enhancements to Partial-trust apps, improved deployment, and improved performance.

Things that came with WPF 4.0 include multi-touch support — compatible with surface API v2, win7 support like jump lists, new common dialogs etc., new controls like datagrid, calendar etc, easing animation functions (bounce, elastic), enhanced styling with visual state manager, improved layout on pixel boundaries, non-blurry text but some limitations so must opt-in, deployment improvements, and performance improvements.

Silverlight in comparison is a light-weight version for web. It chose to follow WPF appraoch. First released in 2007 and in April 2010 4th version was released near WPF 4. There is often confusion when to use one or

the other. Both can run in and outside the web. Silverlight is ostly a subset of WPF but there are some incompatibilities. Should I use full .net or partial .net and should I have the ability to run on other devices e.g. Macs. Ideally common codebase should work for both but today best case is using #ifdefs to handle incompatibilities.

XAML is primarily used to describe interfaces in WPF and Silverlight. It is also used to express activities and configurations in workflow foundation (WF) and windows communication foundation (WCF). Its a common language for programmers and other experts e.g. UI design experts. Field specific development tools can be made. Field experts are graphic designers. They can use a design tool such as expression blend. Other than co-ordinating with designers, XAML is good for a concise way to represent UI or hierarchies of objects, encourages separation of front-end and back-end, tool support with copy and paste, used by all WPF tools.

XAML is xml with a set of rules. Its a declrative programming language for creating and initializing objects. Itsa way to use .net apis.

Comparisons with SVG, HTML etc are misguided. It has few keywords, not much elemetns. It doesn't make sense without .net, like C# doesnt make sense without .net. Microsoft formalized XAML vocabluries, e.g. WPF XAML vocabluary.

Online specifications for XAML object specification language, and WPF and Silverlight vocabluaries are available. XAML is used by other technologies as well although originally it was designed for WPF. Also using XAML in WPF projects is optional. Everything can be done in procedural code as well (althought its rare to find it done that way).

Chapter 13

Lecture 13

XAML specification defines rules that map .NET namespaces, types, properties, and events into XML namespaces, elements, and attributes. Let's see XAML and equivalent C#.

```
<Button xmlns="http://schemas.microsoft.com/winfx/2006/xaml/presentation" Content="OK"/>
```

And the corresponding C# code is

```
System.Windows.Controls.Button b= new System.Windows.Controls.Button();

B.Content= "OK";
```

Declaring an XML element in XAML (known as an object element) is equivalent to instantiating the corresponding .NET object via a default constructor. Setting an attribute on the object element is equivalent to setting a property of the same name (called a property attribute) or hooking up an event handler of the same name (called an event attribute).

```
<Button xmlns="http://schemas.microsoft.com/winfx/2006/xaml/presentation" Content="OK"
Click="button_Click"/ and the corresponding C# code is
System.Windows.Controls.Button b=new System.Windows.Controls.Button();
B.Click += new System.Windows.routedeventhandler(button_Click);

   B.Content = "OK";
```

XAML can no longer run standalone in the browser becuase of the butto click method — event handlers are attached before properties are set.

Let's discuss namespaces. Mapping to the namespace used above and other WPF namespaces is hard-coded in-side the WPF assemblies. Its just an arbitrary string like any namespace. The root object element in XAML must specify at least one XML namespace that is used to qualify itself and any child lements. Additional XML namespaces (on the root or on children) must be given a distinct prefix to be used on any identifiers from that namespace. E.g, typically a second namespace with the prefix x (denoted by using xmlns:x instead of just xmlns): xmlns:x="http://schemas.microsoft.com/winfx/2006/xaml"

This is the XAML language namespace, which maps to types in the System.Windows.Markup namespace but also defines some special directives for the XAML compiler or parser.

All of the following are mapped with http://schemas.microsoft.com/winfx/2006/xaml/presentation.

```
System.Windows
System.Windows.Automation
System.Windows.Controls
System.Windows.Controls.Primitives.System.Windows.Data
System.Windows.Documents
System.Windows.Forms.Integration
System.Windows.Ink
System.Windows.Input
System.Windows.Media
System.Windows.Media.Animation
System.Windows.Media.Effects
System.Windows.Media.Imaging
```

```
System.Windows.Media.Media3D
System.Windows.Media.textformatting.System.Windows.Navigation
System.Windows.Shapes
System.Windows.Shell
```

Property elements in XAML enable rich composition.

```
System.Windows.Controls.Button b= new System.Windows.Controls.Button();
System.Windows.Shapes.Rectangle r = new System.Windows.Shapes.Rectangle();
R.Width = 40;
R.Height = 40;
R.Fill= System.Windows.Media.Brushes.Black;
B.Content= r; // Make the square the content of the Button
```

And in XAML

```
<Button xmlns="http://schemas.microsoft.com/winfx/2006/xaml/presentation">
<Button.Content>
<Rectangle  Height="40" Width="40" Fill="Black"/>
</Button.Content ></Button >
```

The period distinguishes property elements from object elements. It doesn't have attributes except x:uid for localization. The syntax can be used for simple properties as well. The following two are equivalent.

```
<Button xmlns="http://schemas.microsoft.com/winfx/2006/xaml/presentation" Content="OK"
Backgroun
<Button xmlns="http://schemas.microsoft.com/winfx/2006/xaml/presentation">
<Button.Content>OK</Button.Content >
<Button.Background>White</Button.Background >
</Button >
```

Let's discuss type converters.

```
System.Windows.Controls.Button b= new System.Windows.Controls.Button();
B.Content= "OK";
B.Background =System.Windows.Media.Brushes.White;
```

WPF provides type converters for many common data types: Brush, Color, fontweight, Point, and so on. Classes deriving from typeconverter (brushconverter, colorconverter, and so on) convert from string to the corresponding types. You can also write your own type converters for custom data types.

```
<Button xmlns="http://schemas.microsoft.com/winfx/2006/xaml/presentation" Content="OK">
<Button.Background>
<solidcolorbrush  Color="White"/>
</Button.Background >
</Button>
```

Without the type converter for color

```
<Button xmlns="http://schemas.microsoft.com/winfx/2006/xaml/presentation" Content="OK">
<Button.Background>
<solidcolorbrush>
<solidcolorbrush.Color>
<Color A="255" R="255" G="255" B="255"/>
</solidcolorbrush.Color>
</solidcolorbrush>
</Button.Background>
</Button>
```

Even that requires a type converter for Byte.

```
System.Windows.Controls. Button b = new System.Windows.Controls. Button ();
B.Content = "OK";
```

```
B.Background = (Brush) System.componentmodel. Typedescriptor. Getconverter (
Typeof (Brush)). Converterfrominvariantstring ("White");
```

Constants in strings cause problems with run time exception not caught at compile time. Althought for xaml Visual Studio will check it at compile time. Let's see how to find the type converter for a type.

Let's discuss markup extensions now. Markup extensions, like type converters, enable you to extend the expres-siveness of XAML. Both can evaluate a string attribute value at runtime (except for a few built-in markup extensions for performance reasons) and produce an appropriate object based on the string. As with type converters, WPF ships with several markup extensions built in.

Unlike type converters, however, markup extensions are invoked from XAML with explicit and consistent syntax. For this reason, using markup extensions is a preferred approach for extending XAML.

For example, if you want to set a control's background to a fancy gradient brush with a simple string value, you can write a custom markup extension that supports it even though the built-in brushconverter does not. Whenever an attribute value is enclosed in curly braces ({}), the XAML compiler/parser treats it as a markup extension value rather than a literal string (or something that needs to be type-converted). Here are few examples:

```
<Button xmlns=''http://schemas.microsoft.com/winfx/2006/xaml/presentation''
Content=''{}{This is not a markup extension!}''/>
<Button xmlns=''http://schemas.microsoft.com/winfx/2006/xaml/presentation''>
<Button.Content>{This is not a markup extension!}</Button.Content> </Button>
```

Because markup extensions are just classes with default constructors, they can be used with property element syntax.

```
<Button xmlns="http://schemas.microsoft.com/winfx/2006/xaml/presentation"
            Xmlns:x="http://schemas.microsoft.com/winfx/2006/xaml"
            Background = "{x:Null}"
            Height = "{x:Static systemparameters.iconheight}"
            Content = "{Binding Path=Height, relativesource={relativesource
Mode=Self}}"/>
```

It works because positional parameters have corresponding property value. Markup extension has the real code to be executed.

An object element can have three types of children: a value for a content property, collection items, or a value that can be type-converted to the object element. Designated property is the content property.

```
<Button xmlns=''http://schemas.microsoft.com/winfx/2006/xaml/presentation''
Content=''OK''/>
```

Could be rewritten as follows:

```
<Button xmlns=''http://schemas.microsoft.com/winfx/2006/xaml/presentation''> OK
</Button>
```

Also

```
<Button xmlns=”http://schemas.microsoft.com/winfx/2006/xaml/presentation”>
<Button.Content>
<Rectangle Height=”40” Width=”40” Fill=”Black”/> </Button.Content>
</Button>
```

Could be rewritten as follows:

```
<Button xmlns=”http://schemas.microsoft.com/winfx/2006/xaml/presentation”> <Rectangle
Height=”40” Width=”40” Fill=”Black”/>
```

```
</Button>
```

There is no requirement that the content property must actually be called Content; classes such as combobox, listbox, and tabcontrol (also in the System.Windows.Controls namespace) use their Items property as the content property. Its designated with a custom attribute.

XAML enables you to add items to the two main types of collections that support indexing: lists and dictionaries.

```
<listbox xmlns=”http://schemas.microsoft.com/winfx/2006/xaml/presentation”>
<listbox.Items>
<listboxitem Content=”Item 1”/>
<listboxitem Content=”Item 2”/> </listbox.Items>
</listbox>
```

Its equivalent to:

```
System.Windows.Controls.listbox listbox= new System.Windows.Controls.listbox();
System.Windows.new System.Windows.Controls.listboxitem();
System.Windows.Controls.listboxitem item2= new System.Windows.Controls.listboxitem();
Item1.Content=Item 1 ;
Item2.Content= Item 2;
```

OR

```
<listbox xmlns=”http://schemas.microsoft.com/winfx/2006/xaml/presentation”> <listboxitem
Content=”Item 1”/>
<listboxitem Content=”Item 2”/>
</listbox>
```

Let’s see how to use a dictionary.

```
<resourcedictionary xmlns=”http://schemas.microsoft.com/winfx/2006/xaml/presentation”
xmlns:x=”http://schemas.microsoft.com/winfx/2006/xaml”>
<Color x:Key=”1” A=”255” R=”255” G=”255” B=”255”/>
<Color x:Key=”2” A=”0” R=”0” G=”0” B=”0”/>
</resourcedictionary>
```

We used the XAML Key keyword. The equivalent code is:

```
System.Windows. Resourcedictionary d = new System.Windows. Resourcedictionary ();
System.Windows.Media. Color color1 = new System.Windows.Media. Color ();
System.Windows.Media. Color color2 = new System.Windows.Media. Color ();
Color1.A = 255; color1.R = 255; color1.G = 255; color1.B = 255;
Color2.A = 0; color1.R = 0; color1.G = 0; color1.B = 0;
D.Add ("1", color1);
D.Add ("2", color2);
```

Chapter 14

Lecture 14

<solidcolorbrush>White</solidcolorbrush>

As discussed in the last lecture, this is equivalent to the following:

<solidcolorbrush Color="White"/>

It works because a typeconverter exists that converts string to solidcolorbrush, although no designated content property exists.
<Brush>White</Brush>

In fact, the above also works even although Brush is abstract. It works because of a type converter can convert a string to a solidcolorbrush.

XAML works with classes designated with marked attributes. Usually these classes have default ctors and useful properties. But what about other classes not designed for XAML. E.g. Hashtable.

```
System.Collections.Hashtable h
H.Add("key1",  7);
H.Add("key2",  23);
```

Heres how it can be represented in XAML:

```
<Collections: Hashtable
       Xmlns: collections = "clr-namespace: System.Collections; assembly = mscorlib"
       Xmlns: sys = "clr-namespace: System; assembly = mscorlib"
       Xmlns: x = "
<Sys: Int32 x: Key = "key1"> 7 </ sys: Int32>
<Sys: Int32 x: Key = "key2"> 23 </ sys: Int32>
</ Collections: Hashtable>
```

Let's study XAML child element rules. If the type implements ilist, call ilist.Add for each child. Otherwise, if the type implements idictionary, call idictionary.Add for each child, using the x:Key attribute value for the key and the element for the value. (Although XAML2009 checks idictionary before ilist and supports other collection interfaces, as described earlier.)

Otherwise, if the parent supports a content property (indicated by System.Windows.Markup.content property attribute) and the type of the child is compatible with that property, treat the child as its value. Otherwise, if the child is plain text and a type converter exists to transform the child into the parent type (and no properties are set on the parent element), treat the child as the input to the type converter and use the output as the parent object instance. Otherwise, treat it as unknown content and potentially raise an error

We can mix XAML and procedural code. Let's try this with xamlreader and xamlwriter.

```
Window window= null;
Using (filestream fs = new filestream("mywindow.xaml", filemode.Open, fileaccess.Read))
{
//Get the root element, which we know is a Window
Window= (Window)xamlreader.Load(fs);
}
```

Let's see how to grab the OK button by walking the children (with hard-coded knowledge!)

```
Stackpanel panel= (stackpanel)window.Content;
```

```
Button okbutton = (Button)panel.Children[4];
```

For easier fetching, we can name XAML elements.

```
<Button x:Name="okbutton">OK</Button>
```

Let's grab the OK button, knowing only its name

```
Button okbutton=(Button)window.findname("okbutton");
```

Classes that already have a name property designate it as the special name using a custom attribute.

Let's see how to use named properties. Let's first use a Binding markup extension.

```
<stackpanel
  Xmlns="http://schemas.microsoft.com/winfx/2006/xaml/presentation">
  <Label Target="{Binding elementname=box}" Content="Enter _text:"/>
  <textbox Name="box"/>
</stackpanel>
```

It gives it focus when the Label's access key is pressed.) WPF 4 includes a new, simpler markup extension which works at parse time.

```
<stackpanel
  Xmlns="http://schemas.microsoft.com/winfx/2006/xaml/presentation"
  Xmlns:x="http://schemas.microsoft.com/winfx/2006/xaml"> <Label
  Target="{x:Reference box}" Content="Enter _text:"/> <textbox
  Name="box"/>
  </stackpanel>
```

If a relevant property is marked with the System.Windows.Markup.namereferenceconverter type converter:

```
  <stackpanel
  Xmlns="http://schemas.microsoft.com/winfx/2006/xaml/presentation">
  <Label Target="box" Content="Enter _text:"/> <textbox Name="box"/>
  </stackpanel>
```

Compiling xaml involves 3 steps — converting a XAML file into a special binaryformat, embedding the converted content asa binary resource in the assembly being built,and performing the plumbing that connects XAML with procedural code automatically. C# and VB have best support in typical case, the first step is specifying a subclass for the root element in a XAML file. Use the Class keyword for that.

```
<Window xmlns="http://schemas.microsoft.com/winfx/2006/xaml/presentation"
Xmlns:x="http://schemas.microsoft.com/winfx/2006/xaml"
X:Class="mynamespace.mywindow"></Window >
Namespace  mynamespace
{
Partial class mywindow: Window
{
Public mywindow()
{
// Necessary to call in order to load XAML-defined content!
Initializecomponent();

}
Any other members can go here...
}
}
```

This is often refered as the code-behind file. We define event handlers here. The partial keyword is important. When we add new items to certain WPF projects, Visual Studio does all this automatically for us.

Chapter 15

Lecture 15

BAML is binary application markup language. It just a compressed representation of XAML. There is even a BAML reader available. Earlier there was CAML which stands for compiled application markup language but its not used now.

Some glue code is generated when we use x:Class. Its kind of same as loading and parsing the XAML file. We must call initializecomponent and we can refer named elements like class members.

Procedural code can even be written inside the XAML file.

```
<Window xmlns="http://schemas.microsoft.com/winfx/2006/xaml/presentation"
xmlns:x="http://schemas.microsoft.com/winfx/2006/xaml" x:Class="mynamespace.mywindow">
    <Button Click="button_Click">OK</Button>
    <x:Code>
        <![CDATA[
            Void button_Click(object sender, routedeventargs e)
            {
                This.Close();
            }
        ]]>
    </x:Code>
</Window>
```

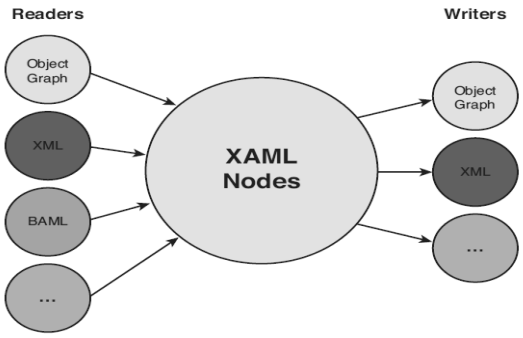
We must avoid]]. If we have to use them, we need &lt &amp etc. There is no good reason to embed code. Internally its build into .cs file by build system. BAML can be converted back to XAML.

```
System.Uri uri = new System.Uri("/wpfapplication1;component/mywindow.xaml",
System.urikind.Relative);
Window window= (Window)Application.loadcomponent(uri);
String xaml = xamlwriter.Save(window);
```

There are different loading mechanisms. We can load from resource, identified by original xaml file, or actually integrated baml loaded.

Key features of XAML 2009 include full generics support using typearguments, dictionary keys of any type, builtin system data types, creating objects with non-default ctors, getting instances via factory methods, event handlers can be markup extensions returning delegates, define additional members and properties.

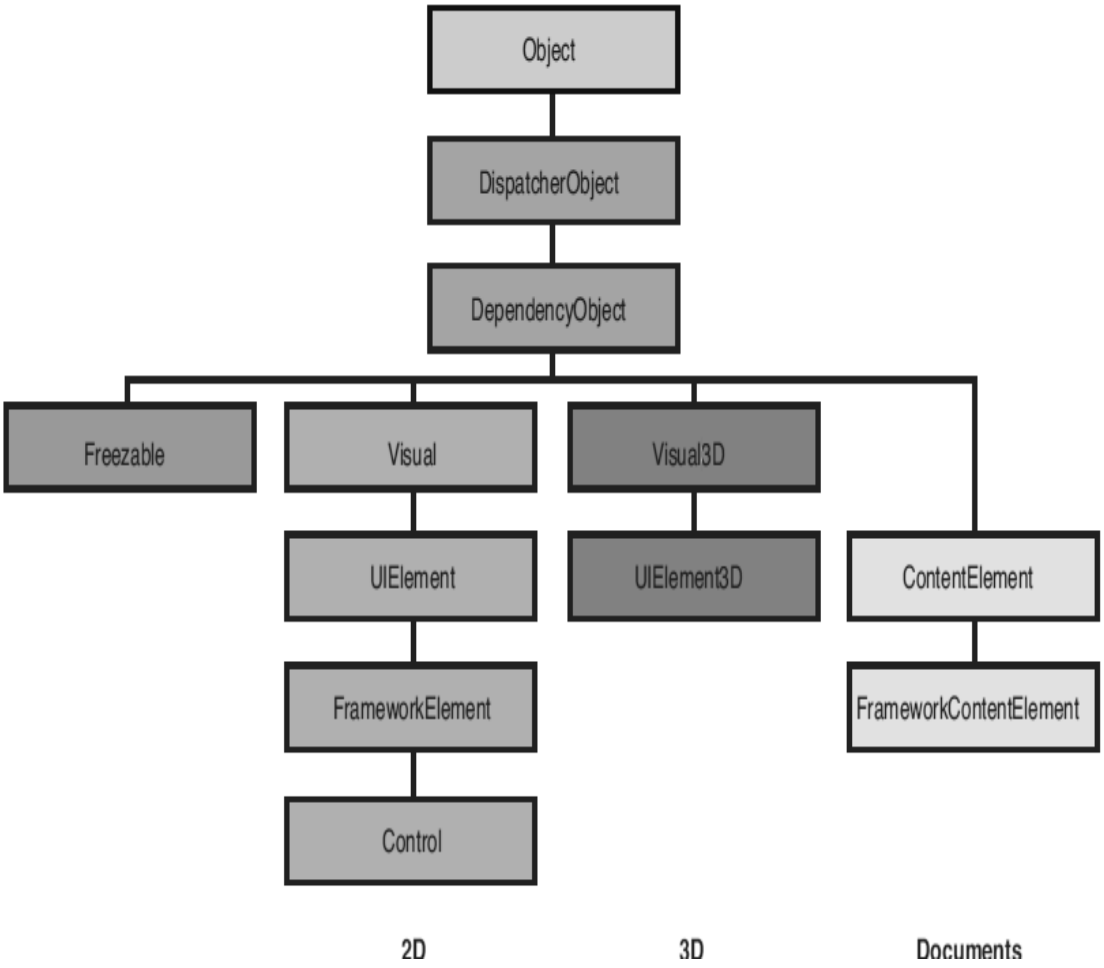
System.Xaml.xamlxmlreader etc. Can be extended with readers and writers for a lot of formats. It abstracts differences in Xaml formats like accessing a content property, property element, or a property attribute.



XAML 2006 keywords include x:asyncrecords - Controls the size of asynchronous XAML-loading chunks, x:Class, x:classmodifier - visibility, public by default, x:Code, x:connectionid - not for public use, x:fieldmodifier - field visibility (internal def), x:Key, x:Name, x:Shared - =false means same resource instance not shared, x:subclass - only needed when partial not supported, x:Synchronous mode - xaml loaded in async mode, x:typearguments - used only with root with x:Class in xaml2006, x:Uid - represents system.uri, and x:xdata - data opaque for xaml parser.

Markup extensions often confused as keywords are x:Array - use with x:Type to define type of array, x:Null, x:Reference - to named element, x:Static - static property field, and x:Type - just like typeof operator. In summary, we can convert xaml into C#, often apis are optimized for xaml and can look clunky in C#, wpf applications have deep hierarchy, small building blocks. Two complaints are that xml too verbose to type and slow to parse (tools and baml fix it partially).

Let's discuss wpf fundamentals now. WPF concepts are above and beyond .net concepts. Its the cause of the steep learning curve of wpf. It has a deep inheritance hierarchy. There are a handful of fundamental classes.



Object is the base class for all .NET classes and the only class in the figure that isn't WPF specific.

Dispatcherobject is the base class meant for any object that wishes to be accessed only on the thread that created it. Most WPF classes derive from dispatcherobject and are therefore inherently thread-unsafe. The Dispatcher part of the name refers to wpfs version of a Win32-like message loop, discussed further in

Chapter 7, “Structuring and Deploying an Application” of the book.

Dependencyobject is the base class for any object that can support dependency properties, one of the main topics in this chapter.

Freezable is the base class for objects that can be “frozen” into a read-only state for performance reasons. Freezables, once frozen, can be safely shared among multiple threads, unlike all other dispatcherobjects. Frozen objects can never be unfrozen, but you can clone them to create unfrozen copies. Most Freezables are graphics primitives such as brushes, pens, and geometries or animation classes.

Visual is the base class for all objects that have their own 2D visual representation. Visuals are discussed in depth in Chapter 15, “2D Graphics” of the book.

Uielement is the base class for all 2D visual objects with support for routed events, command binding, layout, and focus. These features are discussed in Chapter 5, “Layout with Panels,” and Chapter 6, “Input Events: Keyboard, Mouse, Stylus, and Multi-Touc.” Of the book.

Visual3D is the base class for all objects that have their own 3D visual representation. Visual3Ds are discussed in depth in Chapter 16, “3D Graphics” of the book.

Uielement3d is the base class for all 3D visual objects with support for routed events, command binding, and focus, also discussed in Chapter 16.

Contentelement is a base class similar to uielement but for document-related pieces of content that don’t have rendering behavior on their own. Instead, contentelements are hosted in a Visual-derived class to be rendered onthe screen. Each contentelement often requires multiple Visuals to render correctly (spanning lines, columns, and pages).

Frameworkelement is the base class that adds support for styles, data binding, resources, and a few common mechanisms for Windows-based controls, such as tooltips and context menus.

Frameworkcontentelement is the analog to frameworkelement for content. Chapter 11, “Images, Text, and Other Controls,” of the book examines the frameworkcontentelements in WPF.

Control is the base class for familiar controls such as Button, listbox, and statusbar. Control adds many properties to its frameworkelement base class, such as Foreground, Background, and fontsize, as well as the ability to be completely restyled. Part III, “Controls,” examines wpfs controls in depth.

Let’s discuss logical and visual trees.

Xaml good for UI because of hierarchical nature. Logical tree exists even if there is no xaml. Properties, events, resources are tied to logical trees. Properties propagated down and events can be routed up or down the tree. Its a simplification of whats actually going on when rendered. Visual tree can be thought of as an extension of the logical tree though some things can be dropped as well. Visual tree exposes visual implementation details e.g. A listbox is a border, two scrollbars and more. Only things from visual or visual3d appear in a visual tree. Avoid depending on visual tree in your code.

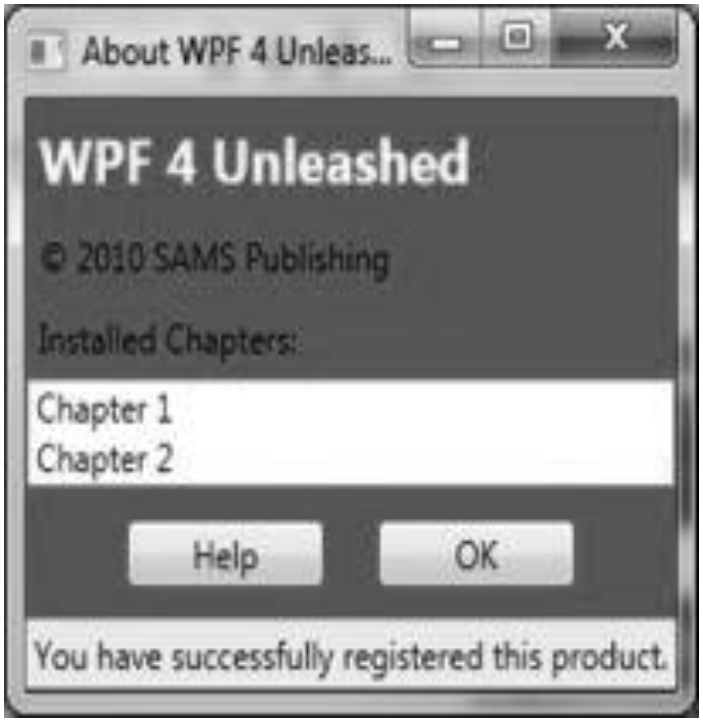
```
<Window xmlns="http://schemas.microsoft.com/winfx/2006/xaml/presentation" Title="About
WPF 4 Unleashed" sizetocontent="widthandheight" Background="orangered">

    <stackpanel>
        <Label fontweight="Bold" fontsize="20" Foreground="White">WPF 4 Unleashed
        </Label>
        <Label>© 2010 SAMS Publishing
        </Label>
```

```
        <Label>Installed Chapters:
        </Label>
        <listbox>
            <listboxitem>Chapter 1</listboxitem>
            <listboxitem>Chapter 2</listboxitem>
        </listbox>
        <stackpanel Orientation="Horizontal" horizontalalignment="Center">
            <Button minwidth="75" Margin="10">Help</Button>
            <Button minwidth="75" Margin="10">OK</Button>
        </stackpanel>
        <statusbar>You have successfully registered this product.</statusbar>
    </stackpanel>
</Window>
```

We can traverse using System.Windows.logicaltreehelper and System.Windows.Media.visualtreehelper. Here is a code-behind file that will print these trees, assuming the gluing code is there i.e. X:Class and xmlns:x.

```
Using System;
Using System.Diagnostics;
Using System.Windows;
Using System.Windows.Media; 5

Public partial class aboutdialog  : Window
{
Public  aboutdialog()
{
Initializecomponent();
Printlogicaltree(0, this);
}
Protected  override void oncontentrendered(eventargs e)
{
Base.oncontentrendered(e);
Printvisualtree(0, this);
}
Void printlogicaltree(int depth, object obj) 19 {
// Print the object with preceding  spaces that represent its depth
Debug.writeline(new string(’ ’, depth)+ obj);
// Sometimes leaf nodes  aren  t  dependencyobjects (e.g. Strings)
If (!(obj is dependencyobject))
Return;

// Recursive call for each logical child
Foreach (object child in logicaltreehelper.getchildren( obj as dependencyobject))
Printlogicaltree(depth+ 1, child);
}
Void printvisualtree(int depth, dependencyobject obj)
{
// Print the object with preceding  spaces that represent its depth
Debug.writeline(new string(’ ’, depth)+ obj);
// Recursive call for each visual child
For(int i= 0; i < visualtreehelper.getchildrencount(obj); i++)
Printvisualtree(depth + 1, visualtreehelper.getchild(obj, i));

}

}
```

Visual tree is empty until the dialog box is rendered. Navigating either can be done in instance methods of the elements themselves e.g. Visual class has protected members visualparent, visualchildrencount, and getvisualchild. Frameworkelement and frameworkcontentelement have Parent property and logicalchildren property.



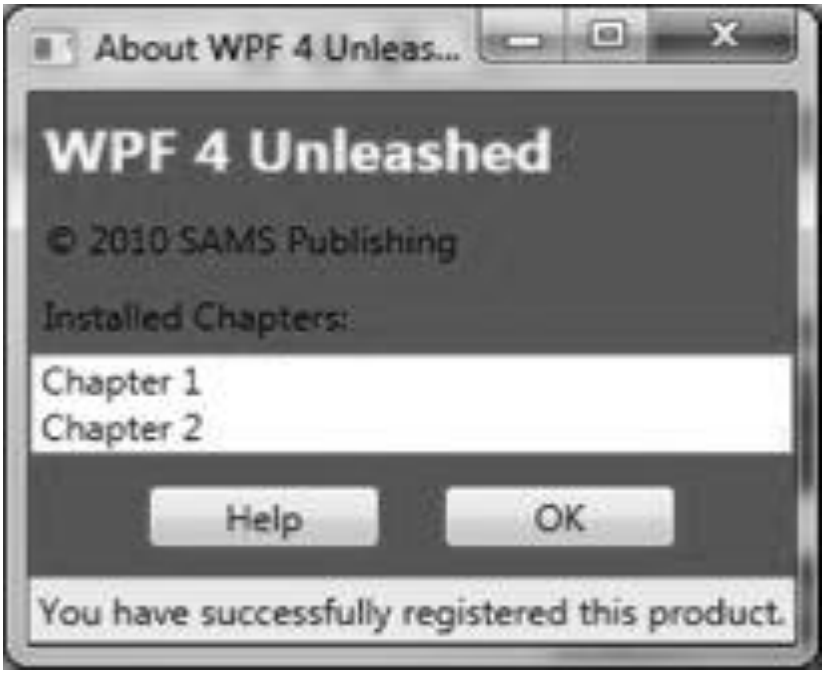
Chapter 16

Lecture 16

Let's discuss dependency properties. They complicates .net types but once you realize the problem it solves, you realize its importance. Dependency properties depends on multiple providers for determining its value at any point in time. These providers can be an animation continuously changing its values. A parent whose property propagates down. Arguably biggest feature is change notification. Motivation is to add rich functionality from declarative markup. Key to declarative-friendly design is heavy use of properties. Button e.g. Has 111 public properties (98 inherited). Properties can be set using xaml, directly or in a design tool, without procedural code. Without the extra work in dependency properties, it would be hard. We looked at implementation of a dependency property and then see its benefits on top of .net properties. Change notification, property value inheritance, and support for multiple providers are key features.

Understanding most nuances is important only for custom control authors. However what they are and how they work important for everyone. Can only style and animate dependency properties. After using wpf for a while, one wishes all properties were dependency properties. In practice, dependency properties are normal .net properties with extra wpf infrastructure. No .net language other than xaml has an intrinsic understanding of dependency properties.

Let's look at a standard dependency property implementation.




```
Public class Button: buttonbase
{
// The dependency  property
Public static readonly  dependencyproperty  isdefaultproperty;
Static Button()
{
```

```
// Register the property
Button.isdefaultproperty = dependencyproperty.Register("isdefault", typeof(bool), typeof

}
// A .NET property  wrapper(optional)
Public bool isdefault
{
Get
{
Return(bool)getvalue(Button.isdefaultproperty);
}
Set
{
Setvalue(Button.isdefaultproperty, value);
}
}
// A property  changed  callback(optional)
Private static void onisdefaultchanged(_dependencyobject o,
dependencypropertychangedeventar
{
//...
}
//...
}
```

Dependency properties are represented by System.Windows.dependencyproperty. By convention, public static and Property suffix are used. Its required by several infrastructure pieces e.g. Localization tools, xaml loading. Optionally (via different overloads of Register), you can pass metadata that customizes how the property is treated by WPF, as well as callbacks for handling property value changes, coercing values, and validating values. .net prop wrapper optional. It helps setting from xaml and isnatural. Otherwise getvalue and setvalue are to be used. Getvalue and setvalue methods inherited from System.Windows.dependencyobject. Getvalue setvalue does not support generic. Dependency properties were introduced before generic support was added in C#.

Visual Studio has a snippet called propdp that automatically expands into a definition of a dependency property, which makes defining one much faster than doing all the typing yourself! .NET property wrappers are bypassed at runtime when setting dependency prop- erties in XAML! Although the XAML compiler depends on the property wrapper at compile time, WPF calls the underlying getvalue and setvalue methods directly at runtime! Therefore, to maintain parity between setting a property in XAML and procedural code, its crucial that property wrappers not contain any logic in addition to the getvalue/setvalue calls. If you want to add custom logic, thats what the registered callbacks are for. All of wpfs built-in property wrappers abide by this rule, so this warning is for anyone writing a custom class with its own dependency properties.

On the surface, it too much code but saves per-instance cost. Only static field and an efficient sparse storage system. If all were .net props with backing store, would consume lot more space e.g. 111 fields for button, 104 for label but 89 and 82 are dependency properties. Also code to check thread access, prompt containing element to be re-rendered.

Dependency properties support change notification. Its based on metadata at register time. Actions can be re-rendering the appropriate elements, updating the current layout, refreshing data bindings, and much property triggers - imagine you want color change on hovering.

```
<Button  mouseenter="Button_mouseenter" mouseleave="Button_mouseleave" minwidth="75"
Margin="10">
<Button  mouseenter="Button_mouseenter" mouseleave="Button_mouseleave" minwidth="75"
```

```
Margin="10">

// Change the foreground to blue when the mouse enters the button
Void Button_mouseenter(object sender, mouseeventargs e)
{
Button b = sender as Button;
If(b != null)
B.Foreground= Brushes.Blue;
}
// Restore the foreground to black when the mouse exits the button
Void Button_mouseleave(object sender, mouseeventargs e)
{
Button b = sender as Button;
If(b != null)
B.Foreground= Brushes.Black;
}
```

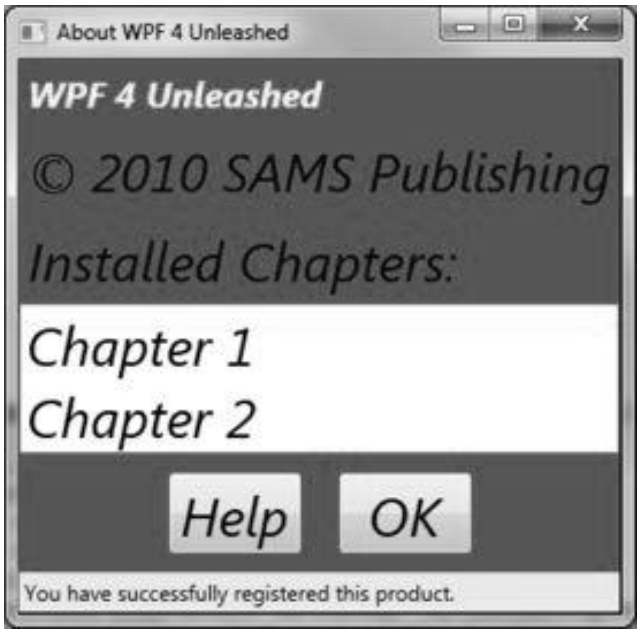
And with Xaml

```
<Trigger  Property="ismouseover" Value="True">
<Setter Property="Foreground" Value="Blue"/>
</Trigger >
Eed to worry about reverting but cannot be applied directly to button elements.

<Button minwidth="75" Margin="10">
<Button.Style>
<Style targettype="{x:Type Button}">
<Style.Triggers>
<Trigger  Property="ismouseover" Value="True">
<Setter Property="Foreground" Value="Blue"/>
</Trigger >
</Style.Triggers >
</Style >
</Button.Style >  OK
</Button >
```

Property value inheritance is flowing of prop values down an element tree.

```
<Window xmlns="http://schemas.microsoft.com/winfx/2006/xaml/presentation" Title="About WPF
 4 Unl
<stackpanel>
<Label fontweight="Bold" fontsize="20" Foreground="White">
WPF4 Unleashed
</Label >
<Label> 2010 SAMS Publishing</Label>
<Label>Installed  Chapters:</Label>
<listbox>
<listboxitem>Chapter 1</listboxitem>
<listboxitem>Chapter 2</listboxitem>
</listbox >
<stackpanel  Orientation="Horizontal" horizontalalignment="Center">
<Button minwidth="75" Margin="10">Help</Button>
<Button minwidth="75" Margin="10">OK</Button>
</stackpanel >
<statusbar>You have successfully  registered this product.</statusbar>
</stackpanel >
```

```
</Window >
```



It does not effect status bar. Not every dependency property participates in inheritance (Actually they can opt-in). May be other higher priority sources setting property value. Some controls like status bar internally set their values to system defaults. Property value inheritance in other places like to triggers inside a definition.

WPF contains many powerful mechanisms that independently attempt to set the value of dependency properties. Dependency properties were designed to depend on these providers in a consistent and orderly manner. Its a 5 step process

Step 1: determine base value. Most providers factor into base value determination. Highest to lowest precedence. Ten providers that can set value for most dep. Props. 1. Local valuedependencyobject.setvalue, prop assignment in xaml or2. Parent template trigger3. Parent template4. Style triggers5. Template triggers6. Style setters7. Theme style triggers8. Theme style setters9. Property value inheritancealready seen10. Default valueinitial value registered with the property. Thats why status bar did not get font propagated. Use dependencypropertyhelper.getvaluesource to find which was the source used.

Step 2: evaluate. Expressions need evaluation. It is used in data binding.

Step 3: apply animation. Animations can alter value from step 2 or replace it.

Step 4: coerce. The almost final value passed to coercevaluecallback delegate if one is registered.

Step 5: validate. Passed to validatevaluecallback delegate. If one was registered. Returning false causes exception canceling the entire process. Wpf 4 adds new value in dependencyobject called setcurrentvalue. It updates current value without changing value source.

Let's discuss attached properties now. They are special dependency properties that can be attached to arbitrary objects. Sounds strange but there are many applications for it. There is new xaml syntax for attached props.

```
<Window xmlns="http://schemas.microsoft.com/winfx/2006/xaml/presentation"
```

```
Title="About WPF4 Unleashed" sizetocontent="widthandheight"
Background="orangered">
<stackpanel>
<Label fontweight="Bold" fontsize="20" Foreground="White">
WPF4 Unleashed
</Label >
<Label> 2010 SAMS Publishing</Label>
<Label>Installed  Chapters:</Label>
<listbox>
<listboxitem>Chapter 1</listboxitem>
<listboxitem>Chapter 2</listboxitem>
</listbox >
<stackpanel  textelement.fontsize="30" textelement.fontstyle="Italic"
Orientation="Horizontal”>
<Button minwidth="75" Margin="10">Help</Button>
<Button minwidth="75" Margin="10">OK</Button>
</stackpanel >
<statusbar>You have successfully  registered this product.</statusbar>
</stackpanel >
</Window >
```

Chapter 17

Lecture 17

We discussed this example in the last lecture.

```
<Window xmlns="http://schemas.microsoft.com/winfx/2006/xaml/presentation"
Title="About WPF4 Unleashed" sizetocontent="widthandheight"
Background="orangered">
<stackpanel>
<Label fontweight="Bold" fontsize="20" Foreground="White">
WPF4 Unleashed
</Label >
<Label> 2010 SAMS Publishing</Label>
<Label>Installed  Chapters:</Label>
<listbox>
<listboxitem>Chapter 1</listboxitem>
<listboxitem>Chapter 2</listboxitem>
</listbox >
<stackpanel  textelement.fontsize="30" textelement.fontstyle="Italic" Orientation="Horizo
<Button minwidth="75" Margin="10">Help</Button>
<Button minwidth="75" Margin="10">OK</Button>
</stackpanel >
<statusbar>You have successfully  registered this product.</statusbar>
</stackpanel >
</Window >
```

Xaml parser requires setfontsize, setfontstyle on textelement and therefore equivalent code in C# is.

```
Stackpanel panel= new stackpanel();
Textelement.setfontsize(panel, 30);
Textelement.setfontstyle(panel, fontstyles.Italic);
Panel.Orientation = Orientation.Horizontal;
Panel.horizontalalignment= horizontalalignment.Center;
Button  helpbutton= new Button();
Helpbutton.minwidth= 75;
Helpbutton.Margin = new Thickness(10);
Helpbutton.Content = "Help";
Button okbutton= new Button();
Okbutton.minwidth = 75;
Okbutton.Margin= new Thickness(10);
Okbutton.Content= "OK";
Panel.Children.Add(helpbutton);
Panel.Children.Add(okbutton);
```

Enumeration values such as fontstyles.Italic, Orientation.Horizontal, and horizontalalignment.Center were pre-viously specified in XAML simply as Italic, Horizontal, and Center, respectively. This is possible thanks to the enumconverter type converter C#. C# code shows no real magic, no .net property involved internally. There are just calls to dependencyobject.setvalue and getvalue like normal property wrappers, must not go anything else.

```
Public static void setfontsize(dependencyobject element, double value)
{
Element.setvalue(textelement.fontsizeproperty, value);
}

Public static double  getfontsize(dependencyobject  element)
{
```

```
Return(double)element.getvalue(textelement.fontsizeproperty);
}
```

We are setting the fontsize property by unrelated class textelement. We could also have used textblock but textelement.fontsizeproperty is a separate dependencyproperty field from Control.fontsizeproperty.

```
Textelement.fontsizeproperty= dependencyproperty.registerattached(
 "fontsize", typeof(double), typeof(textelement), new frameworkpropertymetadata(
  Systemfonts.messagefontsize, frameworkpropertymetadataoptions.Inherits |
  Frameworkpropertymetadataoptions.affectsrender |
  Frameworkpropertymetadataoptions.affectsmeasure),
 New validatevaluecallback(textelement.isvalidfontsize));
```

Using registerattached us optimized for attached property metadata. Control on the other hand just calls addowner.

```
Control.fontsizeproperty = textelement.fontsizeproperty.addowner(
Typeof(Control), new frameworkpropertymetadata(systemfonts.messagefontsize,
Frameworkpropertymetadataoptions.Inherits));
```

These font related properties all controls inherit are from textelement in most cases, the class exposing the attached property is the same that defines the normal dependency propery. Many dependency properties have a Tag property for storing arbitrary custom data. A great mechanism for even extending "sealed" classes. In procedural code, you can actually use "any" property by calling setvalue.

```
// Attach an unrelated  property to a Button and set its value to true:
Okbutton.setvalue(itemscontrol.istextsearchenabledproperty, true);
```

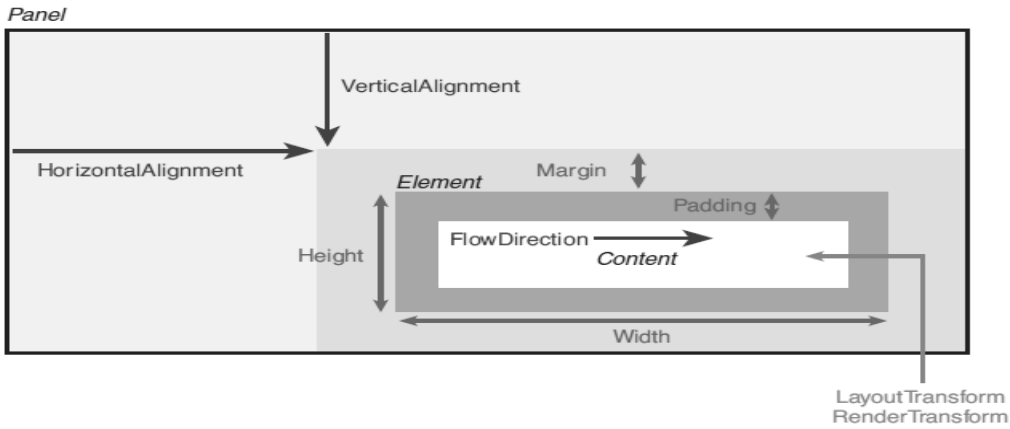
We can use this property any way we want but its better way to use Tag.

```
Geometrymodel3d  model= new geometrymodel3d();
Model.setvalue(frameworkelement.tagproperty, "my custom data");
```

Its most commonly used for layout of UI elements. Various panel derived classes define attached properties designed to be attached to their children. This way each panel can define custom behavior without burdening all possible children with their own set of properties. It allows extensibility.

In summary, wpf could have exposed its features by api's but instead they focused on adding features to the core. Multiple types of properties, multiple tree, multiple ways to achieve the same thing. Hopefully you can appreciate the value in coming lectures. These concepts will fade in the background as we accomplish different tasks in coming lectures.

Let's discuss sizing, positioning, and transforming elements. Sizing and positioning called "layout".

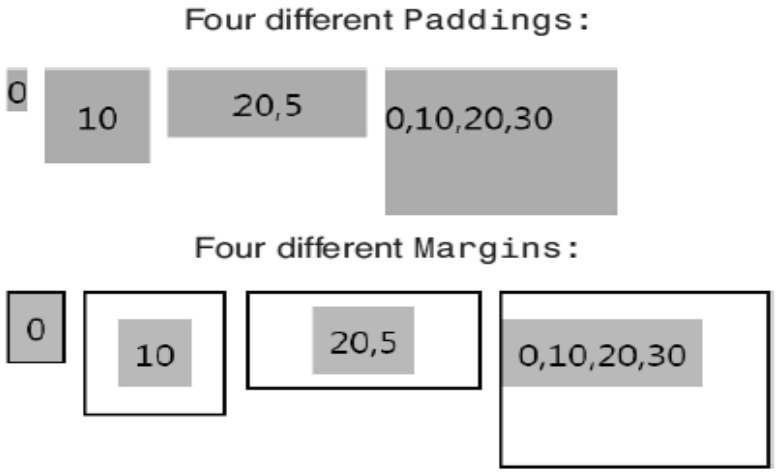


It boils down to parent child relationships. They work together to determine their final sizes and positions. Parents tell where to render and how much space they get. But more like collaborators they also ask children how much they really need. Panels are parents supporting layout of multiple children. They derive from the abstract System.Windows.Controls.Panel class. All the elements involved in the layout process derive from System.Windows.uielement. Several properties control various aspects of layout.

Size related properties are shown in blue, position related in red, green transforms can effect both. We will discuss various panels that arrange their children in specific ways.

Let's first discuss size. Children tell their size to parents. Wpf elements tend to size to content and no larger. Even the whole window when you set sizetocontent property. All frameworkelements have Height and Width properties of type double. Also have minheight, maxheight, minwidth, maxwidth. If explicit height width they takeprecedence when in range (avoid explicit). Min=0 and max=Infinity by default. Nan, Auto, Double.isnan mean size to content. Read only properties desiredsize, rendersize, actualheight, actualwidth. They are useful to act programatically on final size.

Next is margin and padding. All framework elements have margin, controls (and Border) have padding. Margin is extra space "outside" and padding is "inside" edges of the element. They are of type Thickness that can represent 1, 2, or 4 values. Negative margin is possible. Label's default padding=5. Xaml specification is helped by typeconverter.



```
Mylabel.Margin = new   Thickness (10); //(10); // Same as Margin="10" in XAML
Mylabel.Margin = new Thickness (20,5,20,5); // Same as Margin="20,5" in XAML
Mylabel.Margin = new Thickness (0,10,20,30);// Same as Margin="0,10,20,30" in XAML
```

The lengthconverter type converter associated with the various length properties supports specifying explicit units of cm, pt, in, or px (the default). Default are device independent pixels. These logical pixels are 1/96 in regardless of screen DPI setting. They are always stored as "double". A typical display is 96 DPI. Important to note that all measurements are DPI independent.

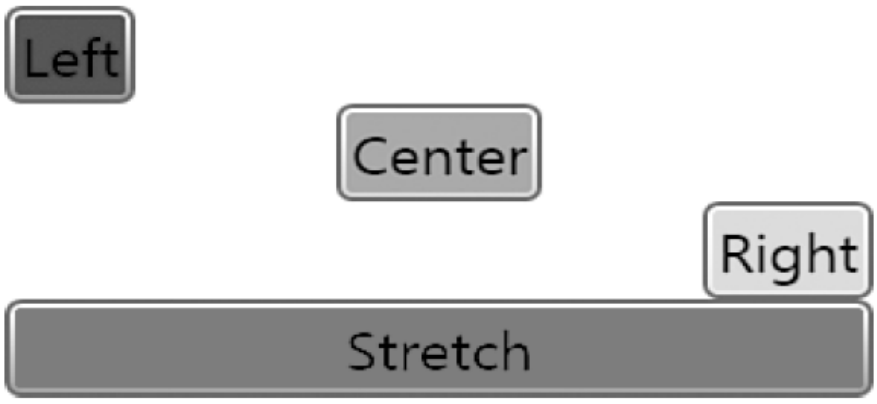
Next is Visibility. It can have three values: Visible - The element is rendered and participates in layout, Collapsed The element is invisible and does not participate in layout, Hidden - The element is invisible yet still participates in layout.

Compare the following two stackpanels.

```
<stackpanel  Height="100"  Background="Aqua">
<Button  Visibility="Collapsed">Collapsed Button</Button>
<Button>Below a Collapsed Button</Button>
</stackpanel >

<stackpanel  Height="100"  Background="Aqua">
<Button  Visibility="Hidden">Hidden Button</Button>
<Button>Below a Hidden Button</Button>
</stackpanel >
```

Next is Position and alignmnet. Position is not given as (x,y), instead its parent controlled. Rather alignment and flow direction is given vs. Explicit height. Horizontalalignment can taken values Left, Center, Right, and Stretch and verticalalignment can take values Top, Center, Bottom, and Stretch.

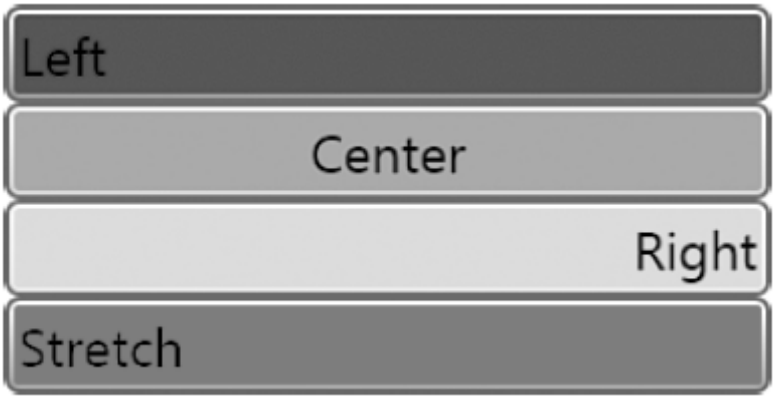
```
<stackpanel>
<Button  horizontalalignment="Left" Background="Red">Left</Button>
<Button  horizontalalignment="Center" Background="Orange">Center</Button>
<Button  horizontalalignment="Right" Background="Yellow">Right</Button>
<Button  horizontalalignment="Stretch" Background="Lime">Stretch</Button>
</stackpanel >
```



Its only useful when parent gives more space than needed.

Control class also has horizontalcontentalignment and verticalcontentalignment. It sets how a control content fills the space inside, more like the relationship between margin n padding. Default is left and top except button which overrides it. Textblock doesnt strech like a control.

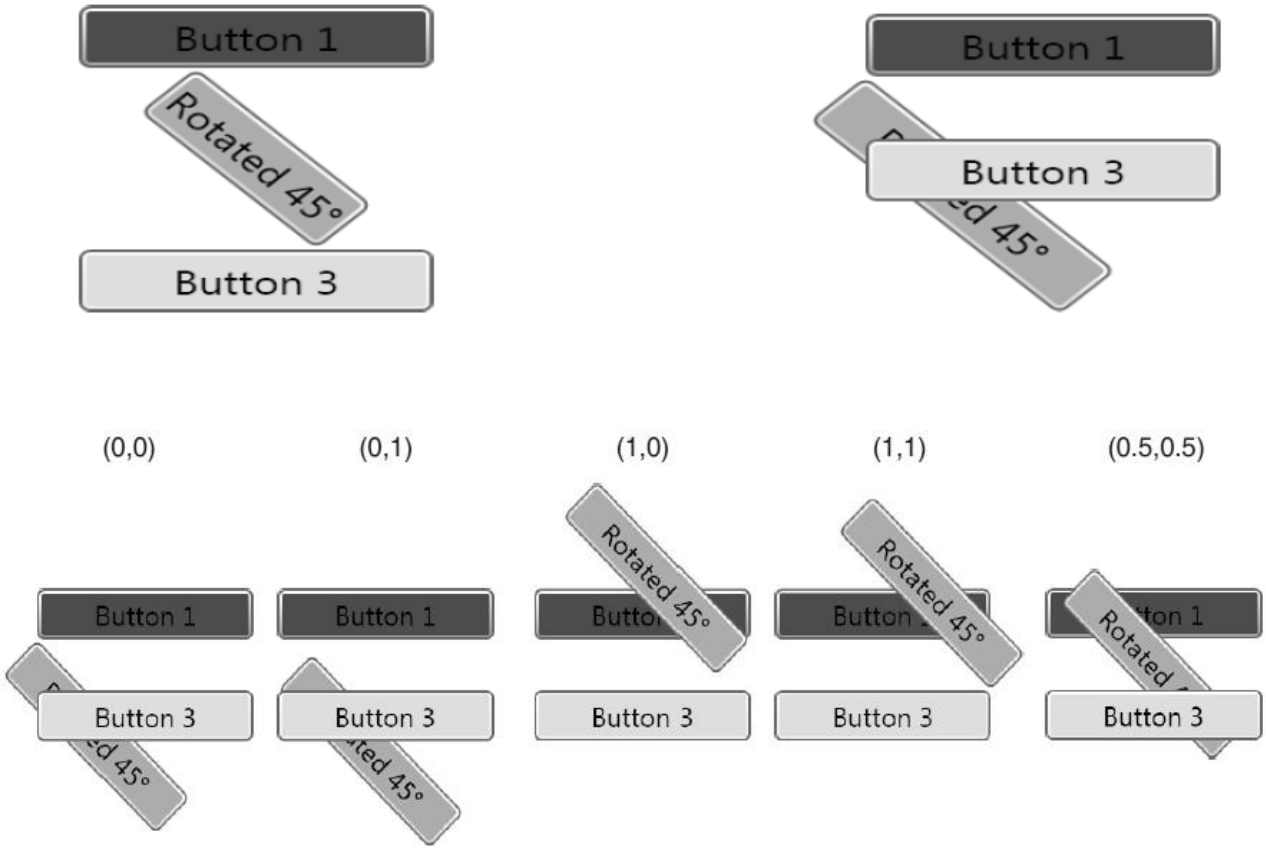
```
<stackpanel>
<Button  horizontalcontentalignment="Left" Background="Red">Left</Button>
<Button  horizontalcontentalignment="Center" Background="Orange">Center</Button>
<Button  horizontalcontentalignment="Right" Background="Yellow">Right</Button>
<Button  horizontalcontentalignment="Stretch" Background="Lime">Stretch</Button>
</stackpanel >
```



Frameworkelement.flowdirection can reverse the direction. It can be used for languages that are read right to left.

```
<stackpanel>
<Button  flowdirection="lefttoright" horizontalcontentalignment="Left"
verticalcontentalignme
<Button  flowdirection="righttoleft" horizontalcontentalignment="Left"
verticalcontentalignme
</stackpanel >
```

Transforms are from the base class System.Windows.Media.Transform. All framework elements have layouttransform and rendertransform. One applied before layout and one after. Uielements have rendertransformorigin.


Button 1
Rotated 45°
Button 3
Button 1
Button 3
(0,0)
(0,1)
(1,0)
(1,1)
(0.5,0.5)
Button 1
Button 3
Button 1
Button 3
Button 3
Button 3
Button 3

Chapter 18

Lecture 18

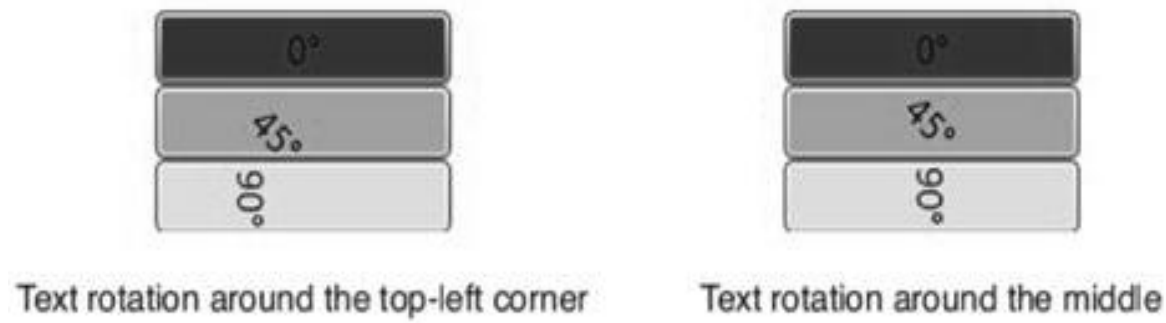
Let's revise size and position properties.
Layout transform vs render transform. One is applied before rendering and one after.

Pointconverter is used to specify the origin.

```
<Button  rendertransformorigin="0.5,0.5" Background="Orange">
<Button.rendertransform>
<rotatetransform  Angle="45"/>
</Button.rendertransform >
Rotated 45
</Button >
```

Builtin transforms are Rotate Transform, Scale Transform, Skew Transform, Translate Transform, and Matrix Transform. Rotatetransform has Angle, centerx, centery with defaults 0. Center useless for layout transform. Center is also useful when grouping rendertransforms.
Transform can be on inner element.

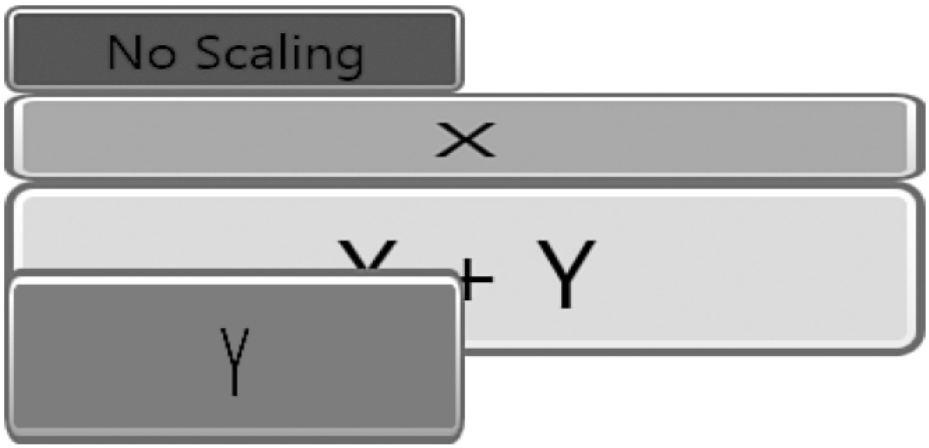
```
<Button  Background="Orange">
<textblock  rendertransformorigin="0.5,0.5">
<textblock.rendertransform>
<rotatetransform  Angle="45"/>
</textblock.rendertransform >
45
</textblock >
</Button >
```



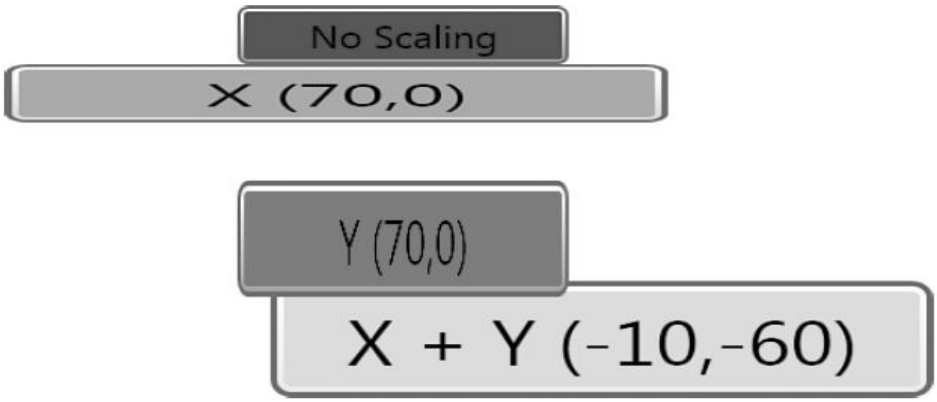
Scale Transform has scalex, scaley, centerx, centery.

```
<stackpanel Width="100">
  <Button  Background="Red">No Scaling</Button>
  <Button  Background="Orange">
    <Button.rendertransform>
      <scaletransform  scalex="2"/>
    </Button.rendertransform >
    X
  </Button >
  <Button  Background="Yellow">
    <Button.rendertransform>
      <scaletransform  scalex="2" scaley="2"/>
    </Button.rendertransform >
```

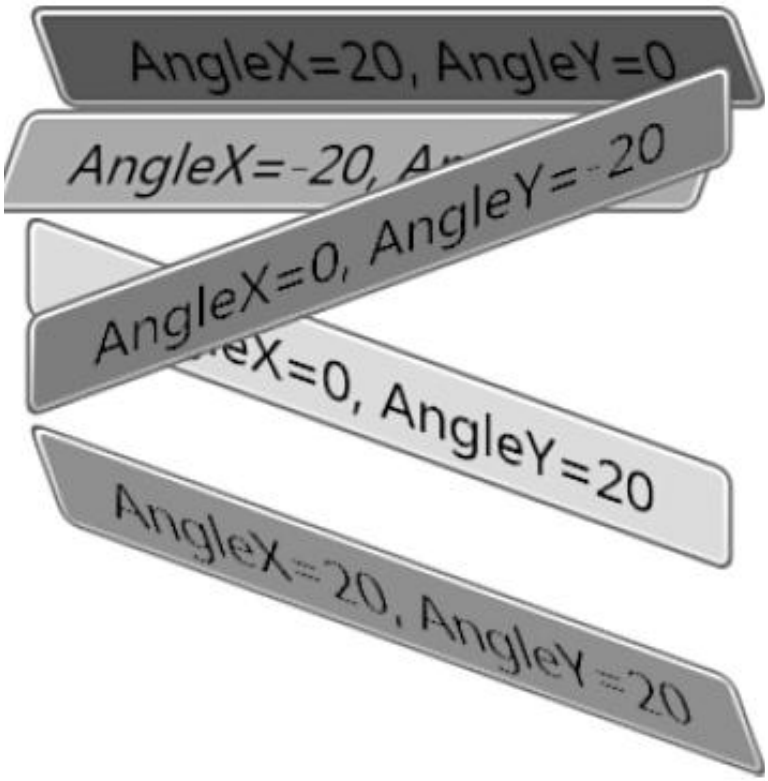
```
    X + Y
  </Button >
  <Button  Background="Lime">
    <Button.rendertransform>
      <scaletransform  scaley="2"/>
    </Button.rendertransform >
    Y
  </Button> 21
</stackpanel >
```



Stretch and scaletransform only effect if more than stretch. Padding is scaled but margin is not. Does not effect actualhieght actualwidth rendersize etc.
Skew Transform has anglex — Amount of horizontal skew (default value = 0), angley — Amount of vertical skew



(default value = 0), centerx — Origin for horizontal skew (default value = 0), centery — Origin for vertical skew (default value = 0). It is applied as a render transform here.



Translate Transform has X — Amount to move horizontally (default value = 0), Y — Amount to move vertically (default value = 0). It has no effect as a layout transform.

Matrix Transform has a single Matrix property (of type System.Windows.Media.Matrix) representing a 3x3 affine transformation matrix.

```
<Button  rendertransform=1,0,0,1,10,20/>
```

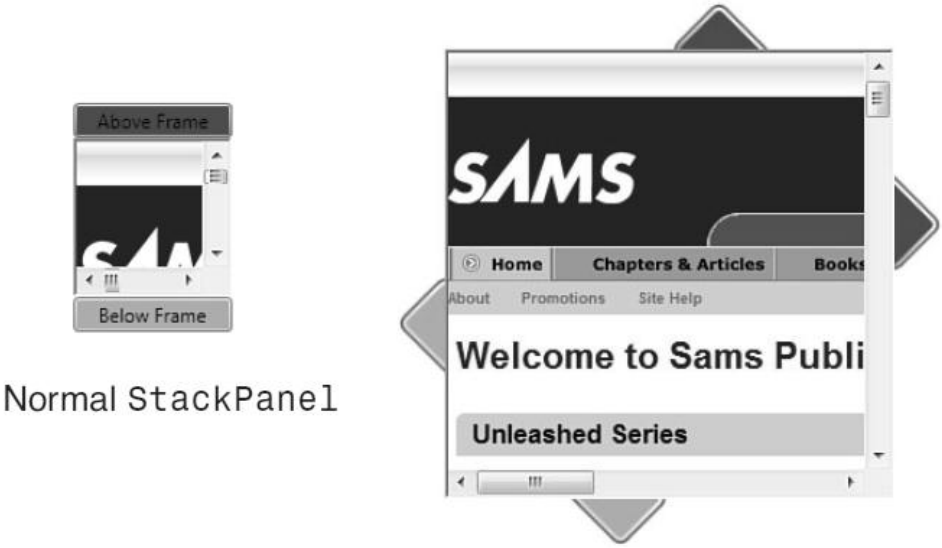
M11	M12	0
M21	M22	0
OffsetX	OffsetY	1

Its the only transform with a type converter to convert a string.

Combining transforms can be done. We can apply both layout and render transforms. Can figure out a

combined matrix and apply matrix transform or use transformgroup a Transform derived class and let it be calculated for you.

```
<Button>
  <Button.rendertransform>
    <transformgroup>
      <rotatetransform  Angle="45"/>
      <scaletransform  scalex="5" scaley="1"/>
      <skewtransform  anglex="30"/>
    </transformgroup >
  </Button.rendertransform >
  OK
</Button >
```



Not all elements can be transformed.

In Summary, we learned about layout of individual elements. The layout is finally decided by parent. We have used stackpanel. Let's see this and other panels for layout.

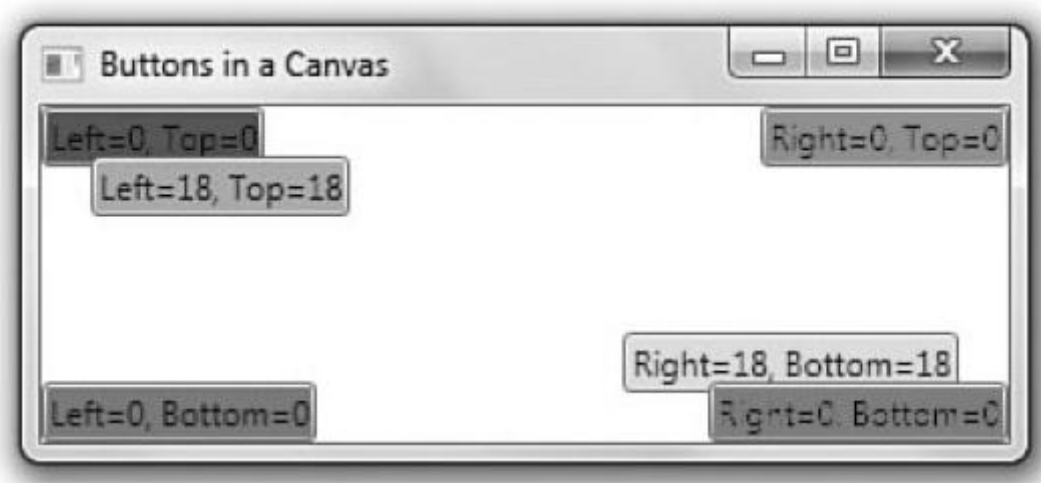
Static pixel based placement and sizes results in limited but different screen sizes. Builtin panels that make it easy.

5 main builtin panels in System.Windows.Controls: Canvas, stackpanel, wrappanel, dockpanel, and Grid. Content overflow is when parents and children cant agree on the use of available space.

Canvas is the most basic. Probably never used. "classic" notion of explicit coordinates but device independent pixels and relative to any corner of the window. We can position elements in a Canvas by using its attached properties: Left, Top, Right, and Bottom. Default offsets are from top left.

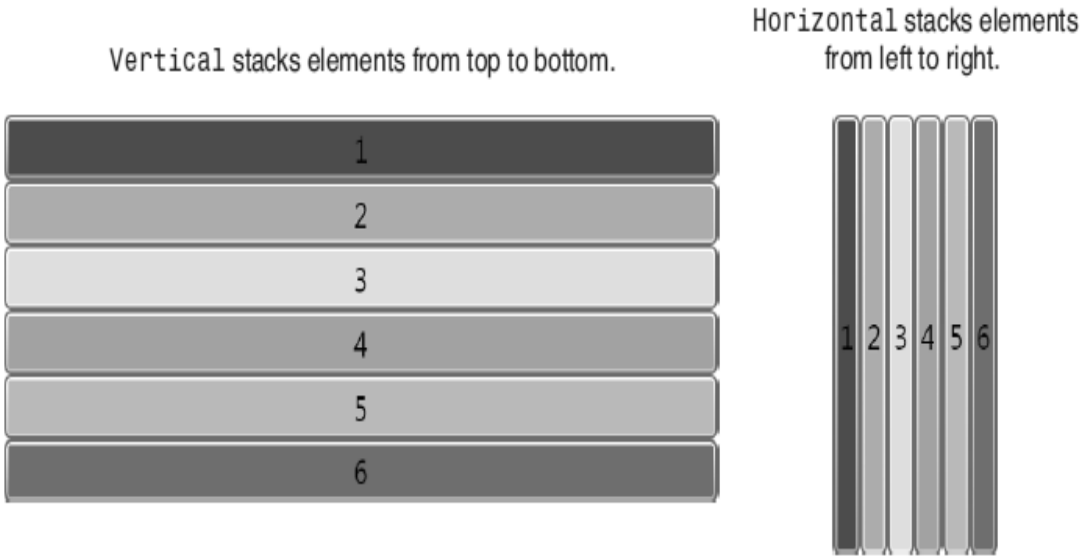
```
<Window xmlns="http://schemas.microsoft.com/winfx/2006/xaml/presentation"
Title="Buttons in a Canvas">
  <Canvas>
    <Button  Background="Red">Left=0, Top=0</Button>
    <Button Canvas.Left="18" Canvas.Top="18"
    Background="Orange"> Left=18, Top=18</Button>
    <Button Canvas.Right="18" Canvas.Bottom="18"
    Background="Yellow">Right=18, Bottom=18</Button>
```

```
    <Button Canvas.Right="0" Canvas.Bottom="0"
    Background="Lime">Right=0, Bottom=0</Button>
    <Button Canvas.Right="0" Canvas.Top="0"
    Background="Aqua">Right=0, Top=0</Button>
    <Button Canvas.Left="0" Canvas.Bottom="0"
    Background="Magenta">Left=0, Bottom=0</Button>
  </Canvas ></Window >
```

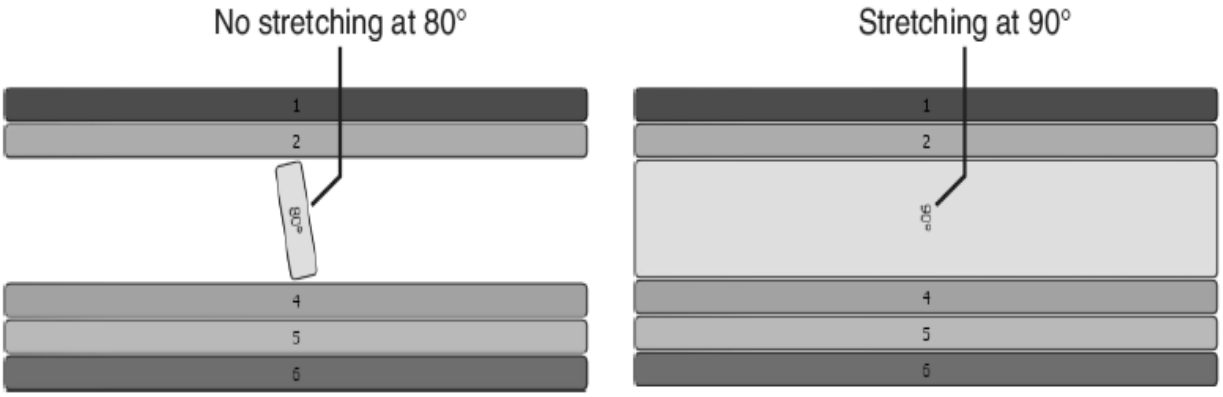


If left and right, right is ignored, top and bottom, bottom is ignored. Whats z order. Z-order decides which element to show in overlapping elements.

```
<Canvas>
  <Button Canvas.zindex="1" Background="Red">On Top!</Button>
  <Button  Background="Orange">On Bottom with a Default zindex=0</Button>
</Canvas >
```



Stack Panel is popular, simple, useful, stacks sequentially. It is one of the few panels that dont even define an attached property. Orientation can be horizontal or vertical (which is default). Default horizontal direction is basedon flowdirection.



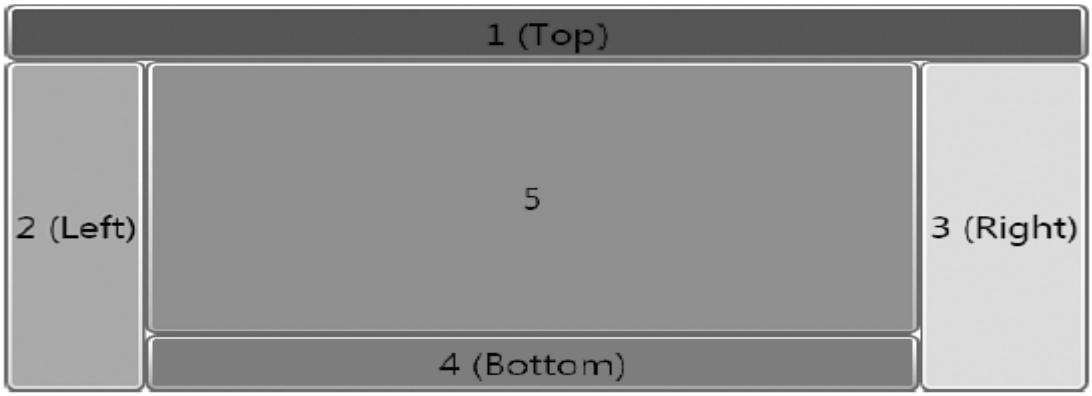
Virtualizing panels e.g. Virtualizingstackpanel save space for offscreen content when data binding e.g. Listbox uses it Wrap panel wraps to additional rows or columns when not enough space. It has no attached properties. Orientation is by default horizontal. Itemheight, itemwidth are uniform for all children and not set by default.

Chapter 19

Lecture 19

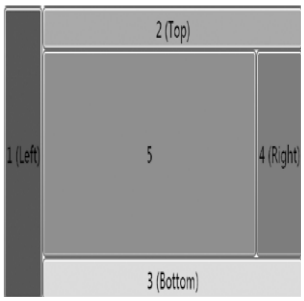
Dock panel allows easy docking of elements to an entire side. Dock attached property has left, right, top, bottom values. Last child fills space unless lastchildfill=false.

```
<dockpanel>
  <Button dockpanel.Dock="Top" Background="Red">1(Top)</Button>
  <Button dockpanel.Dock="Left" Background="Orange">2 (Left)</Button>
  <Button dockpanel.Dock="Right" Background="Yellow">3 (Right)</Button>
  <Button dockpanel.Dock="Bottom" Background="Lime">4 (Bottom)</Button>
  <Button  Background="Aqua">5</Button>
</dockpanel >
```



Why are elements streching. Let's change alignment and see.

```
<dockpanel>
  <Button dockpanel.Dock="Top" horizontalalignment=”Right” Background=”Red”>1 (Top,
Align=Right) </Button>
  <Button dockpanel.Dock=”Left” verticalalignment=”Bottom” Background=”Orange”>2 (Left,
Align=Bottom)</Button>
  <Button dockpanel.Dock=”Right” verticalalignment=”Bottom” Background=”Yellow”>3 (Right,
Align=Bottom)</Button>
  <Button dockpanel.Dock=”Bottom” horizontalalignment=”Right” Background=”Lime”>4
(Bottom, Align=Right)</Button>
  <Button Background=”Aqua”>5</Button></dockpanel>
```

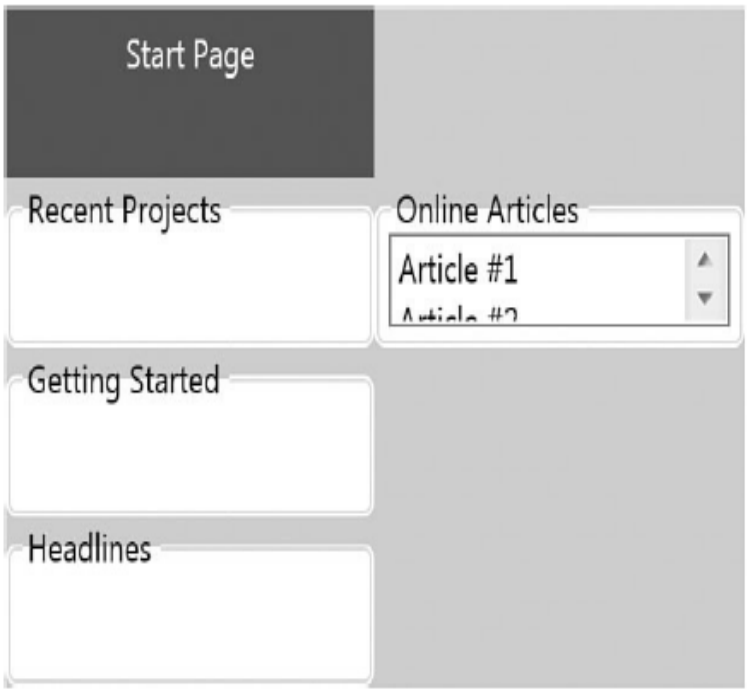


Its useful in a toplevel interface. Order of adding matters for the corners.

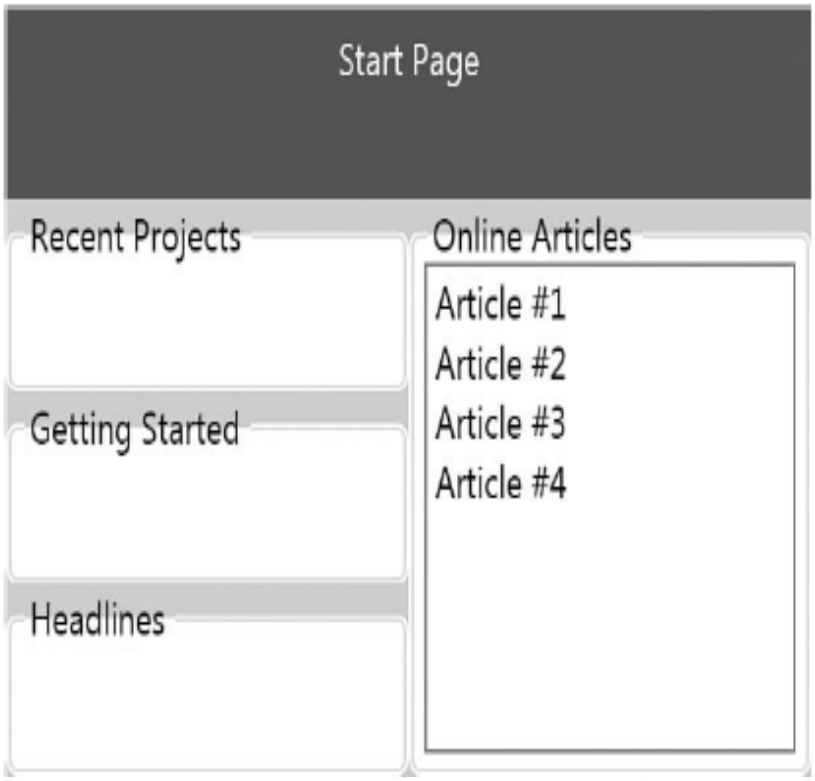
More elements can be added to any side. Its a superset of stackpanel.

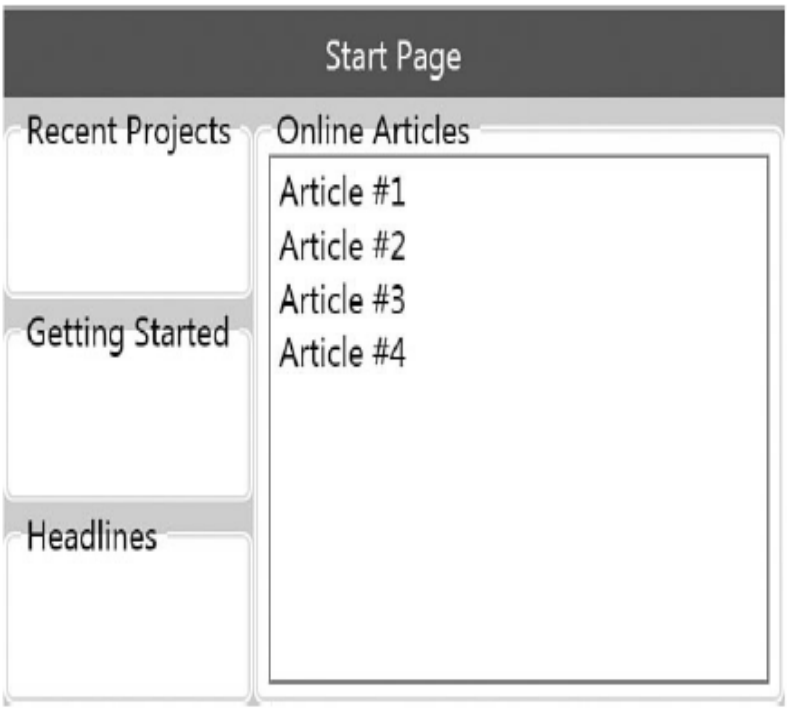
Grid is the most versatile. Its the default panel in VS and expression blend. You arrange things in multirow multicolumn, like a table in html. Also a Table class which is a frameworkcontentelement not a uielement.

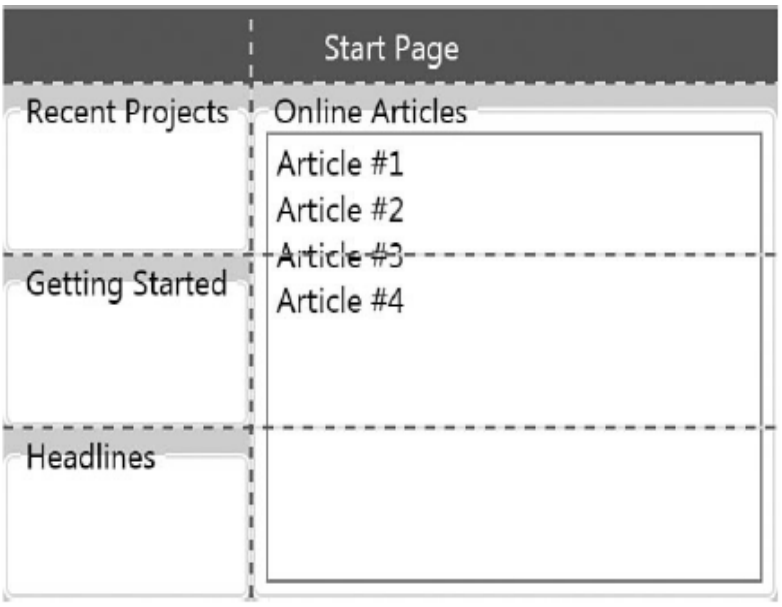
```
<Grid Background="lightblue">
  <!--  Define  four  rows: -->
  <Grid.rowdefinitions>
    <rowdefinition/>
    <rowdefinition/>
    <rowdefinition/>
    <rowdefinition/>
  </Grid.rowdefinitions >
  <!--  Define  two  columns:-->
  <Grid.columndefinitions>
    <columndefinition/>
    <columndefinition/>
  </Grid.columndefinitions >
  <!--  Arrange  the  children: -->
  <Label   Grid.Row="0" Grid.Column="0" Background="Blue" Foreground="White"
horizontalcontenta=""
<groupbox Grid.Row="1" Grid.Column="0" Background="White" Header="Recent Projects">
 </groupbox>
  <groupbox Grid.Row="2" Grid.Column="0" Background="White" Header="Getting
Started"></groupbox>
  <groupbox Grid.Row="3" Grid.Column="0" Background="White"
Header="Headlines"></groupbox>
  <groupbox Grid.Row="1" Grid.Column="1" Background="White" Header="Online
Articles"></groupbox>
  20 <listbox>
    <listboxitem>Article #1</listboxitem>
    <listboxitem>Article #2</listboxitem>
    <listboxitem>Article #3</listboxitem>
    <listboxitem>Article #4</listboxitem>
  </listbox >
  </groupbox >
</Grid >
```



We specify individual rows and cols. Its verbose but useful to specify details. Default is 1 cell. We position elements using Row and Column attached properties (zero-based) default is 0,0. Multiple elements in same cell simply overlap by z-order. Cells can be empty.







Online article list too small and and start page label not full width. We can make spanning rows and cols. Rowspan and columnspan 1 by default. By default height n width are same. Height and Width = "Auto" for sizing to content. Showgridlines = "True" to show grid lines.

For sizing rows and columns, use rowdefinition and Col. Height and Width not double but gridlength and not default to Auto or nan. Three types of sizing. Absoulte sizing: device independent pixels means no grow shrink, Autosizing: size to content, or Proportional or star sizing: grows or shrinks. When 1 row col * all remaining space is taken by it. When more row col * they divides remaining space. It can an be 2* or

5.5*. 2* is twice as * (1*). 5.5* is twice as 2.75*. Remaining space is after absolute and autosized rows or cols. Default width and height are *.

Here is how gridlength is used in procedural code.


```
Gridlength  length = new gridlength(100);
Gridlength  length = new gridlength(0, gridunittype.Auto);
Gridlength  length = new gridlength(100, gridunittype.Pixel);
Gridlength  length = new gridlength(2, gridunittype.Star);
```

Grid splitter is used for interactive sizing. Any number of gridsplitters can be added to a Grid. Can be used to move entire row or column. Atleast one cell resizes and other cells behavior depends on weather absolute sizing or proportional is applied on them. It fits in one cell, but behavior affects entire row or col so better to use span. Which cells are affected depends on gridsplitters alignment values. Default horizontal alignment=right and vertical alignment=stretch. Reasonable use requires stretch in one dimension.

HorizontalAlignment

VerticalAlignment	Left	Right	Center	Stretch
Top	Current cell and cell to the left	Current cell and cell to the right	Cells to the left and right	Current cell and cell above
Bottom	Current cell and cell to the left	Current cell and cell to the right	Cells to the left and right	Current cell and cell below
Center	Current cell and cell to the left	Current cell and cell to the right	Cells to the left and right	Cells above and below
Stretch	Current cell and cell to the left	Current cell and cell to the right	Cells to the left and right	Cells to the left and right if `GridSplitter` is taller than it is wide, *or* cells to the top and bottom if `GridSplitter` is wider than it is tall

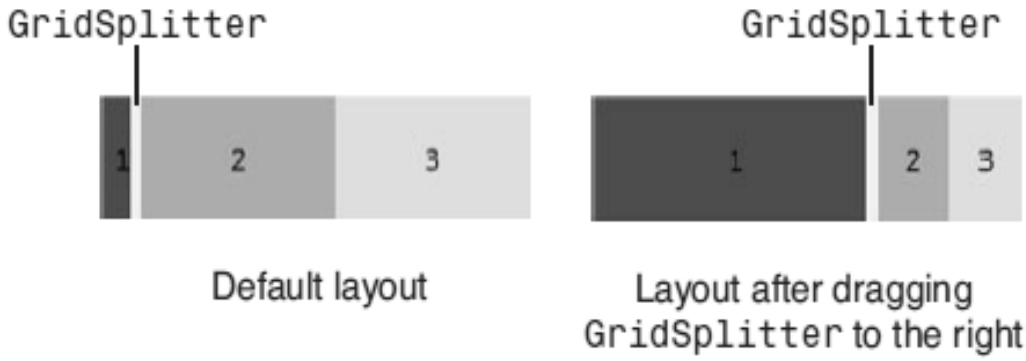
When all proportionally sized, changes co-efficients accordingly. When absolute sizing, changes only top or left of the cells. Remaining cells pushed down or right. Also has resize direction (default Auto, or can be set to Rows or Cols) and resize behavior for explicit control. Resize direction effect only when stretching in both directions. Resize behavior defaults to based on alignment. It can be set to previous and current, current and next, or previous and next to control which 2 rows or columns should be directly affected by resizing. Best way is to place it in its own auto sized row or column so it doesn't overlap existing content.

Chapter 20

Lecture 20

Let's revise grid.

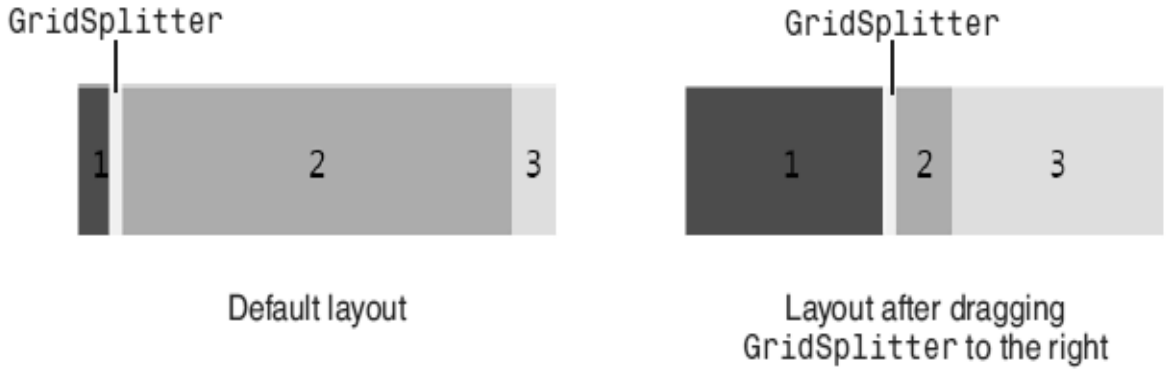
```
<Grid>
  <Grid.columndefinitions>
    <columndefinition  Width="Auto"/>
    <columndefinition/>
    <columndefinition/>
  </Grid.columndefinitions >
  <Label Grid.Column="0" Background="Red" horizontalcontentalignment="Center"
verticalcontenta=""
  </Label >
  <gridsplitter Grid.Column="0" Width="5"/>
  <Label Grid.Column="1" Background="Orange" horizontalcontentalignment="Center"
verticalconte=""
  </Label >
  <Label Grid.Column="2" Background="Yellow" horizontalcontentalignment="Center"
verticalconte=""
  </Label >
</Grid >
```



Sharedsizegroup enables multiple row cols to remain the same width height when length changed via gridsplitter.

```
<Grid issharedsizescope="True">
  <Grid.columndefinitions>
    <columndefinition  Width="Auto" sharedsizegroup="mygroup"/>
    <columndefinition/>
    <columndefinition  sharedsizegroup="mygroup"/>
  </Grid.columndefinitions >
  <Label Grid.Column="0" Background="Red" horizontalcontentalignment="Center"
verticalcontenta=""
  </Label >
  <gridsplitter Grid.Column="0" Width="5"/>
  <Label Grid.Column="1" Background="Orange" horizontalcontentalignment="Center"
verticalconte=""
  </Label >
  <Label Grid.Column="2" Background="Yellow" horizontalcontentalignment="Center"
verticalconte=""
```

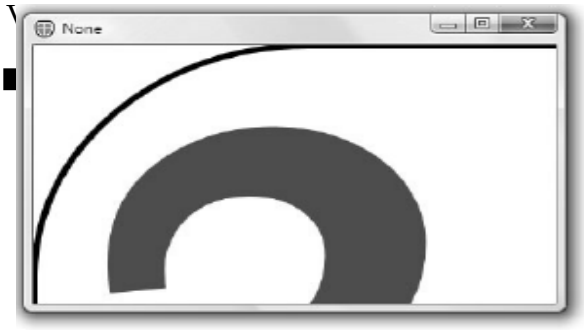
```
  </Label >
</Grid >
```



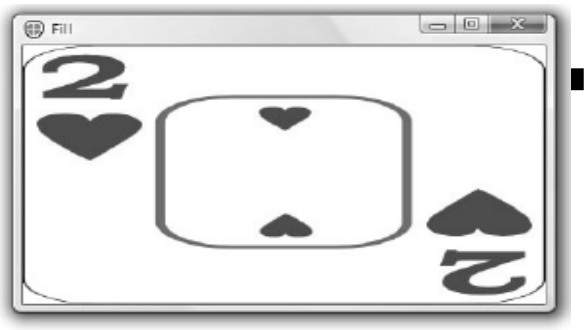
Issharedscope because sizegroups can be shared across multiple grids. All uses must be under a common parent with issharedscope set to true. It is an attached property of grid as well. Grid is usually the best choice. Except wraping, it can do what most panels do.

Content overflow can be dealt with Clipping, Scrolling, Scaling, Wrapping and Trimming. Wrapping already seen. Only way for non-text to wrap is using wrappanel. Trimming is intelligent form of clipping. Supported for text by textblock and accesstext. They have texttrimming=None (default), or characterellipsis, orwordellipsis.

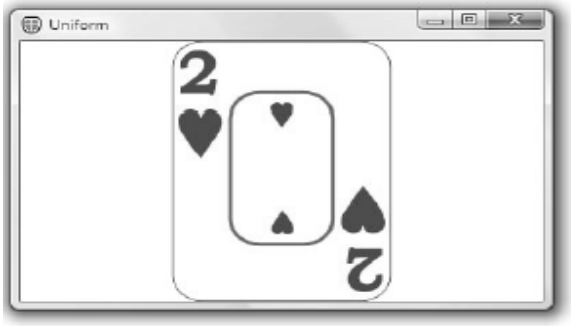
Clipping is default behavior. Edges of panel or cell area of dock area. All uielements have cliptobounds. Controls if it can draw outside bounds. Still cannot draw outside the window or page. Most panels clip regardless of this property. Canvas and uniformgrid dont unless this property is set. Button has this property false by default. Place canvas inside a grid cell to avert clipping. Clipping is done before rendertransform. Cant shrink back.



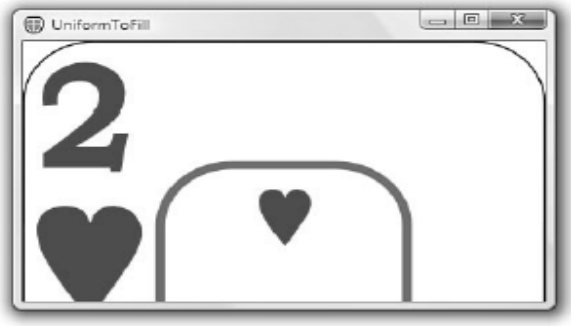
Stretch="None"



Stretch="Fill"



Stretch="Uniform"



Stretch="UniformToFill"

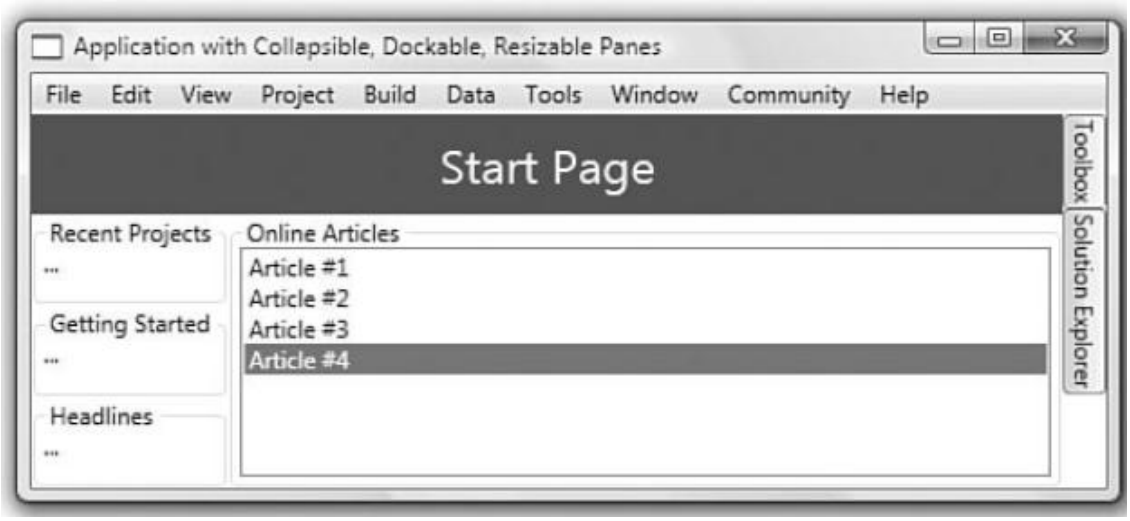
System.Windows.Controls.scrollviewer control can be used for scrolling. Its content property is set and verticalscrollbarvisibility and horizontalscrollbarvisibility properties determine the display.

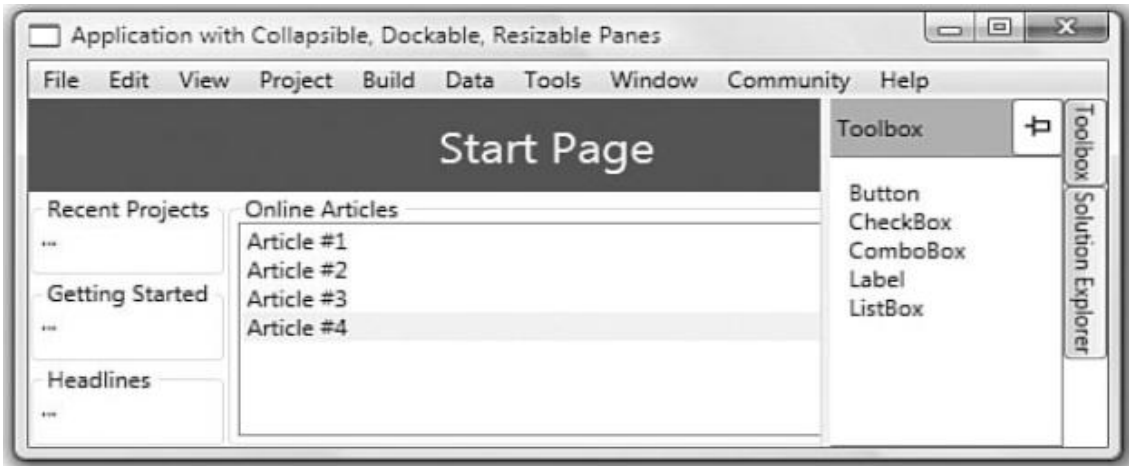
```
<Window Title="Using scrollviewer"
Xmlns="http://schemas.microsoft.com/winfx/2006/xaml/presentation">
    <scrollviewer>
        <stackpanel>
            …
        </stackpanel>
    </scrollviewer>
</Window>
```

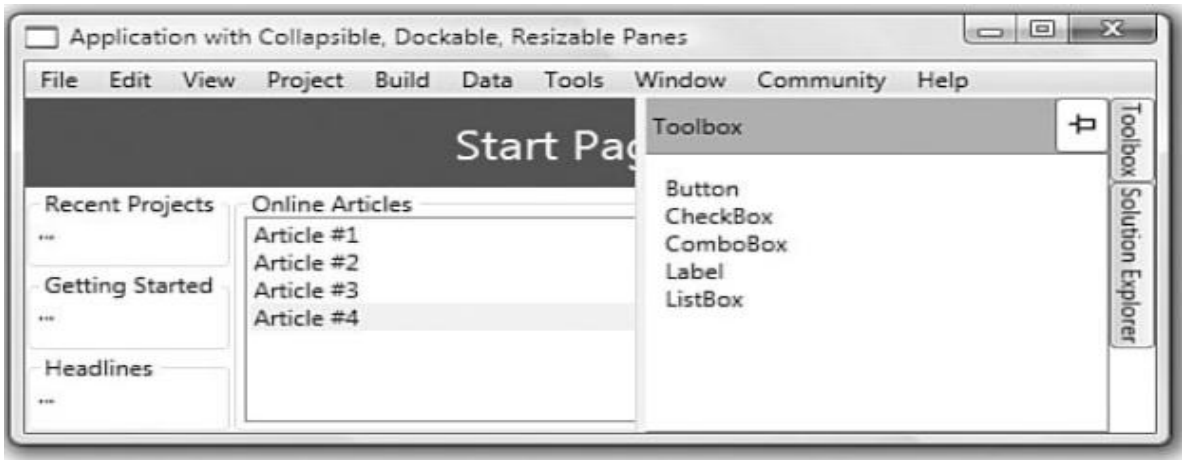
Scrolling more popular but e.g. Card game needs scaling. Scaletransform works relative to their own size not the available size. System.Windows.Controls.Viewbox, a type of class called Decorator (also Border). A panel-like thing but has only one child. Stretches to fill available space by default but also Stretch=None (like not using it at all), Fill, Uniform (aspect ratio, default), uniformtofill (cropped) strechdirection=uponly, downonly, Both (default).

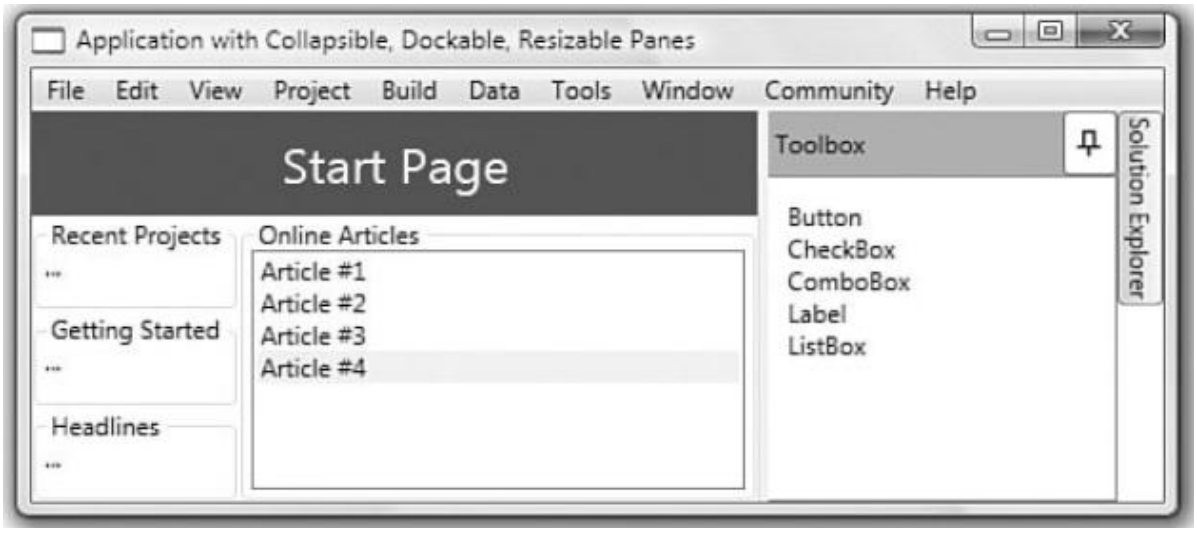
```
<Window Title="Using Viewbox"
Xmlns="http://schemas.microsoft.com/winfx/2006/xaml/presentation">
    <Viewbox>
        <stackpanel>
            …
        </stackpanel>
    </Viewbox>
</Window>
```

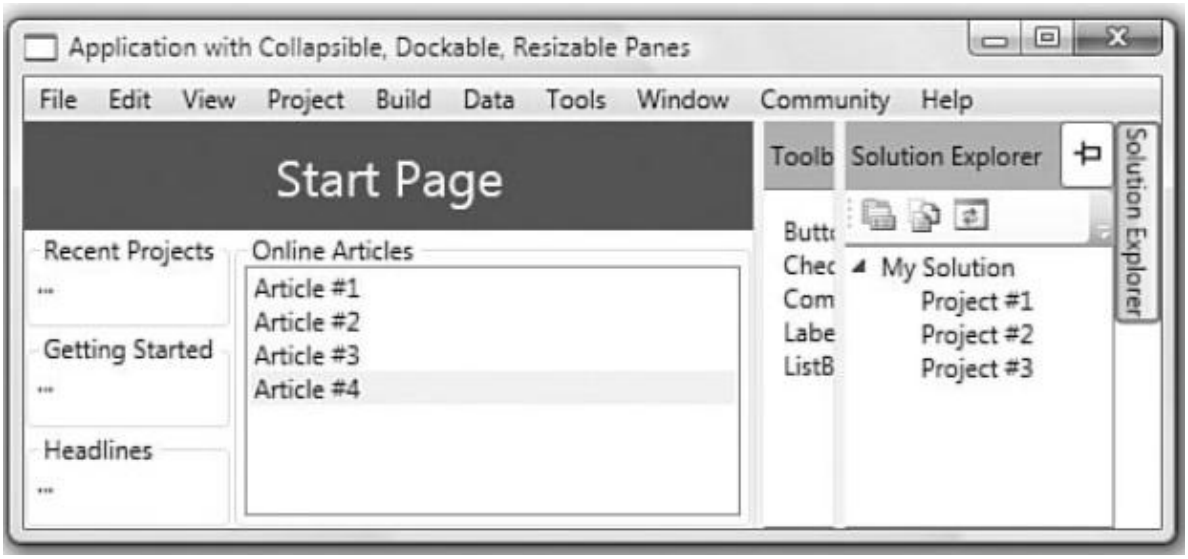
Let's make a visual studio like interface.

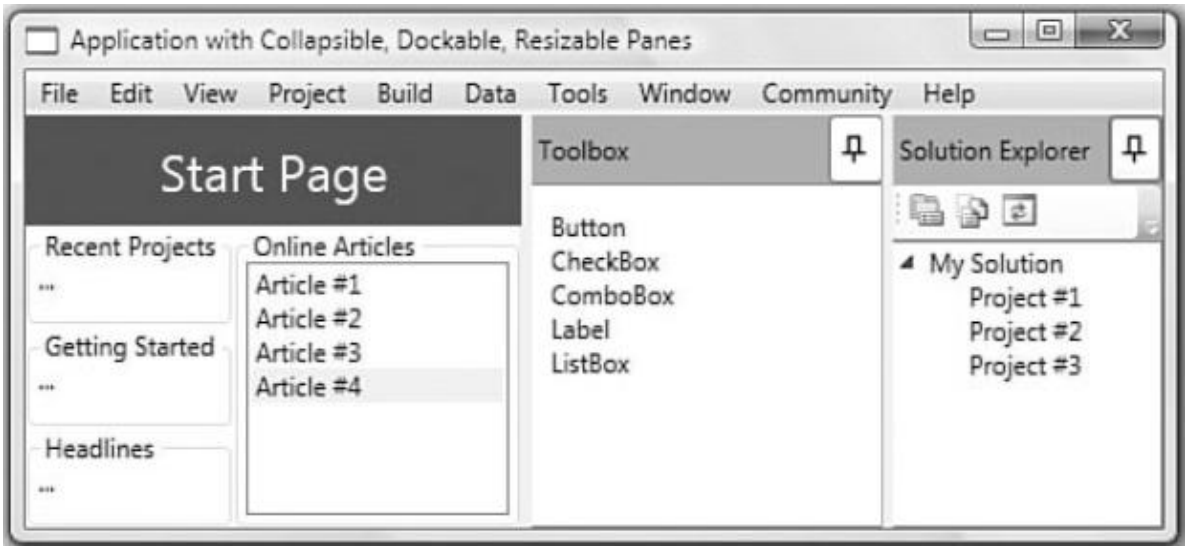






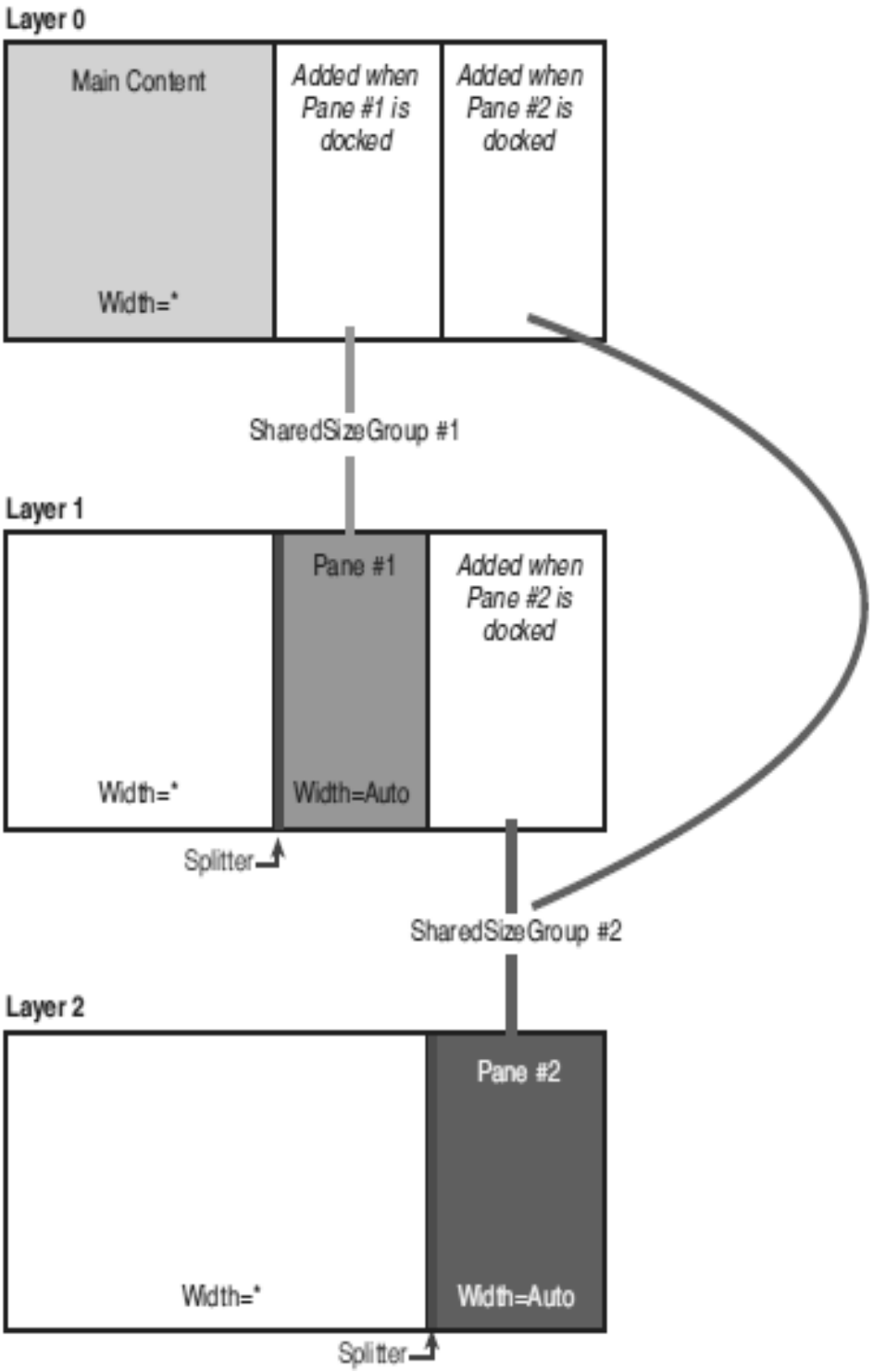






Chapter 21

Lecture 21



Because splitters are needed Grid is a reasonable idea. Three independent grids, because overlapping. Sharedsizegroup used to keep them in sync when docked. When docking cells are added / removed. Z order between layer 1 and 2 so undocked is on top. All 3 placed in another grid of single row col.

```
<Window x:Class="mainwindow"
Xmlns="http://schemas.microsoft.com/winfx/2006/xaml/presentation"
Xmlns:x="http://schemas.microsoft.com/winfx/2006/xaml"
Title="Application with Collapsible, Dockable, Resizable Panes">
    <dockpanel>
        <Menu dockpanel.Dock="Top">
            My Interface
        </Menu>


        <!-- The bar of buttons docked along the right edge: -->
        <stackpanel Name="buttonbar" Orientation="Horizontal" dockpanel.Dock="Right">
            <stackpanel.layouttransform>
                <rotatetransform Angle="90"/>
            </stackpanel.layouttransform>
            <Button Name="pane1button" mouseenter="pane1button_mouseenter">
                Toolbox
            </Button>
            <Button Name="pane2button" mouseenter="pane2button_mouseenter">
                Solution Explorer
            </Button>
        </stackpanel>
        <!-- The Grid containing the three child Grids fills the dockpanel: -->
        <Grid Name="parentgrid" Grid.issharedsizescope="True">
            <Grid.columndefinitions>
                <columndefinition Width="5*"/>
                <columndefinition Width="364*"/>
            </Grid.columndefinitions>
            <!-- Layer 0: -->
            <Grid Name="layer0" mouseenter="layer0_mouseenter" Grid.columnspan="2">
                <!-- … (content of this Grid is similar to Listing 5.2) -->
            </Grid>
            <!-- Layer 1: -->
            <Grid Name="layer1" Visibility="Collapsed" Grid.columnspan="2">
                <Grid.columndefinitions>
                    <columndefinition/>
                    <columndefinition sharedsizegroup="column1" Width="auto"/>
                </Grid.columndefinitions>
                <!-- Column 0 is empty, but column 1 contains a Grid and a gridsplitter:
-->
                <Grid Grid.Column="1" mouseenter="pane1_mouseenter"
                      Background="{dynamicresource {x:Static
systemcolors.activecaptionbrushkey}}">
                    <Grid.rowdefinitions>
                        <rowdefinition Height="auto"/>
                        <rowdefinition/>
                    </Grid.rowdefinitions>
                    <!-- Row 0 contains a header, and row 1 contains pane-specific
content: -->
                    <dockpanel Grid.Row="0">
                        <Button Name="pane1pin" Width="26" dockpanel.Dock="Right"
                                Click="pane1pin_Click" Background="White">
                            <Image Name="pane1pinimage" Source="pinhorizontal.gif"/>
```

```
                        </Button>
                        <textblock Padding="8" texttrimming="characterellipsis"
                                   Foreground="{dynamicresource {x:Static
systemcolors.activecaptiontextbrushkey}}"
                                   Dockpanel.Dock="Left">Toolbox
                        </textblock>
                    </dockpanel>
                    <!-- … (pane-specific content fills row 1) -->
                </Grid>
                <gridsplitter Width="5" Grid.Column="1" horizontalalignment="Left"/>
            </Grid>
            <!-- Layer 2: -->
            <Grid Name="layer2" Visibility="Collapsed" Grid.columnspan="2">
                <Grid.columndefinitions>
                    <columndefinition/>
                    <columndefinition sharedsizegroup="column2" Width="auto"/>
                </Grid.columndefinitions>
                <!-- Column 0 is empty, but column 1 contains a Grid and a gridsplitter:
-->
                <Grid Grid.Column="1" mouseenter="pane2_mouseenter"
                      Background="{dynamicresource {x:Static
systemcolors.activecaptionbrushkey}}">
                    <Grid.rowdefinitions>
                        <rowdefinition Height="auto"/>
                        <rowdefinition Height="auto"/>
                        <rowdefinition/>
                    </Grid.rowdefinitions>
                    <!-- Row 0 contains a header, and rows 1 & 2 contain pane-specific
content: -->
                    <dockpanel Grid.Row="0">
                        <Button Name="pane2pin" Width="26" dockpanel.Dock="Right"
                                Click="pane2pin_Click" Background="White">
                            <Image Name="pane2pinimage" Source="pinhorizontal.gif"/>
                        </Button>
                        <textblock Padding="8" texttrimming="characterellipsis"
                                   Foreground="{dynamicresource {x:Static
systemcolors.activecaptiontextbrushkey}}"
                                   Dockpanel.Dock="Left">Solution Explorer
                        </textblock>
                    </dockpanel>
                    <!-- … (pane-specific content fills rows 1 & 2)-->
                </Grid>
                <gridsplitter Width="5" Grid.Column="1" horizontalalignment="Left"/>
            </Grid>
        </Grid>
    </dockpanel>
</Window>


Using System;
Using System.Windows;
Using System.Windows.Controls;
Using System.Windows.Media.Imaging;

Public partial class mainwindow : Window
{
    // Dummy columns for layers 0 and 1:
```

```
Columndefinition column1cloneforlayer0;
Columndefinition column2cloneforlayer0;
Columndefinition column2cloneforlayer1;
Public mainwindow()
{
    Initializecomponent();
    // Initialize the dummy columns used when docking:
    Column1cloneforlayer0 = new columndefinition();
    Column1cloneforlayer0.sharedsizegroup = "column1";
    Column2cloneforlayer0 = new columndefinition();
    Column2cloneforlayer0.sharedsizegroup = "column2";
    Column2cloneforlayer1 = new columndefinition();
    Column2cloneforlayer1.sharedsizegroup = "column2";
}
// Toggle between docked and undocked states (Pane 1)
Public void pane1pin_Click(object sender, routedeventargs e)
{
    If (pane1button.Visibility == Visibility.Collapsed)
        Undockpane(1);
    Else
        Dockpane(1);
}
// Toggle between docked and undocked states (Pane 2)
Public void pane2pin_Click(object sender, routedeventargs e)
{
    If (pane2button.Visibility == Visibility.Collapsed)
        Undockpane(2);
    Else
        Dockpane(2);
}
// Show Pane 1 when hovering over its button
Public void pane1button_mouseenter(object sender, routedeventargs e)
{
    Layer1.Visibility = Visibility.Visible;
    // Adjust Z order to ensure the pane is on top:
    Grid.setzindex(layer1, 1);
    Grid.setzindex(layer2, 0);
    // Ensure the other pane is hidden if it is undocked
    If (pane2button.Visibility == Visibility.Visible)
        Layer2.Visibility = Visibility.Collapsed;
}
// Show Pane 2 when hovering over its button
Public void pane2button_mouseenter(object sender, routedeventargs e)
{
    Layer2.Visibility = Visibility.Visible;
    // Adjust Z order to ensure the pane is on top:
    Grid.setzindex(layer2, 1);
    Grid.setzindex(layer1, 0);
    // Ensure the other pane is hidden if it is undocked
    If (pane1button.Visibility == Visibility.Visible)
        Layer1.Visibility = Visibility.Collapsed;
}
// Hide any undocked panes when the mouse enters Layer 0
Public void layer0_mouseenter(object sender, routedeventargs e)
{
    If (pane1button.Visibility == Visibility.Visible)
        Layer1.Visibility = Visibility.Collapsed;
```

```
        If (pane2button.Visibility == Visibility.Visible)
            Layer2.Visibility = Visibility.Collapsed;
    }
    // Hide the other pane if undocked when the mouse enters Pane 1
    Public void pane1_mouseenter(object sender, routedeventargs e)
    {
        // Ensure the other pane is hidden if it is undocked
        If (pane2button.Visibility == Visibility.Visible)
            Layer2.Visibility = Visibility.Collapsed;
    }
    // Hide the other pane if undocked when the mouse enters Pane 2
    Public void pane2_mouseenter(object sender, routedeventargs e)
    {
        // Ensure the other pane is hidden if it is undocked
        If (pane1button.Visibility == Visibility.Visible)
            Layer1.Visibility = Visibility.Collapsed;
    }
    // Docks a pane, which hides the corresponding pane button
    Public void dockpane(int panenumber)
    {
        If (panenumber == 1)
        {
            Pane1button.Visibility = Visibility.Collapsed;
            Pane1pinimage.Source = new bitmapimage(new Uri("pin.gif", urikind.Relative));
            // Add the cloned column to layer 0:
            Layer0.columndefinitions.Add(column1cloneforlayer0);
            // Add the cloned column to layer 1, but only if pane 2 is docked:
            If (pane2button.Visibility == Visibility.Collapsed)
                Layer1.columndefinitions.Add(column2cloneforlayer1);
        }
        Else if (panenumber == 2)
        {
            Pane2button.Visibility = Visibility.Collapsed;
            Pane2pinimage.Source = new bitmapimage(new Uri("pin.gif", urikind.Relative));
            // Add the cloned column to layer 0:
            Layer0.columndefinitions.Add(column2cloneforlayer0);
            // Add the cloned column to layer 1, but only if pane 1 is docked:
            If (pane1button.Visibility == Visibility.Collapsed)
                Layer1.columndefinitions.Add(column2cloneforlayer1);
        }
    }
    // Undocks a pane, which reveals the corresponding pane button
    Public void undockpane(int panenumber)
    {
        If (panenumber == 1)
        {
            Layer1.Visibility = Visibility.Visible;
            Pane1button.Visibility = Visibility.Visible;
            Pane1pinimage.Source = new bitmapimage(new Uri("pinhorizontal.gif",
urikind.Relative));
            // Remove the cloned columns from layers 0 and 1:
            Layer0.columndefinitions.Remove(column1cloneforlayer0);
            // This won't always be present, but Remove silently ignores bad columns:
            Layer1.columndefinitions.Remove(column2cloneforlayer1);
        }
        Else if (panenumber == 2)
        {
```

```
            Layer2.Visibility = Visibility.Visible;
            Pane2button.Visibility = Visibility.Visible;
            Pane2pinimage.Source = new bitmapimage(new Uri("pinhorizontal.gif",
urikind.Relative));
            // Remove the cloned columns from layers 0 and 1:
            Layer0.columndefinitions.Remove(column2cloneforlayer0);
            // This won't always be present, but Remove silently ignores bad columns:
            Layer1.columndefinitions.Remove(column2cloneforlayer1);
        }
    }
}
```

Let's discuss Input Events now i.e. Keyboard, mouse, stylus, multi-touch. We will start with routed events and commands. Routed events are like dependency properties on top of .net properties. Similarly routed events are a layer on top of .net events. They can travel up or down a visual or logical tree. Helps apps remain oblivious to the visual tree. E.g. Button exposes Click based on mouseleftbuttondown and keydown but actually buttonchrome or textblock visual child fires it. Event travels up the tree and Button eventually sees it. Or like in a previous example rectangle on top of button. So arbitrary complex content but Click still raised. Otherwise would require custom code on inner content or consumers. Very similar to dependency property concept.

Its an implementation, not a language feature other than xaml, like dependency properties. A handful of API's. Public static routedevent fields, and by convention Event suffix. Registered in the static ctor and a .net event wrapper. (wrapper shouldn't do anything else).

```
Public class Button: buttonbase
{
// The routed event
Public static readonly  routedevent  clickevent;
Static Button()
{
// Register the event
Button.clickevent = eventmanager.registerroutedevent("Click",routingstrategy.Bubble,
Typeof(routedeventhandler), typeof(Button));
//
}
// A .NET event wrapper (optional)
Public event  routedeventhandler  Click
{
Add{ addhandler(Button.clickevent, value);}
Remove{ removehandler(Button.clickevent, value);}
}
Protected  override void  onmouseleftbuttondown(mousebuttoneventargs e)
{
//
// Raise the event
Raiseevent(new routedeventargs(Button.clickevent, this));
//
}
//
}
```

Addhandler, removehandler, raiseevent are from uielement not dependencyobject.

Routing Strategies include Tunneling: from root down to source or until marked handled, Bubbling: from source to root or until marked handled, and Direct: only on source just like .net events but still participate in routed event specific things like event triggers.

Handlers for routed events have a System.Object parameter (sender to which handler was attached), and System.eventargs, and source which is the logical tree element that originally raised the event and originalsource which is the element in visual tree that raised the event e.g. Textblock on top of a button. Routedevent, actual routed event object like Button.clickevent which is useful when same handler is used for multiple events.

There aer many keyboard, mouse, multi-touch, and stylus events. Most are bubbling events but many have a tunneling counterpart. By convention tunneling event names are prefixed with Preview and come just before the bubbling event comes. Previewmousemove comes before mousemove. By convention, actions are only taken on bubbling event so tunneling gives a chance to cancel or modify the event e.g. Textbox with restrictive input should use previewkeydown.

```
<Window xmlns="http://schemas.microsoft.com/winfx/2006/xaml/presentation"
Xmlns:x="http://schemas.microsoft.com/winfx/2006/xaml"
X:Class="aboutdialog" mouserightbuttondown="aboutdialog_mouserightbuttondown"
Title="About WPF 4 Unleashed" sizetocontent="widthandheight"
Background="orangered">
    <stackpanel>
        <Label fontweight="Bold" fontsize="20" Foreground="White">
            WPF 4 Unleashed
        </Label>
        <Label>© 2010 SAMS Publishing</Label>
        <Label>Installed Chapters:</Label>
        <listbox>
            <listboxitem>Chapter 1</listboxitem>
            <listboxitem>Chapter 2</listboxitem>
        </listbox>
        <stackpanel Orientation="Horizontal" horizontalalignment="Center">
            <Button minwidth="75" Margin="10">Help</Button>
            <Button minwidth="75" Margin="10">OK</Button>
        </stackpanel>
        <statusbar>You have successfully registered this product.</statusbar>
    </stackpanel>
</Window>
```

```
Using System.Windows;
Using System.Windows.Input;
Using System.Windows.Media;
Using System.Windows.Controls;

Public partial class aboutdialog : Window
{
    Public aboutdialog()
    {
        Initializecomponent();
    }
    Void aboutdialog_mouserightbuttondown(object sender, mousebuttoneventargs e)
    {
        // Display information about this event
        This.Title = "Source = " + e.Source.gettype().Name + ", originalsource = " +
            E.originalsource.gettype().Name + " @ " + e.Timestamp;
        // In this example, all possible sources derive from Control
        Control source = e.Source as Control;
        // Toggle the border on the source control
        If (source.borderthickness != new Thickness(5))
        {
            Source.borderthickness = new Thickness(5);
            Source.borderbrush = Brushes.Black;
        }
        Else
            Source.borderthickness = new Thickness(0);
    }
}
```

Chapter 22

Lecture 22

Let's go over the code from last lecture again.

```
Using System.Windows;
Using System.Windows.Input;
Using System.Windows.Media;
Using System.Windows.Controls;

Public partial class aboutdialog : Window
{
    Public aboutdialog()
    {
        Initializecomponent();
    }
    Void aboutdialog_mouserightbuttondown(object sender, mousebuttoneventargs e)
    {
        // Display information about this event
        This.Title = "Source = " + e.Source.gettype().Name + ", originalsource = " +
            E.originalsource.gettype().Name + " @ " + e.Timestamp;
        // In this example, all possible sources derive from Control
        Control source = e.Source as Control;
        // Toggle the border on the source control
        If (source.borderthickness != new Thickness(5))
        {
            Source.borderthickness = new Thickness(5);
            Source.borderbrush = Brushes.Black;
        }
        Else
            Source.borderthickness = new Thickness(0);
    }

   }
```

Note that we never receive an event on listbox item. Button has no border element so setting border has no effect. Handling a halted event can only be done from procedural code.

Let's discuss attached events. Attached events can bubble and tunnel through elements who have not defined the event. Like prop value inheritance and attach properties. Let's see a new example with window handling listbox selectionchanged and button click events but does not define them.

```
<Window xmlns="http://schemas.microsoft.com/winfx/2006/xaml/presentation"
Xmlns:x="http://schemas.microsoft.com/winfx/2006/xaml"
X:Class="aboutdialog" listbox.selectionchanged="listbox_selectionchanged"
Button.Click="Button_Click"
Title="About WPF Unleashed" sizetocontent="widthandheight"
Background="orangered">
    <stackpanel>
        <Label fontweight="Bold" fontsize="20" Foreground="White">
            WPF 4 Unleashed
        </Label>
        <Label>© 2010 SAMS Publishing</Label>
        <Label>Installed Chapters:</Label>
```

```
        <listbox>
            <listboxitem>Chapter 1</listboxitem>
            <listboxitem>Chapter 2</listboxitem>
        </listbox>
        <stackpanel Orientation="Horizontal" horizontalalignment="Center">
            <Button minwidth="75" Margin="10">Help</Button>
            <Button minwidth="75" Margin="10">OK</Button>
        </stackpanel>
        <statusbar>You have successfully registered this product.</statusbar>
    </stackpanel>
</Window>


Using System.Windows;
Using System.Windows.Controls;
Public partial class aboutdialog : Window
{
    Public aboutdialog()
    {
        Initializecomponent();
    }
    Void listbox_selectionchanged(object sender, selectionchangedeventargs e)
    {
        If (e.addeditems.Count > 0)
            Messagebox.Show("You just selected " + e.addeditems[0]);
    }
    Void Button_Click(object sender, routedeventargs e)
    {
        Messagebox.Show("You just clicked " + e.Source);
    }
}
```

Xaml valid because compiler sees relevant events defined in button and listbox at runtime however addhandler is directly called. Equivalent code is:

```
Public  aboutdialog()
{
Initializecomponent();
This.addhandler(listbox.selectionchangedevent,
New selectionchangedeventhandler(listbox_selectionchanged));
This.addhandler(Button.clickevent,new routedeventhandler(Button_Click));
}
```

Theoretically one handler can do all event handling. We can use delegate contravariance i.e. Arguments can be of base type than delegate.

```
Void generichandler(object sender, routedeventargs e)
{
If(e.routedevent == Button.clickevent)
{
Messagebox.Show("You just clicked "+ e.Source);
}
Else if (e.routedevent == listbox.selectionchangedevent)
{
Selectionchangedeventargs sce= (selectionchangedeventargs)e;
If(sce.addeditems.Count > 0)
Messagebox.Show("You just selected " + sce.addeditems[0]);
}
```

```
}
```

Let's discuss keyboard events keydown, keyup, and preview versions of them. Keyeventargs contains Key, imepro-cessedkey, deadcharprosessedkey, systemkey, isup, isdown, istoggled, keystates, isrepeat, and keyboarddevice.System.Windows.Input.Keyboard and its primarydevice property are accessible everywhere.

```
protected override void OnKeyDown(KeyEventArgs e)
{
if ((e.KeyboardDevice.Modifiers & ModifierKeys.Alt) == ModifierKeys.Alt && (e.Key ==
Key.A || e.SystemKey == Key.A))
{
// Alt+A has been pressed, potentially also with Ctrl, Shift, and/or Windows
}
base.OnKeyDown(e);
}

protected override void OnKeyDown(KeyEventArgs e)
{
if (e.KeyboardDevice.Modifiers == ModifierKeys.Alt && (e.Key == Key.A || e.SystemKey ==
Key.A))
{
// Alt+A and only Alt+A has been pressed
}
base.OnKeyDown(e);
}
```

You can use keyboarddevice.iskeydown to even check if left or right alt is down etc. What is keyboard focus. Uiele-ment Focusable property (true by default). Focusablechanged event, iskeyboardfocused, iskeyboardfocuswithin (readonly). To set focus, use Focus or movefocus. Iskeyboardfocusedchanged, iskeyboardfocuswithinchanged, gotkeyboardfocus, lostkeyboardfocus, previewgotkeyboardfocus, and previewlostkeyboardfocus are intersting events.

Mouse events are mouseenter and mouseleave (e.g. To create rollover effect however the preferred way is to use trigger with ismouseover property or ismousedirectlyover and ismousedirectlyoverchanged properties), mousemove and previewmousemove, mouseleftbuttondown, mouserightbuttondown, mouseleftbuttonup, mouserightbuttonup, and the more generic mousedown and mouseup, as well as the previewxxx versions of all six of these events, mousewheel and previewmousewheel. If Visibility=Collapsed no mouse events are generated but opacity=0 generates all events. Null background fill stroke produce areas with no events however "transparent" or color does. So null and transparent brush look same but different for hit test.

Mouseeventargs has 5 props of type mousebuttonstate (Pressed or Released) leftbutton, rightbutton, middlebutton, xbutton1, and xbutton2. Getposition function returning a Point with x and y props. Pass null for screen relative position or an element for position with respect to that element. Mousewheel and previewmousewheel get mousewheeleventargs derived from mouseeventargs (add a Delta property). 12 events in mouse up down get mousebuttoneventargs adds a changedbutton property, a buttonstate property (for the changed button) and a clickcount (clicks with time ¡ double click speed). Button base class raises a mousedoubleclick by checking clickcount in mouseleftbuttondown, similarly in preview versions.

For drag and drop, we have dragenter, dragover, dragleave, with previewdragenter, previewdragover, and previewdragleave, Drop and previewdrop, querycontinuedrag and previewquerycontinuedrag. Drag and drop of clipboard content to/from elements not elements themselves. Its enabled by alldrop=true. First two sets give drageventargs with getposition, Data, Effects and allowedeffects (Copy, Move, Link, Scroll, All, None), keystates (leftmousebutton, rightmousebutton, middlemousebutton, shiftkey, controlkey, altkey, or None) Continue evnets are raised when a keyboard or mouse state changes during the operation and querycontinuedragargs has keystates, escapepressed, Action (Continue, Drop, Cancel).

To understand mouse capturing imagine drag and drop of elements. How to implement with mouseleftbuttondown, mousemove, and mouseleftbuttonup. But mouse too fast or under another element. Uielements can capture mouse and release mousecapture. Properties is mouse captured and is mouse capture within, and the events got mouse capture, lost mouse capture, is mouse capture changed, and is mouse capture within changed are related to capturing. Inside mouse move you can use a layout or rendertransform to move.

Stylus can behave like a mouse but has higher resolution, inverted, in air or not, pressure sensitivity. Similarly multi-touch events are like mouse but not the other way around. There are available on win7 or later with multi-touch hardware. We can either use basic touch events or higher level manipulation events.

Basic touch events are touchenter and touchleave, touchmove and previewtouchmove, touchdown, touchup, previewtouchdown and previewtouchu, gottouchcapture and losttouchcapture. With multiple fingers, events raised for each finger separately. For first finger mouse events are generated as well. Toucheventargs has gettouchpoint, getintermediatetouchpoints, touchdevice. Touchpoint has Position, Size, Bounds, touchdevice, Action (Down, Up, Move). Each finger has its own touchdevice identified by Id porp.

Some notebooks support only two simultaneous touch points. Silverlight 4 doesnt support them. A lower level framereported event that isnt even routed is however raised. Now Let's discuss manipulation events for panning, rotating, and zooming. They are easy to apply with a transform. Swipe easy with the other events as well but rotate etc. Difficult with previous events and lack of consistency in their behaviour between applications. Therefore using the manipulation events is preferred.

Chapter 23

Lecture 23

Let's see an example using touch events.

```
<Window x:Class="touchevents.mainwindow"
Xmlns="http://schemas.microsoft.com/winfx/2006/xaml/presentation"
Xmlns:x="http://schemas.microsoft.com/winfx/2006/xaml"
Title="Touch Events">
    <Canvas Name="canvas">
        <Canvas.Background>
            <lineargradientbrush>
                <gradientstop Color="Black"/>
                <gradientstop Color="Red" Offset="1"/>
            </lineargradientbrush>
        </Canvas.Background>
    </Canvas>
</Window>

Using System;
Using System.Collections.Generic;
Using System.Windows;
Using System.Windows.Controls;
Using System.Windows.Input;
Using System.Windows.Media;
Using System.Windows.Media.Imaging;

Namespace touchevents
{
    Public partial class mainwindow : Window
    {
        // Keep track of which images are used for which touchdevices
        Dictionary<touchdevice, Image> fingerprints =
        New Dictionary<touchdevice, Image>();
        Public mainwindow()
        {
            Initializecomponent();
        }
        Protected override void ontouchdown(toucheventargs e)
        {
            Base.ontouchdown(e);
            // Capture this touch device
            Canvas.capturetouch(e.touchdevice);
            // Create a new image for this new touch
            Image fingerprint = new Image {
                Source = new bitmapimage(new
Uri("pack://application:,,,/fingerprint.png"))
            };
            // Move the image to the touch point
            Touchpoint point = e.gettouchpoint(canvas);
            Fingerprint.rendertransform = new translatetransform(point.Position.X,
point.Position.Y);
            // Keep track of the image and add it to the canvas
```

```
            Fingerprints[e.touchdevice] = fingerprint;
            Canvas.Children.Add(fingerprint);
        }
        Protected override void ontouchmove(toucheventargs e)
        {
            Base.ontouchmove(e);
            If (e.touchdevice.Captured == canvas)
            {
                // Retrieve the right image
                Image fingerprint = fingerprints[e.touchdevice];
                Translatetransform transform =
                Fingerprint.rendertransform as translatetransform;
                // Move it to the new location
                Touchpoint point = e.gettouchpoint(canvas);
                Transform.X = point.Position.X;
                Transform.Y = point.Position.Y;
            }
        }
        Protected override void ontouchup(toucheventargs e)
        {
            Base.ontouchup(e);
            // Release capture
            Canvas.releasetouchcapture(e.touchdevice);
            // Remove the image from the canvas and the dictionary
            Canvas.Children.Remove(fingerprints[e.touchdevice]);
            Fingerprints.Remove(e.touchdevice);
        }
    }
}
```

Manipulation events are manipulationstarting and manipulationstarted, manipulationdelta, manipulationcompleted. We combine information from multiple events. Manipulation events work if ismanipulationenabled=true on this or a parent and basic events not handled. Manipulation starting and started events at first finger touchdown, delta at touchmove, and completed after touchup for all fingers. Starting and started events allow to customize or cancel or disallow some manipulations. Manipulationdelta class has Translation, Scale, Rotation, Expansion (like Scale but dev ind px instead of scale factor). Manipulationdeltaeventargs has deltamanipulation and cumulativemanipulation. Let's see an example to move rotate zoom a photo.

```
<Window x:Class="manipulationevents.mainwindow"
Xmlns="http://schemas.microsoft.com/winfx/2006/xaml/presentation"
Xmlns:x="http://schemas.microsoft.com/winfx/2006/xaml"
Title="Manipulation Events">
    <Canvas Name="canvas" ismanipulationenabled="True">
        <Image Name="photo" Source="photo.jpg">
            <Image.rendertransform>
                <matrixtransform/>
            </Image.rendertransform>
        </Image>
    </Canvas>
</Window>

Using System;
Using System.Windows;
Using System.Windows.Input;
Using System.Windows.Media;
```

```
Namespace manipulationevents
{
    Public partial class mainwindow : Window
    {
        Public mainwindow()
        {
            Initializecomponent();
            Canvas.manipulationdelta += Canvas_manipulationdelta;
        }
        Void Canvas_manipulationdelta(object sender, manipulationdeltaeventargs e)
        {
            Matrixtransform transform = photo.rendertransform as matrixtransform;
            If (transform != null)
            {
                // Apply any deltas to the matrix,
                // then apply the new Matrix as the matrixtransform data:
                Matrix matrix = transform.Matrix;
                Matrix.Translate(e.deltamanipulation.Translation.X,
e.deltamanipulation.Translation.Y);
                Matrix.rotateat(e.deltamanipulation.Rotation, e.manipulationorigin.X,
e.manipulationorigin.Y);
                Matrix.scaleat(e.deltamanipulation.Scale.X, e.deltamanipulation.Scale.Y,
e.manipulationorigin.X, e.manipulationorigin.Y);
                Transform.Matrix = matrix;
                E.Handled = true;
            }
        }
    }
}
```

Manipulations always done relative to a manipulation container. By default element with is manipulation enabled=true or handle manipulation starting event and set manipulation starting event args.manipulation container. Now what is Inertia. Manipulation inertia starting event when all fingers loose contact even before a completed event. By default manipulationcompleted thrown right away but you can set manipulation inertia starting event args.translation behavior, rotation behavior, and/or expansion behavior which will cause deltas thrown and then completed until inertia is completed. Translation behavior has desired displacement, desireddeceleration, and initialvelocity. Rotation behavior has desired rotation, desired deceleration, and initial velocity.

Expansionbehavior has desiredexpansion, desireddeceleration, initialradius, and initialvelocity. Typically only set desireddeceleration or the behavior-specific desireddisplacement, desiredrotation, or desiredexpansion. The latter properties are useful for ensuring that the element doesnt go too far. By default, initialvelocity and initialradius are initialized with the current values. You can get the various velocities by checking manipulationinertiastartingeventargs.initialvelocities, which has linearvelocity, angularvelocity, and expansionvelocity.

```
Using System;
Using System.Windows;
Using System.Windows.Input;
Using System.Windows.Media;
Namespace manipulationevents
{
    Public partial class mainwindow : Window
    {
        Public mainwindow()
        {
            Initializecomponent();
```

```
            Canvas.manipulationdelta += Canvas_manipulationdelta;
            Canvas.manipulationinertiastarting += Canvas_manipulationinertiastarting;
        }
        Void Canvas_manipulationinertiastarting(object sender,
manipulationinertiastartingeventargs e)
        {
            E.translationbehavior.desireddeceleration = 0.01;
            E.rotationbehavior.desireddeceleration = 0.01;
            E.expansionbehavior.desireddeceleration = 0.01;
        }
        Void Canvas_manipulationdelta(object sender, manipulationdeltaeventargs e)
        {
            Matrixtransform transform = photo.rendertransform as matrixtransform;
            If (transform != null)
            {
                // Apply any deltas to the matrix,
                // then apply the new Matrix as the matrixtransform data:
                Matrix matrix = transform.Matrix;
                Matrix.Translate(e.deltamanipulation.Translation.X,
e.deltamanipulation.Translation.Y);
                Matrix.rotateat(e.deltamanipulation.Rotation, e.manipulationorigin.X,
e.manipulationorigin.Y);
                Matrix.scaleat(e.deltamanipulation.Scale.X, e.deltamanipulation.Scale.Y,
e.manipulationorigin.X, e.manipulationorigin.Y);
                Transform.Matrix = matrix;
                E.Handled = true;
            }
        }
    }
}
```

To be boundary aware manipulationboundaryfeedback event is used. Inside a manipulationdelta event handler, you can call the reportboundaryfeedback method on the passed-in manipulationdeltaeventargs instance to make the window bounce similar to iphone bounce list behavior. Let's see a spin prize wheel example.

```
<Window x:Class="spintheprizewheel.mainwindow"
Xmlns="http://schemas.microsoft.com/winfx/2006/xaml/presentation"
Xmlns:x="http://schemas.microsoft.com/winfx/2006/xaml"
Title="Spin the Prize Wheel">
    <Window.Background>
        <lineargradientbrush>
            <gradientstop Color="White"/>
            <gradientstop Color="Orange" Offset="1"/>
        </lineargradientbrush>
    </Window.Background>
    <Grid Name="grid" ismanipulationenabled="True">
        <Image Name="prizewheel" rendertransformorigin="0.5,0.5"
               Source="prizewheel.png" Margin="0 30 0 0">
            <Image.rendertransform>
                <rotatetransform/>
            </Image.rendertransform>
        </Image>
        <Image Source="arrow.png" verticalalignment="Top" Stretch="None"/>
    </Grid>
</Window>
```


```
Using System;
Using System.Windows;
Using System.Windows.Input;
Using System.Windows.Media;
Namespace spintheprizewheel
{
    Public partial class mainwindow : Window
    {
        Public mainwindow()
        {
            Initializecomponent();
            Grid.manipulationstarting += Grid_manipulationstarting;
            Grid.manipulationdelta += Grid_manipulationdelta;
            Grid.manipulationinertiastarting += Grid_manipulationinertiastarting;
            Grid.manipulationcompleted += Grid_manipulationcompleted;
        }
        Void Grid_manipulationstarting(object sender, manipulationstartingeventargs e)
        {
            E.Mode = manipulationmodes.Rotate; // Only allow rotation
        }
        Void Grid_manipulationdelta(object sender, manipulationdeltaeventargs e)
        {
            (prizewheel.rendertransform as rotatetransform).Angle +=
            E.deltamanipulation.Rotation;
        }
        Void Grid_manipulationinertiastarting(object sender,
manipulationinertiastartingeventargs e)
        {
            E.rotationbehavior.desireddeceleration = 0.001;
        }
        Void Grid_manipulationcompleted(object sender, manipulationcompletedeventargs e)
        {
            // Now that the wheel has stopped, tell the user what s/he won!
        }
    }
}
```

You can take advantage of panning support built into scrollviewer by setting its panningmode property to horizontalonly, verticalonly, horizontalfirst, verticalfirst, or Both. You can download the Surface Toolkit for Windows Touch to get numerous slick Microsoft Surface WPF controls that are optimized for multi-touch. This includes "surface versions" of most common controls (such as surfacebutton and surfacecheckbox) and brand-new controls (such as scatterview and librarystack).

Chapter 24

Lecture 24

Commands are a more abstract and loosely coupled version of events e.g. Cut copy paste commands. They are exposed in various ways. They can be enabled disabled e.g. If there is nothing to paste. Two-way communication gets cumbersome especially if you dont want the list of controls hard-coded.

WPF defines a number of built-in commands. Commands have automatic support for input gestures (such as keyboard shortcuts). Some of wpfs controls have built-in behavior tied to various commands. Any object implementing icommand which defines Execute, canexecute, canexecutechanged can work as a Command. For Cut, Copy, and Paste, you could define and implement three classes implementing icommand, find a place to store them (perhaps as static fields of the main Window), call Execute from relevant event handlers (when canexecute returns true), and handle the canexecutechanged event to toggle the isenabled property on the relevant pieces of user interface. Controls have logic to interface with commands through Command property. We can also do it all in xaml.

Following are pre-defined builtin commands. Applicationcommands e..g Close, Copy, Cut, Delete, Find, Help, New, Open, Paste, Print, printpreview, Properties, Redo, Replace, Save, saveas, selectall, Stop, Undo, and more. Componentcommands e.g. Movedown, moveleft, moveright, moveup, scrollbyline, scrollpagedown, scrollpageleft, scrollpageright, scrollpageup, selecttoend, selecttohome, selecttopagedown, selecttopageup, and more. Mediacommands e.g. Channeldown, channelup, decreasevolume, fastforward, increasevolume, mutevolume, nexttrack, Pause, Play, previoustrack, Record, Rewind, Select, Stop, and more. Navigationcommands e.g. Browseback, browseforward, browsehome, browsestop, Favorites, firstpage, gotopage, lastpage, nextpage, previouspage, Refresh, Search, Zoom, and more. Editingcommands e.g. Aligncenter, alignjustify, alignleft, alignright, correctspellingerror, decreasefontsize, decreaseindentation, enterlinebreak, enterparagraphbreak, ignorespellingerror, increasefontsize, increaseindentation, movedownbyline, movedownbypage, movedownbyparagraph, moveleftbycharacter, moveleftbyword, moverightbycharacter, moverightbyword, and more.

All instances of routeduicommand implement icommand and support bubbling.

Helpbutton.Command= applicationcommands.Help;

If you run it would be always disabled. We need to add commandbinding to the element or a Parent (bubbling). All uielements have commandbindings collection.

```
This.commandbindings.Add(new commandbinding(applicationcommands.Help, helpexecuted,
helpcanexecuted));
// Navigate to a page instance
Photopage  nextpage= new photopage();
This.navigationservice.Navigate(nextpage);
// Or navigate to a page via a URI
This.navigationservice.Navigate(new Uri("photopage.xaml", urikind.Relative));
This.commandbindings.Add( new commandbinding(applicationcommands.Help, helpexecuted,
helpcanexecuted));

<Window xmlns="http://schemas.microsoft.com/winfx/2006/xaml/presentation"
Xmlns:x="http://schemas.microsoft.com/winfx/2006/xaml"
X:Class="aboutdialog"
Title="About WPF Unleashed" sizetocontent="widthandheight"
Background="orangered">
    <Window.commandbindings>
```

```
        <commandbinding Command="Help"
Canexecute="helpcanexecute" Executed="helpexecuted"/>
    </Window.commandbindings>
    <stackpanel>
        <Label fontweight="Bold" fontsize="20" Foreground="White">
            WPF 4 Unleashed
        </Label>
        <Label>© 2010 SAMS Publishing</Label>
        <Label>Installed Chapters:</Label>
        <listbox>
            <listboxitem>Chapter 1</listboxitem>
            <listboxitem>Chapter 2</listboxitem>
        </listbox>
        <stackpanel Orientation="Horizontal" horizontalalignment="Center">
            <Button minwidth="75" Margin="10" Command="Help"
                    Content= "{
                        Binding relativesource={relativesource Self}, Path=Command.Text
                    }"/>
            <Button minwidth="75" Margin="10">OK</Button>
        </stackpanel>
        <statusbar>You have successfully registered this product.</statusbar>
    </stackpanel>
</Window>

Using System.Windows;
Using System.Windows.Input;
Public partial class aboutdialog : Window
{
    Public aboutdialog()
    {
        Initializecomponent();
    }
    Void helpcanexecute(object sender, canexecuteroutedeventargs e)
    {
        E.canexecute = true;
    }
    Void helpexecuted(object sender, executedroutedeventargs e)
    {
        System.Diagnostics.Process.Start("http://www.adamnathan.net/wpf");
    }
}
```

There is a commandconverter type converter. X:Static applicationcommands.Help.

Custom commands dont get this treatment. Now even F1 works, its an input gesture. You can also add keybindings and mousebindings yourself.

```
This.inputbindings.Add(
New keybinding(applicationcommands.Help, new keygesture(Key.F2)));

<Window.inputbindings>
<keybinding  Command="Help" Key="F2"/>
<keybinding  Command="notacommand" Key="F1"/>
</Window.inputbindings >

This.inputbindings.Add(
New keybinding(applicationcommands.notacommand, new keygesture(Key.F1)));
```

Controls with builtin command bindings e.g. Textbox that responds to ctrl-z etc.

```
<stackpanel xmlns="http://schemas.microsoft.com/winfx/2006/xaml/presentation"
Xmlns:x="http://schemas.microsoft.com/winfx/2006/xaml"
Orientation="Horizontal" Height="25">
<Button Command="Cut" commandtarget="{Binding  elementname=textbox}"
Content="{Binding  relativesource={relativesource Self}, Path=Command.Text}"/>
<Button Command="Copy" commandtarget="{Binding  elementname=textbox}"
Content="{Binding  relativesource={relativesource Self}, Path=Command.Text}"/>
<Button Command="Paste" commandtarget="{Binding  elementname=textbox}"
Content="{Binding  relativesource={relativesource Self}, Path=Command.Text}"/>
<Button Command="Undo" commandtarget="{Binding  elementname=textbox}"
Content="{Binding  relativesource={relativesource Self}, Path=Command.Text}"/>
<Button Command="Redo" commandtarget="{Binding  elementname=textbox}"
```


```
Content="{Binding  relativesource={relativesource Self}, Path=Command.Text}"/>
<textbox x:Name="textbox" Width="200"/>
</stackpanel >
```

Button and textbox have no direct knowledge of each other. Through commands we have a rich interaction. The more the standardization on builtin commands, the more seamless and declarative the interaction can be between controls.

In summary, wpf input events make possible rich interactive content. We focused on uielement but the same events can be used with contentelement.

Chapter 25

Lecture 25

Let's discuss structuring and deploying an application. A standard windows app or partial trust web app or loose xaml. We will discuss Photo Gallery example with the book. App.xaml and mainwindow.xaml and code-behind files on a new wpf project. WPF Window is a win32 window, same chrome (non-client area) same taskbar behavior etc. Icon Title windowstyle topmost showintaskbar properties. Left and Top props or windowstartuplocation=centerscreen or centerowner. Any number of child windows can be made by instantiating a Window dervied class and calling Show. Child window like parent window but gets closed when parent and similarly minimized, also called modeless dialog.

Another windows Owner property after parent shown, ownedwindows property. There are Activated & Deactivated events. Activate method (like setforegroundwindow). Showactivated=false, initially not shown.

```
Public partial class mainwindow : Window
{
Public  mainwindow()
{
Initializecomponent();
}
Protected  override void onclosing(canceleventargs e)
{
Base.onclosing(e);
If(messagebox.Show("Are you sure you want to close Photo Gallery?",
"Annoying Prompt", messageboxbutton.yesno, messageboximage.Question)
== messageboxresult.No)
E.Cancel = true;
}
Protected  override void onclosed(eventargs e)
{
Base.onclosed(e);
// Persist the list of favorites
//
}
Protected  override void oninitialized(eventargs e)
{
Base.oninitialized(e);
// Retrieve the persisted list of favorites
//
}
Void exitmenu_Click(object sender, routedeventargs e)
{
This.Close();
}
}
Public static void Main()
{
Mainwindow  window = new mainwindow();
Window.Show();
}
```

End result same whether you attach event handler or override this method. Technically a bit faster and more

neat when subclass wants to handle something base class throws. Dispatcherobject cant be accessed from a different thread which avoids many issues. App terminating, you need to add a message loop to process windows messages and pass them to the correct window.

```
[stathread]
Public static void Main()
{
Application app = new Application();
Mainwindow  window = new mainwindow();
Window.Show();
App.Run(window);
}
[stathread]
Public static void Main()
{
Application app = new Application();
App.startupuri = new Uri("mainwindow.xaml", urikind.Relative);
App.Run();
}


<Application x:Class="photogallery.App"
Xmlns="http://schemas.microsoft.com/winfx/2006/xaml/presentation"
Xmlns:x="http://schemas.microsoft.com/winfx/2006/xaml"
Startupuri="mainwindow.xaml"/>
Using System.Windows;
Namespace  photogallery
{
Public partial class App : Application
{
Public App()
{
Initializecomponent();
}
}
}
Main()
[System.stathreadattribute()]
Public static void Main()
{
Photogallery.App app = new photogallery.App();
App.initializecomponent();
App.Run();
}
```

App code-behind can be omitted altogether. But where is main. App.xaml is assigned applicationdefinition which causes App.g.cs to be generated and can be seen in Solution Explorer if you Show all files. System.Environment.getcommandlinea or set build action to Page and write Main. Startup, Exit,Activated, Deactivated (any window), sessionending (can-cellable logoff shutdown). A read-only Windows collection, mainwindow property (read-write). Shutdownmode (when to end the app). Properties dictionary for sharing between windows. Application.Current from any window. Let's discuss creating a single instance app. Can create app without Application using Dispatcher.Run or Win32 message loop but explicit termination and other issues. One UI Thread and one render thread. Can create more with Dispatcher.Run which can improve responsiveness. Can schedule on UI thread with Dispatcher.Invoke and begininvoke and priority from send (now) to systemidle.

```
Bool mutexisnew;
Using (System.Threading.Mutex m =
New System.Threading.Mutex(true, uniquename, out mutexisnew))
{
If (mutexisnew)
// This is the first instance. Run the application.
Else
// There is already an instance  running. Exit!
}
```

You can also create a splash screen. Nothing fancy because wpf not loaded yet. In wpf project - add new item -splash screen, adds an images with build action splashscreen.

Modal dialogs include common dialogs which are actually provided by win32. Instantiate, call showdialog and process result.

```
Void printmenu_Click(object sender, routedeventargs e)
{
String filename = (picturebox.selecteditem as listboxitem).Tag as string;
Image image = new Image();
Image.Source = new bitmapimage(new Uri(filename, urikind.relativeorabsolute));
Printdialog pd = new printdialog();
If (pd.showdialog()== true) // Result could be true, false, or null
Pd.printvisual(image, Path.getfilename(filename)+ " from Photo Gallery");
}


Void renamemenu_Click(object sender, routedeventargs e)
{
String filename = (picturebox.selecteditem as listboxitem).Tag as string;
Renamedialog  dialog = new renamedialog(Path.getfilenamewithoutextension(filename));
If (dialog.showdialog() == true) // Result could be true, false, or null
{
// Attempt to rename the file
Try
{
File.Move(filename, Path.Combine(Path.getdirectoryname(filename),
Dialog.newfilename)+ Path.getextension(filename));
}
Catch (Exception ex)
{
Messagebox.Show(ex.Message,   Cannot  Rename  File , messageboxbutton.OK,
Messageboximage.Error);
}
}
}
```

A window to be shown as a dialog sets its dialogresult to bool. Setting it closes window or set Button's isdefault prop to true.

```
Void okbutton_Click(object sender, routedeventargs e)
{
This.dialogresult = true;
}
```

```
[stathread]
Public static void Main()
{
Mainwindow  window = new mainwindow();
Window.showdialog();
}
```

Let's discuss persisting and restoring. Can use registry or file system but Let's use .net isolated storage. Physically located in the users document folder in a hidden folder. VS generated Settings class provide an even easier and strongly typed way but in a app. Config file.

```
Protected  override void onclosed(eventargs e)
{
Base.onclosed(e);
// Write each favorites item when the application is about to close
Isolatedstoragefile f = isolatedstoragefile.getuserstoreforassembly();
Using (isolatedstoragefilestream  stream = new isolatedstoragefilestream("myfile",
filemode.Create, f));
Using (streamwriter  writer = new streamwriter(stream));
{
Foreach (treeviewitem item in favoritesitem.Items)
Writer.writeline(item.Tag as string);
}
}
Protected  override void oninitialized(eventargs e)
{
Base.oninitialized(e);
// Read each favorites item when the application is initialized
Isolatedstoragefile f = isolatedstoragefile.getuserstoreforassembly();
Using (isolatedstoragefilestream  stream =
New isolatedstoragefilestream("myfile", filemode.openorcreate, f))
Using (streamreader  reader = new streamreader(stream))
{
String line = reader.readline();
While (line != null)
{
Addfavorite(line);
Line = reader.readline();
}
}
//
}
```

Let's discuss clickonce vs. Windows installer. VS setup and deployment projects or a simpler clickonce wizard from build-¿publish menu. Windows installer benefits include showing custom setup UI like a EULA, control over where files installed, arbitrary code at setup time, install shared assemblies in global assembly cache, register COM components and file associations, install for all users, o✏ine installation from cd. Clickonce benefits include builtin support for automatic updates and rollback to prev versions, a web-like "go-away" experience or start menu and control panel program list, all files in isolated area, so no effect on other apps, clean uninstallation, but full trust apps can do things while they run, .net code access security, partial trust.

Chapter 26

Lecture 26

Let's discuss navigation based apps like windows explorer, media player, photo gallery. Can use navigation support for a wizard or organize the whole ui around navigation. With nav, content is usually in "Page" a simpler version of Window. Then hosted in a navigationwindow or a Frame. They provide support for navigating, a history journal, and nav related events. Navigationwindow more like a top-level window whereas Frame more like an HTML frame or iframe By default navwind has a bar on top with back/fwd, frame doesnt but can be shown.

```
<navigationwindow
Xmlns="http://schemas.microsoft.com/winfx/2006/xaml/presentation"
Xmlns:x="http://schemas.microsoft.com/winfx/2006/xaml"
Xmlns:d="http://schemas.microsoft.com/expression/blend/2008"
Xmlns:mc="http://schemas.openxmlformats.org/markup-compatibility/2006"
Mc:Ignorable="d" x:Class="photogallery.Container"
Title="Photo Gallery" Source="mainpage.xaml"
D:designwidth="1320" d:designheight="919"
/>
```

Nav enabled version points startup uri to this and and mainpage.xaml referenced above has the content.

```
<Page x:Class="photogallery.mainpage"
Xmlns="http://schemas.microsoft.com/winfx/2006/xaml/presentation"
Xmlns:x="http://schemas.microsoft.com/winfx/2006/xaml"
Title="Photo Gallery" Loaded="Page_Loaded">
<!--   A p p l i c a t i o n -specific  c o n t e n t -->
</Page >
```

Page does everything window does except onclosed and onclosing. With these changes, nothing much changes in our app, just some disabled back/fwd buttons. Navigation can also happen between html files. A Page can interact with its navigation container by using the navigationservice class, which exposes relevant functionality regardless of whether the container is a navigationwindow or a Frame. You can get an instance of navigationservice by calling the static navigationservice.getnavigationservice method and passing the instance of the Page. But even more easily, you can simply use Pages navigationservice property. For example, you can set a title that is used in the drop-down menu associated with the Back and Forward buttons as follows:

```
This.navigationservice.Title = Main  Photo Gallery  Page;
```

Or you can refresh the current Page as follows:

```
This.navigationservice.Refresh();
```

But Page also contains a few of its own properties that control the behavior of the parent container, such as windowheight, windowwidth, and windowtitle. Can also set in xaml. You can perform navigation in three main ways: Calling the Navigate method, Using Hyperlinks, Using the journal. Navigation containers support a Navigate method.

```
// Navigate to a page instance
Photopage  nextpage= new photopage();
This.navigationservice.Navigate(nextpage);
// Or navigate to a page via a URI
This.navigationservice.Navigate(new Uri("photopage.xaml", urikind.Relative));
```

Root of xaml must be “Page”. Or for html

```
This.navigationservice.Navigate(new Uri("http://www.adamnathan.net/wpf"));
```

Can also use 2 properties. Only useful from xaml.

```
This.navigationservice.Content = nextpage;
This.navigationservice.Source= new Uri("photopage.xaml", urikind.Relative);
```

Hyperlink element is used to link to xaml. Or handle its Click event and call navigate yourself to link from html towpf. Handle Navigating event and look for a sentinel HREF value can also set targetname to update a frame or use# and any named element in a page.

```
<textblock>
Click
<Hyperlink  navigateuri="photopage.xaml">here</Hyperlink> to view the photo.
</textblock >
```

Let’s discuss using Journal. Journal provides logic behind back and fwd. Internally two stacks. Back / fwd moves pages between stacks. Any other action empties the fwd stack. Back fwd can also nav. Containers goback and goforward and cangoback/fwd to avoid exception. Nav. Wnd. Always has journal but frame may depending on its journalownership=ownsjournal, usesparentjournal, Automatic (parent when hosted in wnd or frame). When frame has journal it has back/fwd buttons but can set navigationuivisibility=Hidden. When nav. With URI or hyperlink, always new instance. Can control when calling navigate with page so work to remember state. However, journalentry.keepalive attached prop. Can help for back fwd. Removefromjournal means a page is not in journal.

Other purposes of journal e.g. Application specific undo redo scheme. Nav. Container addbackentry and pass customcontentstate abstract class with a replay method that must be defined. Optionally journalentryname can be set. We use this in photo gallery for undoable image rotation.

```
[Serializable]
Class rotatestate : customcontentstate
{
Frameworkelement  element;
Double rotation;
Public  rotatestate(frameworkelement element, double rotation)
{
This.element= element;
This.rotation= rotation;
}
Public override string  journalentryname
{
Get{ return "Rotate " + rotation + "   "; }
}
Public override void Replay(navigationservice  navigationservice, navigationmode mode)
{
// Rotate the element by the specified  amount
Element.layouttransform = new rotatetransform(rotation);
}
}
```

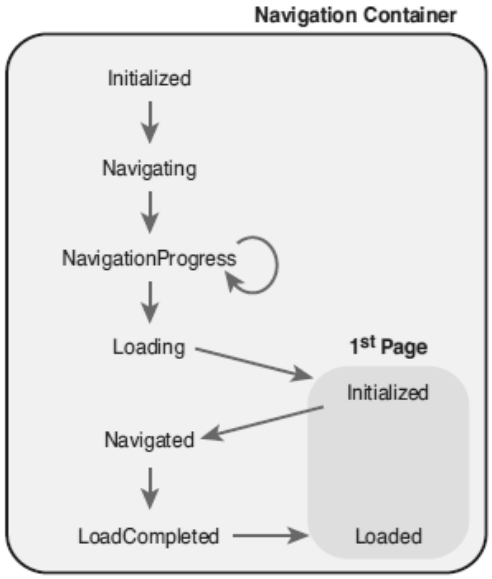


FIGURE 7.6 Navigation events that are raised when the first page is loaded.

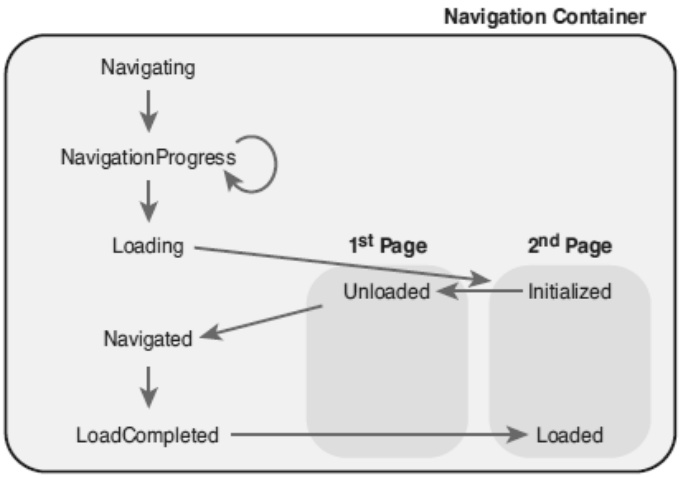


FIGURE 7.7 Navigation events that are raised when navigation occurs between two pages.

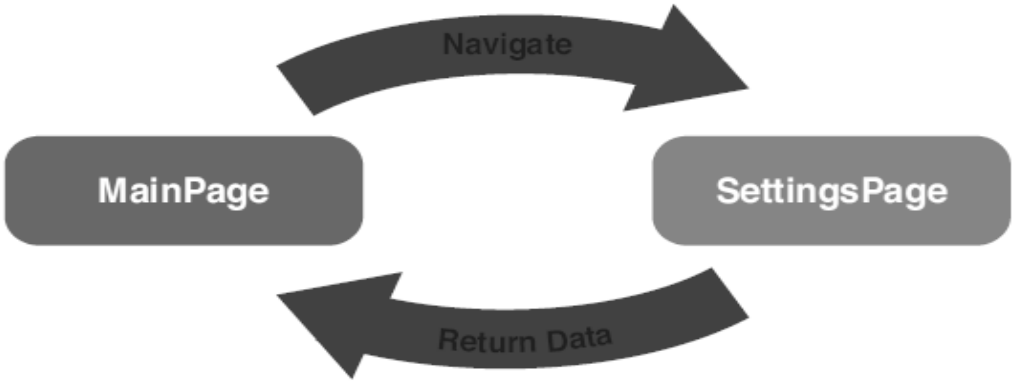
Navigationstopped is an event called instead of loadcompleted if an error occurs or nav. is cancelled. These events also defined in "Application" to handle for any nav. Cont. Html to html nav. Not in journal, no event raised. To send data to pages, Navigate overloads with an extra "object" param. Target page receives it in loadcompleted.

```
Int photoid = 10;
// Navigate to a page instance
Photopage  nextpage= new photopage();
This.navigationservice.Navigate(nextpage, photoid);
// Or navigate to a page via a URI
This.navigationservice.Navigate(new Uri("photopage.xam", urikind.Relative),photoid);


This.navigationservice.loadcompleted += new
Loadcompletedeventhandler(container_loadcompleted);
//
Void  container_loadcompleted(object sender, navigationeventargs e)
{
If(e.extradata != null)
Loadphoto((int)e.extradata);
}


Int photoid = 10;
// Navigate to a page instance
Photopage  nextpage= new photopage(photoid);
This.navigationservice.Navigate(nextpage);

Public photopage(int id)
{
Loadphoto(id);
}
```

```
// Navigate to a page by instance or URI
Application.Properties["photoid"] = 10;
This.navigationservice.Navigate(/* */);
If(Application.Properties["photoid"] != null)
Loadphoto((Application.Properties["photoid"]);
```

Let's see returning data with a pagefunction.

Pagefunction acts likea function.




```
<pagefunction
Xmlns="http://schemas.microsoft.com/winfx/2006/xaml/presentation"
Xmlns:x="http://schemas.microsoft.com/winfx/2006/xaml"
Xmlns:sys="clr-namespace:System;assembly=mscorlib"
X:Class="myproject.pagefunction1"
X:typearguments="sys:String"
Title="pagefunction1">
<Grid>
</Grid >
</pagefunction >
```

```
Pagefunction1nextpage = new pagefunction1<string>();
This.navigationservice.Navigate(nextpage);
Nextpage.Return += new returneventhandler<string>(nextpage_Return);
//
Void nextpage_Return(object sender, returneventargs<string> e)
{
String  returnvalue= e.Result;
}
Onreturn(new returneventargs<string>("the data"));
```

Chapter 27

Lecture 27

Let's discuss XAML Browser apps (XBAPS). Silverlight more popular to deliver partial-trust wpf content in a browser. Differences are that not all features available by default, nav is integrated into the browser, deployment different. In VS, create WPF browser application. Create UI in Page and compile and run. Really just online-only clickonce applications, with some special wpf handling for browser-integrated experience.

```
// Whoops! Partially  trusted code is not allowed to get this data!
Addfavorite(Environment.getfolderpath(Environment.specialfolder.mypictures));
```

Clickonce caching. VS increments. Or you can clear cache. Change some settings and its a xbap. But partial trust restricts many api. E.g. Above call exception. Requires fileiopermission which is not by default. Hit n trial to find what works. Local registry and file system, new windows (popups ok), unmanaged, but others dependent on browser, or on implementation details. Browserinterophelper.isbrowserhosted property to check if its a XBAP.

```
String  filecontents = null;
Openfiledialog ofd = new openfiledialog ();
If (ofd.showdialog() == true) // Result could be true, false, or null
{
Using (Stream s= ofd.openfile())
Using (streamreader sr= new streamreader(s))
{
Filecontents= sr.readtoend();
}
}
```

But u can still use rich text and media, isolated storage upto 512kb, arbitrary files on host web server, use browser file open dialog for local files. How parameters passed in. Either browserinterophelper.Source to retrieve the complete URL or browser cookie retrieved by Application.getcookie method. Any assembly marked with allowpartiallytrustedcallers and placed in GAC can be called by partial-trust code and it can be security loophole too.

There are also full-trust browser apps. In the project file you change

```
<targetzone>Internet</targetzone>
```

To this:

```
<targetzone>Custom</targetzone>
```

And in clickonce application manifest you add:

```
<permissionset  class="System.Security.permissionset" version="1"
ID="Custom" samesite="site" Unrestricted="true"/>
```

You get Integrated Navigation in XBAP. Many XBAP don't take advantage of navigation. Such applications can set shownavigationui=false on Page. IE 7 and later merge the journal and provide a more streamlined interface. However if it appears in an iframe, there is still a separate nav. Bar. To publish, use VS publishing wizard or mage tool in SDK and copy files to webserver which must be configured to serve it. Users can install and run by just navigating to a URL. No security prompt if you dont need non. Stdandard permissions.

Let's discuss downloading files on demand. We will assign a set of loose files to a download group in VS. Under publish-¿application files in proj. Prop. Then use api in System.Deployment.Application to prompt download and benotified e.g. Progress bar while app loads. Page1 starts, loads mygroup and shows page2.

```
Using System;
Using System.Windows.Controls;
Using System.Windows.Threading;
Using System.Deployment.Application;
Public partial class Page1: Page
{
Public Page1()
{
Initializecomponent();
}
Protected  override void oninitialized(eventargs e)
{
Base.oninitialized(e);
If(applicationdeployment.isnetworkdeployed)
{
Applicationdeployment.currentdeployment.downloadfilegroupcompleted +=
Delegate{
Dispatcher.begininvoke(dispatcherpriority.Send,
New dispatcheroperationcallback(gotopage2), null);
};
Applicationdeployment.currentdeployment.downloadfilegroupasync("mygroup");
}
Else
{
Gotopage2(null);
}
}
Private object  gotopage2(object o)
{
Return  navigationservice.Navigate(new Uri("Page2.xaml", urikind.Relative));
}
}
```

Additional events can be used for fine grained progress. Loose xaml is powerful sometimes instead of html. In summary, XBAP have similar impl. From desktop app to web app. Deployment just needs the right .net installed. Wpf 3.5 by default on win7, wpf4 now.

Let's discuss resources e.g. Bitmap, fonts, string tables. Wpf builds on top, has binary resources and logical resources. Binary resources are what rest of .net framework considers a resource. Even compiled xaml stored as a resource. Can be embedded in assembly, loose file that may or may not be known at compile time. Can be localizable or not. To define a binary resource, add a file and set the build action resource or content (loose file). Dont use embeddedresource. Its misleading name but wpf doesnt fully support. Adding as content and loading directly are kind of equal but resource is neat. Should be embed if localizable or single binary file benefits. Let's see how to assign binary resource whether embed or loose. Wont work if not added to proj. If not added be explicit about the path.

```
<stackpanel Grid.Column="1" Orientation="Horizontal" horizontalalignment="Center">
<Button x:Name="previousbutton" tooltip="Previous(Left Arrow)">
<Image Height="21" Source="previous.gif"/>
</Button >
```

```
<Button x:Name="slideshowbutton" tooltip="Play Slide Show (F11)">
<Image Height="21" Source="slideshow.gif"/>
</Button >
<Button x:Name="nextbutton" tooltip="Next (Right Arrow)">
<Image Height="21" Source="next.gif"/>
</Button >
</stackpanel >

<Image Height="21" Source="pack://siteoforigin:,,,/slideshow.gif"/>

Image image = new Image();
Image.Source = new bitmapimage(new Uri("pack://application:,,,/logo.jpg"));
```

Can use subfolders for embedded resources. From procedural code, cant use xaml spec. So there are shortcuts.

Chapter 28

Lecture 28

Let's discuss localizing binary resources. Can partition into satellite assembly and use locbaml to manage string localization. To spec. A default culture and auto. Build a satellite assembly, you can to set uiculture. Need to open project file in a text ed. Add under debug, release etc. Or where it effects all prop. If you rebuild your project with this setting in place, youll find an en-US folder alongside your assembly, containing the satellite assembly named assemblyname.resources.dll. Also mark assembly with the neutral resource language matching.

```
<Project >
<propertygroup>
<uiculture>en-US</uiculture>
[assembly: neutralresourceslanguage("en-US",
Ultimateresourcefallbacklocation.Satellite)]
```

Next, apply Uid directive to any element needing localization. Msbuild /t:updateuid projectname.csproj. Locbaml /parse projectname.g.en-US.resources /out:en-US.csv. Now edit and localize all strings. Locbaml /generate Project-Name.resources.dll /trans:fr-CA.csv /cul:fr-CA. Then copy the assembly with a name matching the locale. To test, System.Threading.Thread.currentthread.currentuiculture (and System.Threading.Thread.currentthread.currentculture) to an instance of the desired cultureinfo.

Logical resources are introduced by wpf. Arbitrary .net object stored and named in an elements Resources prop. Typically meant to be shared by multiple child objects. Frameworkelement and framework content element both have a Resources prop. Often are style or data providers. First examples using simple brushes. Much like css, share objects.

```
<Window xmlns="http://schemas.microsoft.com/winfx/2006/xaml/presentation"
Title="Simple Window" Background="Yellow">
  <dockpanel>
    <stackpanel  dockpanel.Dock="Bottom"
    Orientation="Horizontal"
    Horizontalalignment="Center">
      <Button  Background="Yellow"
      Borderbrush="Red" Margin="5">
        <Image Height="21" Source="zoom.gif"/>
      </Button >
      <Button  Background="Yellow"
      Borderbrush="Red"
      Margin="5">
        <Image Height="21"
        Source="defaultthumbnailsize.gif"/>
      </Button >
      <Button  Background="Yellow"
      Borderbrush="Red"
      Margin="5">
        <Image Height="21"
        Source="previous.gif"/>
      </Button >
      <Button  Background="Yellow"
      Borderbrush="Red"
      Margin="5">
        <Image Height="21"
```

```
        Source="slideshow.gif"/>
      </Button >
      <Button  Background="Yellow"
      Borderbrush="Red"
      Margin="5">
        <Image Height="21"
        Source="next.gif"/>
      </Button >
      <Button  Background="Yellow"
      Borderbrush="Red"
      Margin="5">
        <Image Height="21"
        Source="counterclockwise.gif"/>
      </Button >
      <Button  Background="Yellow"
      Borderbrush="Red"
      Margin="5">
        <Image Height="21"
        Source="clockwise.gif"/>
      </Button >
      <Button  Background="Yellow"
      Borderbrush="Red"
      Margin="5">
        <Image Height="21"
        Source="delete.gif"/>
      </Button >
    </stackpanel >
    <listbox/>
  </dockpanel >
</Window >

<Window xmlns="http://schemas.microsoft.com/winfx/2006/xaml/presentation"
Xmlns:x="http://schemas.microsoft.com/winfx/2006/xaml"
Title="Simple Window">
  <Window.Resources>
    <solidcolorbrush x:Key="backgroundbrush">Yellow</solidcolorbrush>
    <solidcolorbrush x:Key="borderbrush">Red</solidcolorbrush>
  </Window.Resources >
  <Window.Background>
    <staticresource  resourcekey="backgroundbrush"/>
  </Window.Background >
  <dockpanel>
    <stackpanel  dockpanel.Dock="Bottom"
    Orientation="Horizontal"
    Horizontalalignment="Center">
      <Button  Background="{staticresource  backgroundbrush}"
      Borderbrush="{staticresource  borderbrush}" Margin="5">
        <Image Height="21" Source="zoom.gif"/>
      </Button >
      <Button  Background="{staticresource  backgroundbrush}"
      Borderbrush="{staticresource  borderbrush}" Margin="5">
        <Image Height="21" Source="defaultthumbnailsize.gif"/>
      </Button >
      <Button  Background="{staticresource  backgroundbrush}"
      Borderbrush="{staticresource  borderbrush}" Margin="5">
        <Image Height="21" Source="previous.gif"/>
      </Button >
```

```
      <Button  Background="{staticresource  backgroundbrush}"
      Borderbrush="{staticresource  borderbrush}" Margin="5">
        <Image Height="21" Source="slideshow.gif"/>
      </Button >
      <Button  Background="{staticresource  backgroundbrush}"
      Borderbrush="{staticresource  borderbrush}" Margin="5">
        <Image Height="21" Source="next.gif"/>
      </Button >
      <Button  Background="{staticresource  backgroundbrush}"
      Borderbrush="{staticresource  borderbrush}" Margin="5">
        <Image Height="21" Source="counterclockwise.gif"/>
      </Button >
      <Button  Background="{staticresource  backgroundbrush}"
      Borderbrush="{staticresource  borderbrush}" Margin="5">
        <Image Height="21" Source="clockwise.gif"/>
      </Button >
      <Button  Background="{staticresource  backgroundbrush}"
      Borderbrush="{staticresource  borderbrush}" Margin="5">
        <Image Height="21" Source="delete.gif"/>
      </Button >
    </stackpanel >
    <listbox/>
  </dockpanel >
</Window >
<lineargradientbrush x:Key="backgroundbrush" startpoint="0,0" endpoint="1,1">
  <gradientstop Color="Blue" Offset="0"/>
  <gradientstop Color="White" Offset="0.5"/>
  <gradientstop Color="Red" Offset="1"/>
```


```
</lineargradientbrush >
```

Now u can change brushes in one place and have different effects. Staticresource markup extension walks the logical tree or even application level or system resources. Root level or app level resource dict for max sharing. Closest to element chosen. For multi-threaded either Frozen or x:Shared=false.

Let's discuss static vs Dynamic Resources. Dynamicresource reapplied every time it changes nothing special about the resources. Only if you want to see updates or not. Dynamic more overhead but demand loaded. Dynamic can only be used to set dep. Prop. Values. Static can even abstract whole controls.

```
<Window xmlns="http://schemas.microsoft.com/winfx/2006/xaml/presentation"
Xmlns:x="http://schemas.microsoft.com/winfx/2006/xaml">
<Image Height="21" Source="zoom.gif"/>
</Window >
```

Is equivalent to this Window :

```
<Window xmlns="http://schemas.microsoft.com/winfx/2006/xaml/presentation"
Xmlns:x="http://schemas.microsoft.com/winfx/2006/xaml">
  <Window.Resources>
    <Image x:Key="zoom" Height="21" Source="zoom.gif"/>
```

```
  </Window.Resources >
  <stackpanel>
    <staticresource  resourcekey="zoom"/>
  </stackpanel >
</Window >
```

Can be a way to factor but Image can only have one parent so cant share last, static doesnt support fwd. Ref. With dynamic

The following is possible.

```
<Window xmlns="http://schemas.microsoft.com/winfx/2006/xaml/presentation"
Xmlns:x="http://schemas.microsoft.com/winfx/2006/xaml"
Title="Simple Window" Background="{dynamicresource  backgroundbrush}">
  <Window.Resources>
    <solidcolorbrush x:Key="backgroundbrush">Yellow</solidcolorbrush>
    <solidcolorbrush x:Key="borderbrush">Red</solidcolorbrush>
  </Window.Resources >
</Window >
```

Without sharing, x:Shared=false e.g. New instance of image multiple times.

```
<Window xmlns="http://schemas.microsoft.com/winfx/2006/xaml/presentation"
Xmlns:x="http://schemas.microsoft.com/winfx/2006/xaml">
  <Window.Resources>
    <Image x:Shared="False" x:Key="zoom" Height="21" Source="zoom.gif"/>
  </Window.Resources >
  <stackpanel>
    <!--  Applying  the  resource  multiple  times  works!-->
    <staticresource  resourcekey="zoom"/>
    <staticresource  resourcekey="zoom"/>
    <staticresource  resourcekey="zoom"/>
  </stackpanel >
</Window >
```

On in procedural code

```
Window.Resources.Add("backgroundbrush", new solidcolorbrush(Colors.Yellow));
Window.Resources.Add("borderbrush", new solidcolorbrush(Colors.Red));

Button button= new Button();
// The Button must descend from the Window before looking up resources:
Stackpanel.Children.Add(button);
Button.Background = (Brush)button.findresource("backgroundbrush");
Button.borderbrush = (Brush)button.findresource("borderbrush");

Button button= new Button();
Button.setresourcereference(Button.backgroundproperty, "backgroundbrush");
Button.setresourcereference(Button.borderbrushproperty, "borderbrush");
Button button= new Button();
Button.Background = (Brush)window.Resources["backgroundbrush"];
Button.borderbrush = (Brush)window.Resources["borderbrush"];
```

Following are some XAML and equivalent C# examples.

```
XAML:
<Button  Background="systemcolors.windowbrush"/>
C\#:
Button.Background = (Brush)new
Brushconverter().convertfrom("systemcolors.windowbrush");
XAML:
```

```
<Button  Background="{x:Static  systemcolors.windowbrush}"/>
C\#:
Button.Background = systemcolors.windowbrush;

XAML:
<Button  Background="{ staticresource  systemcolors.windowbrushkey}"/>
C\#:
Button.Background = (Brush)findresource("systemcolors.windowbrushkey");
XAML:
<Button  Background="{ staticresource{x:Static  systemcolors.windowbrush}}"/>
C\#:
Button.Background = (Brush)findresource(systemcolors.windowbrush);

XAML:
<Button  Background="{ staticresource{x:Static  systemcolors.windowbrushkey}}"/>
C\#:
Button.Background = (Brush)findresource(systemcolors.windowbrushkey);

XAML:
<Button  Background="{ dynamicresource {x:Static  systemcolors.windowbrushkey}}"/>
C\#:
Button.setresourcereference(
Button.backgroundproperty, systemcolors.windowbrushkey);
```

In summary, resources really imp. In professional apps, enable localization, increase productivity as it consolidates / shares. Most interesting use is with styles and data binding.

Chapter 29

Lecture 29

We will discuss data binding now. Data means an arbitrary .net obj. Data binding, data templates, data triggers are related concepts. Data can be collection obj, xml file, web service, db table, custom obj, even wpf element eg button. So data binding is typing together arbitrary objects. Classic scenario is a visual rep. (e.g. Listbox or datagrid) of items in an xml file, db, or in-memory collection. Instead of iterating and adding items, tell listbox to get its data from another source, keep them up to date, format them etc.

Binding binds two properties together and keeps a communication channel open. Setup Binding once and let it handle all the sync. Eg. In proc. Code.

```
<textblock x:Name="currentfolder" dockpanel.Dock="Top"
Background="aliceblue" fontsize="16" />

Void  treeview_selecteditemchanged(object sender,
Routedpropertychangedeventargs<object> e)
{
Currentfolder.Text =
(treeview.selecteditem as treeviewitem).Header.tostring();
Refresh();
}
Public  mainwindow()
{
Initializecomponent();
Binding binding= new Binding();
// Set source object
Binding.Source = treeview;
// Set source property
Binding.Path= new propertypath("selecteditem.Header");
// Attach to target property
Currentfolder.setbinding(textblock.textproperty, binding);
}
```

When an item with no header selected, then a default value (in this case) returned. No exception is raised. Binding has a source prop and a target prop. Source is an object and a prop. Path. Can also use bindingoperations.setbinding. Bindingoperations.clearbinding. Bindingoperations.clearallbindings. Or set the target prop to a new val. (but clear would allow receiving values from lower prop. Sources).

There is a binding markup extension as well.

```
<textblock x:Name="currentfolder" dockpanel.Dock="Top"
Text="{Binding  elementname=treeview, Path=selecteditem.Header}"
Background="aliceblue" fontsize="16" />
<textblock x:Name="currentfolder" dockpanel.Dock="Top"
Text="{Binding Source={x:Reference  treeview}, Path=selecteditem.Header}"
Background="aliceblue" fontsize="16" />
<textblock Text="{Binding targetnullvalue=Nothing is selected.}" />
<Label x:Name="numitemslabel"
Content="{Binding Source={staticresource  photos}, Path=Count}"
Dockpanel.Dock="Bottom"/>
System.componentmodel.inotifypropertychanged
<Label x:Name="numitemslabel"
```

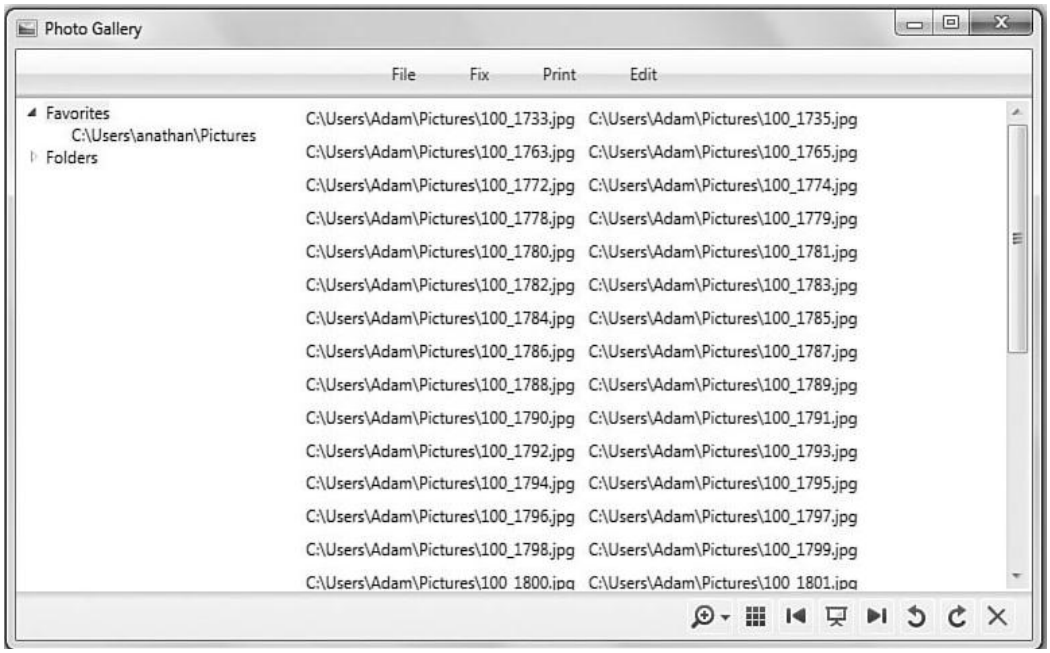
```
Content="{Binding Source={staticresource  photos}}"
Dockpanel.Dock="Bottom"/>
```

Could also set Source but using elementname is easier. Can also set value for nothing is selected.

We can also bind to plain .net properties. But the source obj should implement the System.component model. notify property interface, which has a single propertychanged event. (the right way) and Implement an xxxchanged event, where XXX is the name of the property whose value changed. Public class Photos : observablecollection¡Photo¿ target must be a dep. Prop. When binding to an object, Path is optional

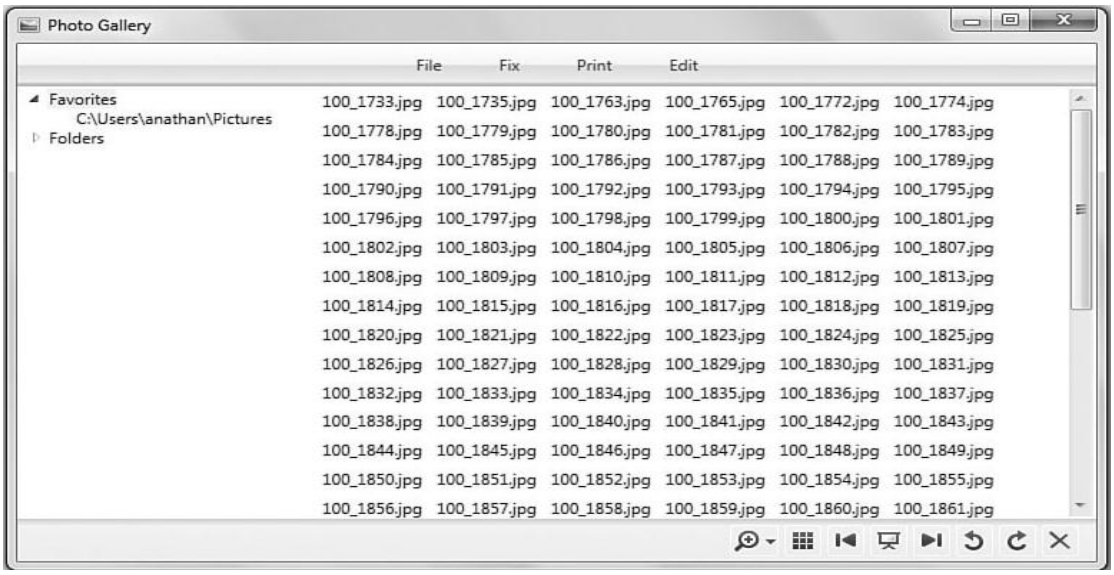

```
<listbox x:Name="picturebox"
Itemssource="{Binding Source={staticresource  photos}}">
</listbox >
```

We can also bind to a collection. How abt binding listbox in photo app to photo collection. When a new dir selected, clear lisbox and created listboxitem for each item. When user del or rename, an event (because of filesystemwatcher) and manually refresh. Raw binding i.e. Items is not dep. Prop but itemsource is. Source prop must impl. Inotifycollectionchanged (aka observablecollection). The simplest



The displaymemberpath property can be used to improve display.

```
<listbox x:Name="picturebox" displaymemberpath="Name"
Itemssource="{Binding Source={staticresource  photos}}">
</listbox >
```



We cant mix Items and itemsource. But we always retrieve from Items. We could add an Image property but there are more flexible ways. 2 ways: data template and value converter. Selected item can also be synchronized but scrolling not synchronized, only first selection synchronized.

```
<listbox  issynchronizedwithcurrentitem="True" displaymemberpath="Name"
Itemssource="{Binding Source={staticresource  photos}}"></listbox>
<listbox  issynchronizedwithcurrentitem="True" displaymemberpath="datetime"
Itemssource="{Binding Source={staticresource  photos}}"></listbox>
<listbox  issynchronizedwithcurrentitem="True" displaymemberpath="Size"
Itemssource="{Binding Source={staticresource  photos}}"></listbox>
```

Implicit data source is provided by a Data Context. We set datacontext of a parent and then dont specify Source or elementname or set parent.datacontext = photos;. It is useful when plugging in resources: usage context or decl.context.

```
<stackpanel  datacontext="{staticresource  photos}">
<Label x:Name="numitemslabel"
Content="{Binding Path=Count}"/>
<listbox x:Name="picturebox" displaymemberpath="Name"
Itemssource="{Binding}">
</listbox >
</stackpanel >
```

Control rendering is easy without binding but there are benefits of data binding. WPF gives three ways. String formatting which only works if the target prop is a string. {} to escape. Can add 1000 seperator etc.

```
<textblock x:Name="numitemslabel"
Text="{Binding  stringformat={}{0} item(s),Source={staticresource
photos},Path=Count}"dockpanel.Dock="Bottom"/>
<textblock x:Name="numitemslabel" dockpanel.Dock="Bottom">
<textblock.Text>
<Binding Source="{staticresource  photos}" Path="Count">
<Binding.stringformat>{0} item(s)</Binding.stringformat>
</Binding >
</textblock.Text >
</textblock >
```

String formatting can be used even without data binding.

```
<listbox  itemstringformat="{}{0:C}"
Xmlns:sys="clr-namespace:System;assembly=mscorlib">
  <sys:Int32>-9</sys:Int32>
  <sys:Int32>9</sys:Int32>
  <sys:Int32>1234</sys:Int32>
  <sys:Int32>1234567</sys:Int32>
</listbox >
```

TABLE 13.1 String Format Properties Throughout WPF

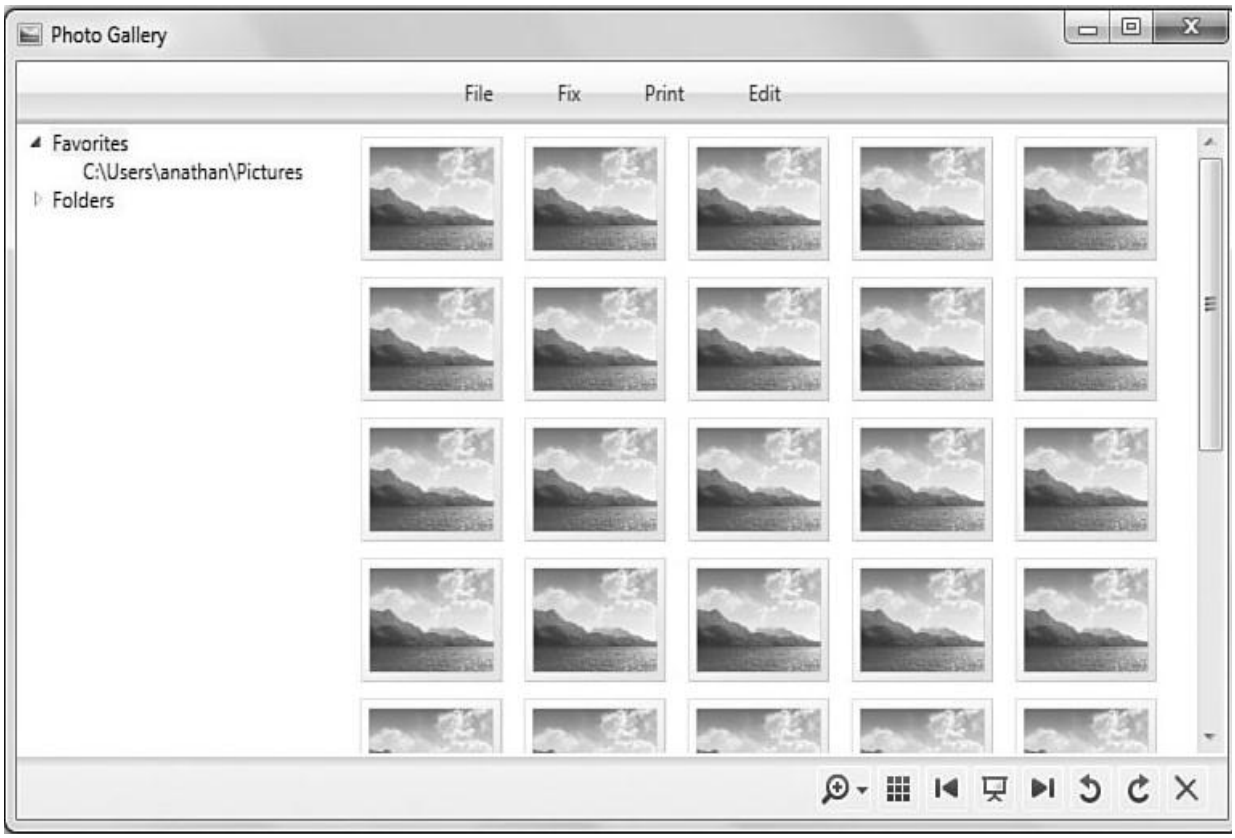
Property	Classes
StringFormat	BindingBase
ContentStringFormat	ContentControl, ContentPresenter, TabControl
ItemStringFormat	ItemsControl, HierarchicalDataTemplate
HeaderStringFormat	HeaderedContentControl, HeaderedItemsControl, DataGridColumn, GridViewColumn, GroupStyle
ColumnHeaderStringFormat	GridView, GridViewHeaderRowPresenter

With Data Template, UI auto-applied to arbitrary .net object when it is rendered. By setting these prop. You can swap in a complete new visual tree easy in xaml, cumbersome in proc. Code try with photo gallery

TABLE 13.2 Properties of Type DataTemplate Throughout WPF

Property	Classes
ContentTemplate	ContentControl, ContentPresenter, TabControl
ItemTemplate	ItemsControl, HierarchicalDataTemplate
HeaderTemplate	HeaderedContentControl, HeaderedItemsControl, DataGridRow, DataGridColumn, GridViewColumn, GroupStyle
SelectedContentTemplate	TabControl
DetailsTemplate	DataGridRow
RowDetailsTemplate	DataGrid
RowHeaderTemplate	DataGrid
ColumnHeaderTemplate	GridView, GridViewHeaderRowPresenter
CellTemplate	DataGridTemplateColumn, GridViewColumn
CellEditingTemplate	DataGridTemplateColumn

```
<listbox x:Name="picturebox"
Itemssource="{Binding Source={staticresource  photos}}">
<listbox.itemtemplate>
<datatemplate>
<Image Source="placeholder.jpg" Height="35"/>
</datatemplate >
</listbox.itemtemplate >
</listbox >
```



With a template there is implicit datacontext. Template can be used on non-data bound but inside the element it almost always make sense to use data binding. Template can be shared as resources. Template can even be auto-applied to some type by setting its datatype prop. There is also hierarchicaldatatemplate that understands hierarchies. Can be used with Treeview or menu.

```
<listbox x:Name="picturebox"
Itemssource="{Binding Source={staticresource  photos}}">
<listbox.itemtemplate>
<datatemplate>
<Image Source="{Binding Path=fullpath}" Height="35"/>
</datatemplate >
</listbox.itemtemplate >
</listbox >
```

Value Convertors morph source value into something else. Introduce custom logic and can change type etc. E.g.brush based on some enumeration. Can be used to customize display.

```
<Window.Resources>
<local:counttobackgroundconverter x:Key="myconverter"/>
</Window.Resources >

<Label Background="{Binding Path=Count, Converter={staticresource  myconverter},
Source={staticresource  photos}}" />
Public class  counttobackgroundconverter : ivalueconverter
{
Public object Convert(object value, Type targettype,
            object parameter, cultureinfo  culture)
{
If(targettype != typeof(Brush))
Throw new invalidoperationexception("The target must be a Brush!");
// Let Parse throw an exception if the input is bad
Int num = int.Parse(value.tostring());
Return(num == 0 ? Brushes.Yellow: Brushes.Transparent);
}
Public object  convertback(object value, Type targettype,
Object parameter, cultureinfo  culture)
{
Return  dependencyproperty.unsetvalue;
}
}
```

Chapter 30

Lecture 30

In last lecture, we discused data binding, binding object, markup extension, bind to any prop with inotifypropertychanged, target must be dependency property, bind to obj. So no property path, binding to collection, issynchronized with current item, datacontext, displaymemberpath, stringformat, datatemplate, valueconverter.

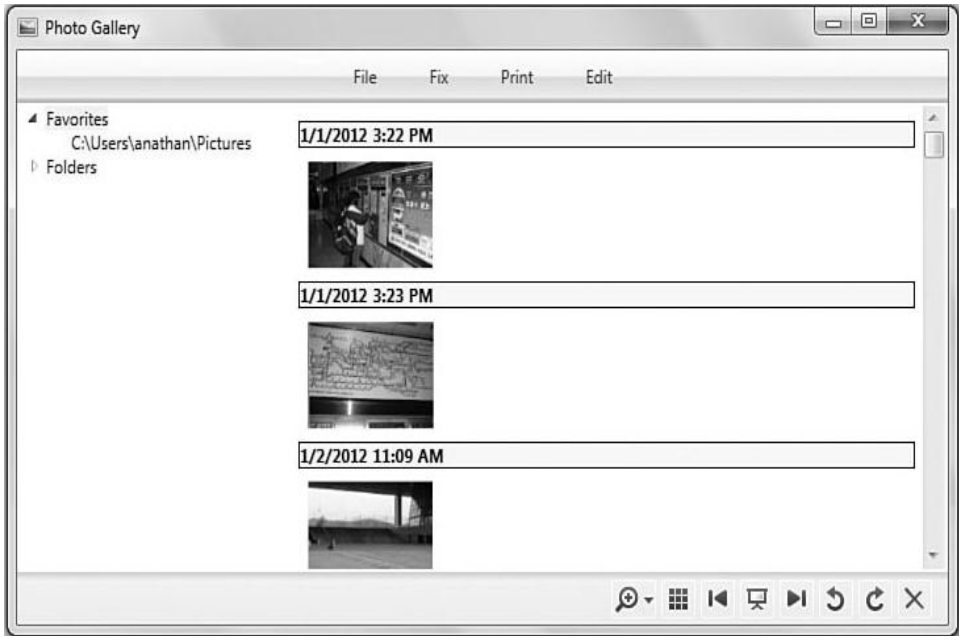
Let's discuss customizing collection view. When issynchronizedwithcurrentitem= true, where is the current item. A default "view" inserted. Object implementing icollectionview. Has also support for sorting, grouping, filtering, and navigating. Let's discuss these four and multiple views for the same source obj.

Sortdescriptions prop. Is a collection of sortdescription which chooses a field and order, sort by datetime, then by name. A clear method to return to unsorted. Eg. Three buttons to sort and toggle.

```
Sortdescription sort = new sortdescription("Name", listsortdirection.Ascending);

View.sortdescriptions.Add(new sortdescription("datetime", listsortdirection.Descending);
View.sortdescriptions.Add(new sortdescription("Name", listsortdirection.Ascending);

// Click event handlers for three different  Buttons:
Void sortbyname_Click(object sender, routedeventargs e)
{
Sorthelper("Name");
}
Void sortbydatetime_Click(object sender, routedeventargs e)
{
Sorthelper("datetime");
}
Void sortbysize_Click(object sender, routedeventargs e)
{
Sorthelper("Size");
}
Void sorthelper(string  propertyname)
{
// Get the default view
Icollectionview view = collectionviewsource.getdefaultview(
This.findresource("photos"));
// Check if the view is already sorted ascending by the current  property
If (view.sortdescriptions.Count > 0
&& view.sortdescriptions[0].propertyname == propertyname
&& view.sortdescriptions[0].Direction == ", listsortdirection.Ascending)
{
// Already sorted ascending, so t o g g l e by sorting  descending
View.sortdescriptions.Clear();
View.sortdescriptions.Add(new sortdescription(
Propertyname, listsortdirection.Descending));
}
Else
{
// Sort ascending
View.sortdescriptions.Clear();
View.
```

```
Tdescriptions.Add(new sortdescription(
Propertyname, listsortdirection.Ascending));
}
}
// Get the default view
Icollectionview view  = collectionviewsource.getdefaultview(
This.findresource("photos"));
// Do the grouping 46 view.groupdescriptions.Clear();
View.groupdescriptions.Add(new propertygroupdescription("datetime"));
<liszbox x:Name="picturebox"
Itemssource="{Binding Source={staticresource  photos}}">
<listbox.groupstyle>
<groupstyle>
<groupstyle.headertemplate>
<datatemplate>
<Border  borderbrush="Black" borderthickness="1">
<textblock Text="{Binding Path=Name}" fontweight="Bold"/>
</Border >
</datatemplate >
</groupstyle.headertemplate >
</groupstyle >
</listbox.groupstyle >
</listbox >
```

There is no explicit relationship with listbox. If additional controls bound, they would sort with it.Now groupdescriptions prop. Containing propertygroupdescription obj. But it has no effect without the groupstyle prop.



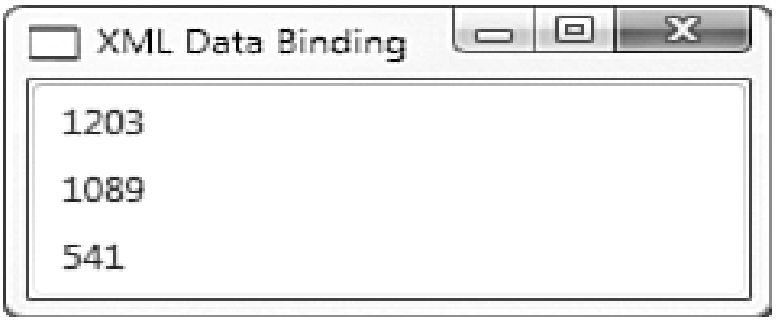

```
<listbox x:Name="picturebox"
Itemssource="{Binding Source={staticresource  photos}}">
<listbox.groupstyle>
<x:Static Member="groupstyle.Default"/>
</listbox.groupstyle >
</listbox >
// Get the default view
```

```
Icollectionview view = collectionviewsource.getdefaultview(
This.findresource("photos"));
// Do the grouping
View.groupdescriptions.Clear();
View.groupdescriptions.Add(
New propertygroupdescription("datetime", new datetimetodateconverter()));
Public class  datetimetodateconverter : ivalueconverter
{
Public object Convert(object value, Type targettype, object parameter,
Cultureinfo  culture)
{
Return((datetime)value).tostring("MM/dd/yyyy");
}
Public object  convertback(object value, Type targettype, object parameter,
Cultureinfo  culture)
{
Return  dependencyproperty.unsetvalue;
}
}
```


```
Icollectionview view = collectionviewsource.getdefaultview(
This.findresource("photos"));
View.Filter = delegate(object o) {
Return((o as Photo).datetime datetime.Now).Days <= 7;
};
Icollectionview view = collectionviewsource.getdefaultview(
This.findresource("photos"));
View.Filter = (o)=> { return (
(o as Photo).datetime datetime.Now).Days <= 7;
};
Void previous_Click(object sender, routedeventargs e)
{
// Get the default view
Icollectionview view = collectionviewsource.getdefaultview(
This.findresource("photos"));
// Move backward
View.movecurrenttoprevious();
// Wrap around to the end
```

```
If(view.iscurrentbeforefirst) view.movecurrenttolast();
}
Void next_Click(object sender, routedeventargs e)
{
// Get the default view
Icollectionview view = collectionviewsource.getdefaultview(
This.findresource("photos"));
// Move forward
View.movecurrenttonext();
// Wrap around to the beginning
If(view.iscurrentafterlast) view.movecurrenttofirst();
}
"{Binding Path=/}"
"{Binding Path=/datetime}"
"{Binding Path=Photos/}"
"{Binding Path=Photos/datetime}"
Collectionviewsource  viewsource= new collectionviewsource();
Viewsource.Source = photos;
<Window.Resources>
<local:Photos x:Key="photos"/>
<collectionviewsource x:Key="viewsource" Source="{staticresource  photos}"/>
</Window.Resources >
<listbox x:Name="picturebox"
Itemssource="{Binding Source={staticresource  photos  viewsource}}" >
</listbox >
<collectionviewsource x:Key="viewsource"
Filter="viewsource_Filter"
Source="{staticresource  photos}">
<collectionviewsource.sortdescriptions>
<componentmodel:sortdescription  propertyname="datetime"
Direction="Descending"/>
</collectionviewsource.sortdescriptions >
<collectionviewsource.groupdescriptions>
<propertygroupdescription  propertyname="datetime"/>
</collectionviewsource.groupdescriptions >
</collectionviewsource >
Void viewsource_Filter(object sender, filtereventargs e)
{
E.Accepted = ((e.Item as Photo).datetime datetime.Now).Days <= 7;
}
<Window xmlns="http://schemas.microsoft.com/winfx/2006/xaml/presentation"
Xmlns:x="http://schemas.microsoft.com/winfx/2006/xaml"
Title="XML Data Binding">
<Window.Resources>
<xmldataprovider x:Key="dataprovider" xpath="gamestats">
<x:xdata>
<gamestats xmlns="">
<!--  One  stat  per  game  type-->
<gamestat Type="Beginner">
<highscore>1203</highscore>
</gamestat >
<gamestat Type="Intermediate">
<highscore>1089</highscore>
</gamestat >
<gamestat Type="Advanced">
<highscore>541</highscore>
</gamestat >
```

```
</gamestats >
</x:xdata >
</xmldataprovider >
</Window.Resources >
<Grid>
<listbox  itemssource="{Binding
Source={staticresource  dataprovider},
Xpath=gamestat/highscore}"/>
</Grid >
</Window >
```



Sorting is applied before grouping. First sorting criteria should be same as grouping otherwise output doesnt make much sense. Can pass null to get total custom control in the value converter.

Filtering gives a property Filter of type Predicate¡Object¿. Its null by default e.g. Show only photos from last 7 days.

Navigation means managing the current item, not the other kind of nav. Icollectionview has currentitem currentposition and also methods for changing them. E.g. Prev/next photo buttons are handled like this view initializes to 0 and first item listbox initializes to -1 and null (unselected).

Prop. Paths in Bindings are useful for master/detail interfaces. Sorting, grouping, filtering automatic. But navigation only when issynchronizedwithcurrentitem=true. Otherwise selecteditem and currentitem are separate. Collectionviewsource can be used to create new views and applied to targets. Collectionviewsource has its own sortdescriptions and groupdescriptions properties and a Filter event to be used from xaml. Must include another namespace. Issynchronizedwithcurrentitem=true by default for custom views. Have to explicitly set false. Kind of a bit inconsistent. But we acknowledging view existance by custom view.

Let's discuss data providers. Source obj can be arbitrary. You could bind to db, registry, excel spreadsheet etc. With enough code. An obj. That exposes right props and notifications and handles messy details work involved might overweight benefits if writing all logic yourself two generic data-binding-friendly way to expose common items. Xmldataprovider and objectdataprovider. Starting with wpf 3.5 sp1 data binding works with LINQ (language independent query) you can set Source or datacontext to a LINQ and the enumerable result is used. Now with LINQ to SQL and LINQ to XML, is an easy way than using wpf data providers.

We use xdata to avoid compiler errors. Namespace pollution avoided with xmlns=. Bindings xpath prop used rather than Path. If external file, even easier. "gamestat/@Type" would fill list with each gamestats Type attribute. Data provided in xmlnode etc. Objects from System.Xml so you can use Path and xpath together. Needs hierarchical data template if you want to bind to xml tree.

```
<xmldataprovider x:Key="dataprovider" xpath="gamestats" Source="gamestats.xml"/>
<Label Content="{Binding Source={staticresource  dataprovider},
Xpath=gamestat/highscore, Path=outerxml}"/>
```

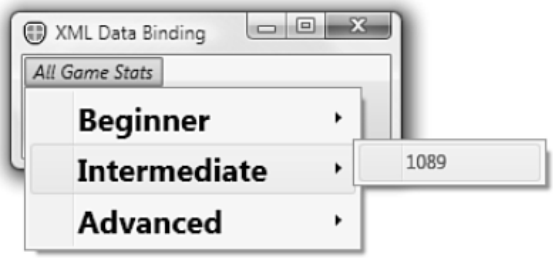
Chapter 31

Lecture 31

In the last lecture, we discussed the view between binding source and target sorting, grouping, filtering, navigating. Creating additional views, data providers (xml and object), used xml data provider to fill a listbox, now use hierarchical data template.

```
<Window xmlns="http://schemas.microsoft.com/winfx/2006/xaml/presentation"
Xmlns:x="http://schemas.microsoft.com/winfx/2006/xaml"
Title="XML Data Binding">
  <Window.Resources>
    <hierarchicaldatatemplate  datatype="gamestats"
    Itemssource="{Binding xpath=*}">
      <textblock  fontstyle="Italic"
      Text="All Game Stats"/>
    </hierarchicaldatatemplate >
    <hierarchicaldatatemplate  datatype="gamestat"
    Itemssource="{Binding xpath=*}">
      <textblock  fontweight="Bold" fontsize="20"
      Text="{Binding xpath=@Type}"/>
    </hierarchicaldatatemplate >
    <datatemplate  datatype="highscore">
      <textblock  Foreground="Blue" Text="{Binding xpath=.}"/>
    </datatemplate >
    <xmldataprovider x:Key="dataprovider" xpath="gamestats">
      <x:xdata>
        <gamestats xmlns="">
          <!--  One  stat  per  game  type-->
          <gamestat Type="Beginner">
            <highscore>1203</highscore>
          </gamestat >
          <gamestat Type="Intermediate">
            <highscore>1089</highscore>
          </gamestat >
          <gamestat Type="Advanced">
            <highscore>541</highscore>
          </gamestat >
        </gamestats >
      </x:xdata >
    </xmldataprovider >
  </Window.Resources >
  <Grid>
    <treeview
    Itemssource="{Binding Source={staticresource  dataprovider},
Xpath=.}"/>
  </Grid >
</Window >
```

The TreeView in Listing 13.3



Changing TreeView to Menu

Hierarchicaldatatemplate for every node and datatemplate for a leaf node. Hierarchicaldatatmplate allow specifyingchildren using itemsource prop. =all children. Datatype means effect all instances within the scope. Datatype corresponds to xmlnode name. No key. Internally datatype value used as key.

```
<rss version="2.0" xmlns:atom="http://www.w3.org/2005/Atom"
Xmlns:georss="http://www.georss.org/georss">
</rss >
Xmldataprovider  Source="http://twitter.com/statuses/user_timeline/24326956.rss"
Xmlnamespacemanager="{staticresource  namespacemapping}"
Xpath="rss/channel" x:Key="dataprovider"/>
<xmlnamespacemappingcollection x:Key="namespacemapping">
  <xmlnamespacemapping Uri="http://www.w3.org/2005/Atom" Prefix="atom"/>
  <xmlnamespacemapping Uri="http://www.georss.org/georss" Prefix="georss"/>
</xmlnamespacemappingcollection >
"{Binding xpath=atom:link}"
```

Now Let's discuss objectdataprovider. .net object as a data source. So what does that add against binding directly to a .net obj. Declaratively instantiate the source object with a parameterized constructor. Bind to a method on the source object. Have more options for asynchronous data binding. When binding not quick, async soui doesnt stuck. Wpf has two ways to mark async. Isasync prop of Binding, xmldataprovider and objectdataprovider have an isasynchronous prop. False by default on objdp, true by default for xmldp. Create source obj on background thread. When isasync ture (false by default), source prop invoked on background thread property getters supposedto be fast so shouldn't need.

First example wraps photos. Its the same, even binding path because binding "unwraps". It can also create the object given its type.

```
<Window.Resources>
<local:Photos x:Key="photos"/>
<objectdataprovider x:Key="dataprovider"
Objectinstance="{staticresource  photos}"/>
</Window.Resources >
<Window.Resources>
<!--  The  collection  object  is  instantiated  internally  by  objectdataprovider:-->
<objectdataprovider x:Key="dataprovider" objecttype="{x:Type local:Photos}"/>
</Window.Resources >
```

Binding to a method is useful for classes that are not designed for data binding. Imagine photos class had getfoldername method. Can use methodparameters. If Path used. It would apply to the obj returned.

```
<objectdataprovider x:Key="dataprovider" objecttype="{x:Type local:Photos}">
```

```
<objectdataprovider.constructorparameters>
<sys:Int32>23</sys:Int32>
</objectdataprovider.constructorparameters >
</objectdataprovider >
<objectdataprovider x:Key="dataprovider"
Objecttype="{x:Type local:Photos}"
Methodname="getfoldername"/>
```

Binding.Mode can be oneway i.e. The target is updated whenever the source changes. Twoway i.e. A change to either the target or source updates the other. Onewaytosource. This is the opposite of oneway. The source is updated whenever the target changes. Onetime. This works just like oneway, except changes to the source are not reflected at the target. The target retains a snapshot of the source at the time the Binding is initiated.

Useful for editable datagrid and textbox (by default) thats why convertback method. Only convertback on oneway-tosource.

Updatesourcetrigger. Do you want the two way source to be updated on every key stroke. Propertychanged. Changed when target prop value changes. Lostfocus. Changed when focus lost. Explicit. Call bindingexpression.updatesource which you can get from any frameworkelement.getbindingexpression

We can also have validation rules. Want to validate as soon as possible for user feedback. Without binding you could insert custom logic but now the data is auto-pushed to target. Let's say we want valid .jpg file. 2 ways. Validation rule or use source exceptions. Binding has validationrules prop that can be set to one or more validationrule derived objects.

```
<textbox>
<textbox.Text>
<Binding >
<Binding.validationrules>
<local:jpgvalidationrule/>
</Binding.validationrules >
</Binding >
</textbox.Text >
</textbox >
Public class  jpgvalidationrule : validationrule
{
Public override  validationresult  Validate(object value, cultureinfo  cultureinfo)
{
String filename = value.tostring();
If(!File.Exists(filename))
Return new validationresult(false, "Value is not a valid file.");
If(!Filename.endswith(".jpg", stringcomparison.invariantcultureignorecase))
Return new validationresult(false, "Value is not a .jpg file.");
Return new validationresult(true, null);
}
}
```

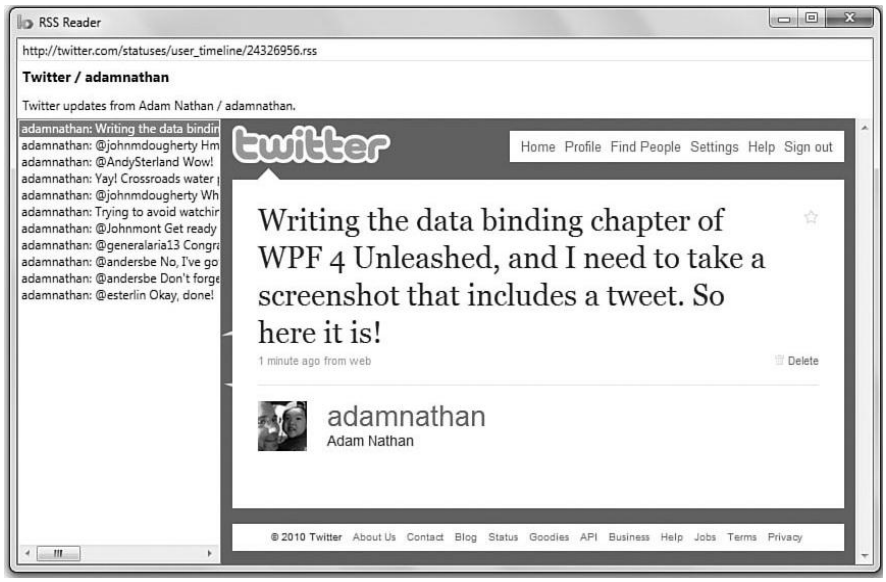
Validation check whenever attempt to update target. In this case on lostfocus because it is default. When invalid, an error adorner rendered on top of element. By default thin red border but can use Validation.errortemplate attached prop on target element to customize it, also other properties for custom control.

```
<textbox>
<textbox.Text>
<Binding >
```

```
<Binding.validationrules>
<exceptionvalidationrule/>
</Binding.validationrules >
</Binding >
</textbox.Text >
</textbox >
<textbox>
<textbox.Text>
<Binding >
<Binding.validationrules>
<exceptionvalidationrule/>
<dataerrorvalidationrule/>
</Binding.validationrules >
</Binding >
</textbox.Text >
</textbox >
<textbox>
<textbox.Text>
<Binding  validatesonexceptions="True" validatesondataerrors="True" />
</textbox.Text >
</textbox >
```

But what if already exceptions thrown by target. Also the source can implement System.componentmodel.idataerrorinfo. Easier way in wpf 3.5 sp1.

Let's write an RSS reader without code. A two-way textbox so that user can change feed address. Bindsdirectlytosource so path does not refer to the rss feed. Updatesourcetrigger of propertychanged so update with every keystroke. Listbox/frame share the source. Master detail.




```
<Window xmlns="http://schemas.microsoft.com/winfx/2006/xaml/presentation"
Xmlns:x="http://schemas.microsoft.com/winfx/2006/xaml" Title="RSS Reader">
  <Window.Resources>
    <xmldataprovider x:Key="Feed"
    Source="http://twitter.com/statuses/user_timeline/24326956.rss"/>
  </Window.Resources >
  <dockpanel
```

```
  Datacontext="{Binding Source={staticresource Feed},
Xpath=/rss/channel/item}">
    <textbox  dockpanel.Dock="Top"
    Text="{Binding Source={staticresource Feed},
Bindsdirectlytosource=true, Path=Source,
Updatesourcetrigger=propertychanged}"/>
    <Label dockpanel.Dock="Top"
    Content="{Binding xpath=/rss/channel/title}"
    Fontsize="14" fontweight="Bold"/>
    <Label dockpanel.Dock="Top"
    Content="{Binding xpath=/rss/channel/description}"/>
    <listbox  dockpanel.Dock="Left" displaymemberpath="title"
    Itemssource="{Binding}"
    Issynchronizedwithcurrentitem="True" Width="300"/>
    <Frame Source="{Binding xpath=link}"/>
  </dockpanel >
</Window >
```

Chapter 32

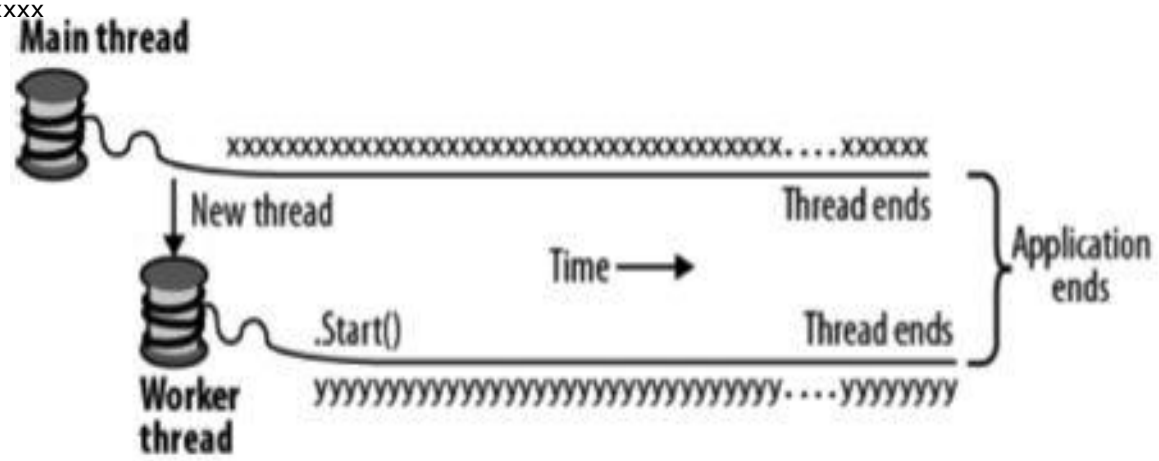
Lecture 32

Concurrency is More than one thing happening at the same time. Comparison with events. Why still concurrency. Responsive user interface. Simultaneous requests. Parallel programming.

Thread: execution path that can proceed independently of others. Normal processes (programs in exec.) Have one thread. Multithreaded programs have more and can share data. Let's create one.

```
Class threadtest
{
Static void Main()
{
Thread t = new Thread(writey);
T.Start();

For(int i= 0; i < 1000; i++) Console.Write("x");
}

Static void writey()
{
For(int i= 0; i < 1000; i++) Console.Write("y");
}
}
```

```
Typical Output:
Xxxxxxxxxxxxxyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyxxxxxxxxxxxxxxxxxxx
xxxxxxx
```



Interleaved because of time slicing. Real parallel on multicore but still repeated blocks cuz of the way Console handles it. A thread is pre-empted because of time slicing. A threads isalive=true once starts and until ends. Name property, Thread.currentthread is the currently executing thread.

Join and Sleep. A thread is "blocked" when waiting. Can use threadstate property.

```
Static void Main()
{
Thread t = new Thread(Go);
```

```
T.Start();
T.Join();

Console.writeline("Thread t has ended!");
}
Static void Go(){ for(int i= 0; i < 1000; i++) Console.Write("y"); }

Thread.Sleep(timespan.fromhours(1));
Thread.Sleep(500);
```

Local vs Shared state.

```
Static void Main()
{
New Thread(Go).Start();
Go();
}
Static void Go()
{
For(int cycles = 0; cycles< 5; cycles++) Console.Write('?');
}
Class threadtest
{
Bool _done;
Static void Main()
{
Threadtest tt= new threadtest();
New Thread(tt.Go).Start();
Tt.Go();
}
Void Go()
{
If(!_done) { _done = true; Console.writeline("Done");}
}
}

Class threadtest
{
Static bool _done;
Static void Main()
{
New Thread(tt.Go).Start();
Go();
}
Static void Go()
{
If(!_done) { _done = true; Console.writeline("Done");}
}
}

Class threadtest
{
Static void Main()
{
Bool done= false;
Threadstart  action= () =>
{
If(!Done) { done= true; Console.writeline("Done"); }
```

```
};
New Thread(action).Start();
Action();
}
}
```

Let's discuss thread safety. In this code, we can output Done twice. And the odds increase if statements are reversed.

```
Class threadsafe
{
Static bool _done;
Static readonly object_locker = new object();
Static void Main()
{
New Thread(Go).Start();
Go();
}
Static void Go()
{
Lock (_locker)
{
If(!_done){ Console.writeline("Done"); _done = true; }
}
}
}
```

Passing Data to Threads. Use lambda expression (easiest) or use object param.

```
Static void Main()
{
Thread t = new Thread(() => Print("Hello from t!"));
T.Start();
}
Static void Print(string message) { Console.writeline(message);}

New Thread (()=>
{
Console.writeline ("I'm running on another thread!");
Console.writeline ("This is so easy!");
}).Start()

Static void Main()
{
Thread t = new Thread(Print);
T.Start("Hello from t!");
}
Static void Print(object  messageobj)
{
String message = (string)messageobj;
Console.writeline(message);
}

Public delegate void threadstart();
Public delegate void  parameterizedthreadstart(object obj);
```

Lambda expressions and captured vars

```
For(int i= 0; i < 10; i++)
New Thread (()=> Console.Write(i)).Start();
Typical Output:
0223557799

For(int i= 0; i < 10; i++)
{
Int temp = i;
New Thread (()=> Console.Write(temp)).Start();
}
```

Exceptions

```
Public static void Main()
{
Try
{
New Thread(Go).Start();
}
Catch (Exception ex)
{
Console.writeline("Exception!");
}
}
Static void Go(){ throw null;}

Public static void Main()
{
New Thread (Go).Start();
}
Static void Go()
{
Try
{
//...
Throw null;
//...
}
Catch (Exception ex)
{
//...
}
}
```

There are isbackground and Priority properties. Signaling between threads using a manualresetevent (simplest).

Waitone and Set. Reset closes the signal.

```
Var signal = new manualresetevent(false);
New Thread (()=>
{
Console.writeline ("Waiting for signal...");
Signal.waitone();
Signal.Dispose();
Console.writeline ("Got signal!");
}).Start();
Thread.Sleep(2000);
Signal.Set();
```

Chapter 33

Lecture 33

Long rnning operations make application unresponsive. Because main thread used for rendering UI and responding to events. Start up worker thread and update UI when finished. But UI updation usually possible only on UI thread. So, forward the request to UI thread.(or marshal it). Low level way is to call begininvoke or Invoke on the elements Dispatcher object. It takes a delegate and queues it on the UI thread. Invoke does same but then blocks until it is done. So you can return a value. But if you dont need begininvoke is better.

```
Partial class mywindow: Window
{
Public mywindow()
{
Initializecomponent();
New Thread(Work).Start();
}
Void Work()
{
Thread.Sleep(5000);
Updatemessage("The answer");
}
Void updatemessage(string message)
{
Action action= () => txtmessage.Text = message;
Dispatcher.begininvoke(action);
}
}
```

Synchronizationcontext can be used for generalized thread marshaling.

```
Partial class mywindow: Window
{
Synchronizationcontext _uisynccontext;
Public mywindow()
{
Initializecomponent();
_uisynccontext = synchronizationcontext.Current;
New Thread(Work).Start();
}
Void Work()
{
Thread.Sleep(5000);
Updatemessage("The answer");
}
Void updatemessage(string message)
{
_uisynccontext.Post(_ => txtmessage.Text = message);
}
}
```

Threadpool save time of thread creation. Cant name thread and difficult debugging. Always background. Blocking can degrade performance. Thread.currentthread.isthreadpoolthread property. Threadpool creates or reduces real threads using a hillclimbing algo to maximize cpu usage and reduce slicing.

```
Threadpool.queueuserworkitem(notused => Console.writeline ("Hello"));
Task.Run(()=> Console.writeline ("Hello from the thread pool"));
```

Let's discuss Tasks. No easy way to get return value from a thread. We can join and use shared data. Exceptions difficult too. Cant tell to start something else when finished. Task: a higher level abstraction. Tasks can be chained using continuations. Can use threadpool. With taskcompletionsource callback approach.

Starting a task is like creating a thread. Except started right away (hot) and in a thread pool. Task.wait is like Join. Task has generic subclass Task¡treturn¿. Task.Result blocks.

```
Task.Run(()=> Console.writeline ("Foo"));
New Thread(()=> Console.writeline ("Foo")).Start();

Task task= Task.Run(()=>
{
Thread.Sleep(2000);
Console.writeline ("Foo");
});
Console.writeline (task.iscompleted);
Task.Wait();

Task<int> task = Task.Run(()=>{ Console.writeline("Foo"); return3;});
// ...
```

Tasks propagate exception to whoever calls wait or accesses Result

```
Int result = task.Result;
Console.writeline (result);
Task task= Task.Run(()=>{ throw null;});
Try
{
Task.Wait();
}
Catch(aggregateexception aex)
{
If(aex.innerexception is nullreferenceexception)
Console.writeline ("Null!");
Else
Throw;
}
```

Continuations says when finished continue with something else. Two ways.

```
Task<int> primenumbertask= Task.Run(()=>
Enumerable.Range(2,3000000).Count(n=>
Enumerable.Range(2,(int)Math.Sqrt(n)-1).All(i=>n%i>0)));

Primenumbertask.continuewith(antecedent=>
{
Int result = antecedent.Result;
Console.writeline (result);
});
Var awaiter= primenumbertask.getawaiter();
Awaiter.oncompleted(()=>
{
```

```
Int result = awaiter.getresult();
Console.writeline (result);
});
```

If synchronization context present, run on UI thread task completion source. Any operation that start and finishes some time later. Slave task you manually drive. Mark finish. Ideal for i/o bound work. All benefits of tasks (returns, exceptions, continuations) without blocking thread. Create a task which you can wait and attach continuations. Controlled by these operations.

```
Public class taskcompletionsource< tresult>
{
Public void setresult (tresult result);
Public void setexception (Exception  exception);
Public void setcanceled();
Public bool trysetresult (tresult result);
Public bool trysetexception (Exception  exception);
Public bool trysetcanceled();
//...
}
```

Calling any function signals the task. Call exactly once. Following example prints 42 after 5s. Calling this is equivalent to creating long running task. Let's do the same without a thread. Then attaching continuation.

```
Var tcs = new taskcompletionsource<int>();
New Thread (()=>{
Thread.Sleep(5000); tcs.setresult (42);
}).Start();
Task<int> task = tcs.Task;
Console.writeline (task.Result)

Task<tresult> Run<tresult>(Func<tresult> function)
{
Var tcs = new taskcompletionsource<tresult>();
New Thread (()=>
{
Try{ tcs.setresult(function());}
Catch (Exception ex) { tcs.setexception (ex);}
}).Start();
Return tcs.Task;
}
//...
Task<int> task = Run (()=>{ Thread.Sleep (5000); return 42;});

Task<int> getanswertolife()
{
Var tcs = new taskcompletionsource<int>();
Var timer= new System.Timers.Timer(5000) { autoreset = false };
Timer.Elapsed+= delegate { timer.Dispose(); tcs.setresult(42); };
Timer.Start();
Return tcs.Task;
}
```

Thread used only when continuation running. Task.delay available.

```
Var awaiter = getanswertolife().getawaiter();
Awaiter.oncompleted(()=> Console.writeline (awaiter.getresult()));

Task Delay(int milliseconds)
```

```
{
Var tcs = new taskcompletionsource<object>();
Var timer= new System.Timers.Timer(milliseconds){ autoreset = false };
Timer.Elapsed+= delegate { timer.Dispose(); tcs.setresult(null);};
Timer.Start();
Return tcs.Task;
}
Delay(5000).getawaiter().oncompleted (()=> Console.writeline (42));
Delay(5000).continuewith (ant => Console.writeline (42));
Task.Delay(5000).getawaiter().oncompleted (()=> Console.writeline (42));
Task.Delay(5000).continuewith(ant => Console.writeline (42));
```

Chapter 34

Lecture 34

In last lecture we discussed Threadpool to avoid thread creation time. Task concept one thread many tasks. Continuations / Exceptions / Returns. Task Completion Source. Discussed Task.Delay.

Synchronous vs. Asynchronous. Async typically return quickly. Called non-blocking. Thread.start, task.run, at-taching continuations. For io bound we can usually work without thread. For cpu bound we can start and return task. Cpu bound async tasks.

```
Int getprimescount(int start, int count)
{
Return
Parallelenumerable.Range(start, count).Count(n =>
Enumerable.Range(2,(int)Math.Sqrt(n) - 1).All(i => n % i> 0));
}

Void displayprimecounts()
{
For(int i= 0; i < 10; i++)
Console.writeline(getprimescount(i * 1000000 + 2, 1000000) +
" primes between " + (i * 1000000) + " and "+ ((i+ 1) * 1000000 - 1));
Console.writeline("Done!");
}
Output:
78498 primes between 0 and999999
70435 primes between 1000000 and1999999
67883 primes between 2000000 and2999999
66330 primes between 3000000 and3999999
65367 primes between 4000000 and4999999
64336 primes between 5000000 and5999999
63799 primes between 6000000 and6999999
63129 primes between 7000000 and7999999
62712 primes between 8000000 and8999999
62090 primes between 9000000 and9999999
```

How to make async. Course grained.

```
Task.Run (()=> displayprimecounts());

Task<int> getprimescountasync(int start, int count)
{
Return Task.Run(() =>
Parallelenumerable.Range(start, count).Count(n =>
Enumerable.Range(2,(int)Math.Sqrt(n) - 1).All(i => n % i> 0)));
}
For(int i= 0; i < 10; i++)
{
Var awaiter = getprimescountasync (i*1000000 + 2, 1000000).getawaiter();
Awaiter.oncompleted(()=>
Console.writeline (awaiter.getresult() + " primes between... "));
}
Console.writeline ("Done");
```

Back to sequential. What if we want displayprimecounts async as well.

```
Void displayprimecounts()
{
Displayprimecountsfrom(0);
}
Void displayprimecountsfrom(int i)
{
Var awaiter = getprimescountasync(i * 1000000 + 2, 1000000).getawaiter();
Awaiter.oncompleted(()=>
{
Console.writeline(awaiter.getresult() + " primes between...");
If(i++ < 10)  displayprimecountsfrom(i);
Else Console.writeline("Done");
});
}
```

Simple but lot of code.

```
Task  displayprimecountsasync()
{
Var machine = new primesstatemachine();
Machine.displayprimecountsfrom (0);
Return machine.Task;
}
Class  primesstatemachine
{
Taskcompletionsource <object> _tcs = new taskcompletionsource<object>();
Public Task Task{ get{ return _tcs.Task; } }
Public void displayprimecountsfrom (int i)
{
Var awaiter = getprimescountasync (i*1000000+2,1000000).getawaiter();
Awaiter.oncompleted(()=>
{
Console.writeline (awaiter.getresult());
If(i++ < 10)  displayprimecountsfrom (i);
Else { Console.writeline("Done"); _tcs.setresult (null); }
});
}
}
```

Its pretty simple in C# 5.0.

```
Async Task displayprimecounts()
{
For(int i= 0; i < 10; i++)
Console.writeline(await  getprimescountasync(i * 1000000 + 2, 1000000) +
" primes between " + (i * 1000000) + " and "+ ((i+ 1) * 1000000 - 1));
Console.writeline("Done!");
}
```

C# 5.0 introduces async and await keywords. Eliminates plumbing for async code. As simple as sync code. “await” simplifies attaching continuations. E.g. Also code to handle if sync completion. And some other details.

```
Var result= await expression;
Statement(s);
Var awaiter = expression.getawaiter();
```

```
Awaiter.oncompleted(()=>
{
Var result= awaiter.getresult();
Statement(s);
);
```

Returning to example. We can call it with await and use “async” keyword so compiler treats await specially. Async can be applied to methods returning void or Task or Task¡tresult¿.

```
Task<int> getprimescountasync(int start, int count)
{
Return Task.Run(() =>
Parallelenumerable.Range(start, count).Count(n =>
Enumerable.Range(2,(int)Math.Sqrt(n) - 1).All(i => n % i> 0)));
}
Int result = await  getprimescountasync (2,1000000);
Console.writeline (result);
Async void displayprimescount()
{
Int result = await  getprimescountasync(2, 1000000);
Console.writeline(result);
}
```

Async doesnt change method so no need to list in interface. Can add async when implementing. Methods with async are called asynchronous. When await. Method returns to caller. But before returning a continuation attached to the task. If exception rethrown, otherwise return value assigned to await statement. Let’s see the expansion again. Await expression returns int because Task¡int¿ also possible to wait on void Task.

```
Void displayprimescount()
{
Var awaiter  = getprimescountasync(2,
Awaiter.oncompleted(()  =>
{
Int result  = awaiter.getresult();
Console.writeline(result);
});
}
Await Task.Delay (5000);
(1000000).getawaiter();
Console.writeline ("Five seconds passed!");
```

Capturing local state in await. Real power of await. When execution resumes in continuation, local vars have the same values compiler translates into state machines. A general method. If we do ourself we may be doing the complex code we showed before. If sync context, same thread e.g. UI. Otherwise whatever free thread Let’s write a UI program that remains responsive with a cpu bound task. Let’s start with the sync version.

```
Async void displayprimecounts()
{
For(int i  =0; i  < 10; i++)
Console.writeline(await  getprimescountasync(i
}
```

Synchronous version. Unresponsive. 2step in making async.

```
Class testui  : Window
{
Button  _button  = new Button  { Content  = "Go"  };
Textblock  _results  = new textblock();
Public testui()
{
    Var panel = new stackpanel();
    Panel.Children.Add(_button);
    Panel.Children.Add(_results);
    Content = panel;
    _button.Click += (sender, args) => Go();
}
Void Go()
{
For(int i= 1; i < 5; i++)
    _results.Text+= getprimescount(i*1000000,1000000) +" primes between "+(i*1000000)+ "
and "+((i+1)*1000000-1) +Environment.newline;
}
Int getprimescount(int start, int count)
{
    Return  parallelenumerable.Range(start, count).Count(n => Enumerable.Range(2,
 (int)Math.Sqrt(n) - 1).All(i => n % i> 0));
}
}
```

Asynchronous version. Simplicity. Call async functions. And await them. Go leases time on UI thread. Task.Run is
The real parallel.

```
Task<int> getprimescountasync(int start, int count)
{
Return Task.Run (()=>
Parallelenumerable.Range (start, count).Count (n =>
Enumerable.Range(2,(int)Math.Sqrt(n) - 1).All(i => n % i> 0)));
}

Async void Go()
{
_button.isenabled = false;
For(int i= 1; i < 5; i++)
    _results.Text+= await  getprimescountasync(i*1000000,1000000)+" primes between "+(i
 *1000000)+ " and "+((i+1)*1000000-1)+Environment.newline;
_button.isenabled = true;
}
```

Simple but have to handle re-entrancy. And real parallel code does not use shared state. Another example. Download web pages and sum their lengths. System.Net.webclient.downloaddatataskasync method. Returns Task¡byte[]¿ so await returns byte[].

```
Async void Go()
{
_button.isenabled = false;
String[] urls= "www.albahari.com www.oreilly.com www.linqpad.net".Split();
Int totallength = 0;
Try
{
Foreach (string url in urls)
```

```
{
Var uri = new Uri("http://" + url);
Byte[] data = await new webclient().downloaddatataskasync(uri);
_results.Text+= "Length of "+ url + " is "+ data.Length +Environment.newline;
Totallength += data.Length;
}
_results.Text+= "Total length: "+ totallength;
}
Catch(webexception ex)
{
_results.Text+= "Error: "+ ex.Message;
}
Finally { _button.isenabled = true; }
}
```

Mirrors how we would write it sync. Even though control returns to caller after first await, the finally blocks is not run until method is logically completed. Let's see underneath on the message loop.

```
While(!Thisapplication.Ended)
{
Wait for something to appear in message queue
Got something: what kind of message is it?
Keyboard/mouse message -> fire an event handler
User begininvoke/Invoke message -> execute  delegate
}
```

Event handler run via this loop. Go runs as far as await and then returns to this loop. Compilers expansion of await ensures a continuation is setup. Because we awaited on a UI thread, the continuation is posted on the sync context, which ensures it runs via message loop. Go keeps running pseudo-concurrently with UI thread. True concurrency occurs while downloaddatataskasync is running.vs. Course grained where the loop itself on worker thread. Gets difficult when progress reporting.

```
_button.Click+=(sender, args) =>
{
_button.isenabled = false;
Task.Run(()=> Go());
};

Void Go()
{
For(int i= 1; i < 5; i++)
{
Int result = getprimescount(i * 1000000, 1000000);
Dispatcher.begininvoke(new Action(() =>
_results.Text+= result+ " primes between " + (i * 1000000) +
" and " + ((i+ 1) * 1000000 - 1)+ Environment.newline));
}
Dispatcher.begininvoke(new Action(() => _button.isenabled = true));
}
```

Chapter 35

Lecture 35

Let's revise the downloading code from last lecture.

```
Async void Go()
{
_button.isenabled = false;
String[] urls= "www.albahari.com www.oreilly.com www.linqpad.net".Split();
Int totallength = 0;
Try
{
Foreach (string url in urls)
{
Var uri = new Uri("http://" + url);
Byte[] data = await new webclient().downloaddatataskasync(uri);
_results.Text+= "Length of "+ url + " is " + data.Length+
Environment.newline;
Totallength += data.Length;
}
_results.Text+= "Total length: "+ totallength;
}
Catch (webexception ex)
{
_results.Text+= "Error: "+ ex.Message;
}
Finally { _button.isenabled = true; }
}
```

We can return a Task from void function without explicitly return it. Enables async call chains. Compiler indirectly uses taskcompletionsource to implement methods returning Tasks. We can expand printanswertolife like this.
Nuances aside.

```
Async Task printanswertolife()
{
Await Task.Delay(5000);
Int answer = 21 * 2;
Console.writeline(answer);
}
Async Task Go()
{
Await  printanswertolife();
Console.writeline("Done");
}
```

Whenever task finishes, execution returns to whoever awaited for the task. You can return Task¡tresult¿ if the method returns tresult. Internally that results in TCS signaled with a value.

```
Task printanswertolife()
{
Var tcs = new taskcompletionsource<object>();
Var awaiter = Task.Delay(5000).getawaiter();
Awaiter.oncompleted(()=>
```

```
{
Try
{
Awaiter.getresult();
Int answer = 21 * 2;
Console.writeline(answer);
Tcs.setresult(null);
}
Catch (Exception ex) { tcs.setexception(ex); }
});
Return tcs.Task;
}


Async Task <int> getanswertolife()
{
Await Task.Delay(5000);
Int answer = 21 * 2;
Return answer;
}
```

Let's see the async version vs. The sync version.

```
Async Task Go()
{
Await  printanswertolife();
Console.writeline("Done");
}
Async Task printanswertolife()
{
Int answer =
Await  getanswertolife();
Console.writeline(answer);
}
Async Task <int> getanswertolife()
{
Await Task.Delay(5000);
Int answer = 21 * 2;
Return answer;
}
Void Go()
{
Printanswertolife();
Console.writeline("Done");
}
Void printanswertolife()
{
Int answer = getanswertolife();
Console.writeline(answer);
}
Int getanswertolife()
{
Thread.Sleep(5000);
Int answer = 21 * 2;
Return answer;
}
```

Same ease of programming as sync. Intentional. So three steps. Write sync. Use async and await. Return Task and Task¡tresult¿ so awaitable. This means only Task.Run for real parallel cpu task and TCS for real parallel IO task. Rest of TCS are taken care of by compiler.

Async call graph exec. Brief sync exec on thread calling Go. Everyone await and returns. When Delay fires a thread. Remaining statements run. Eventually Gos task is marked completed.

```
Async Task Go()
{
Var task = printanswertolife();
Await task;
Console.writeline("Done");
}
Async Task printanswertolife()
{
Var task = getanswertolife();
Int answer = await task;
Console.writeline(answer);
}
Async Task <int> getanswertolife()
{
Var task = Task.Delay(5000);
Await task;
Int answer = 21 * 2;
Return answer;
}
```

Let's discuss Parallelism. Go is not awaited, allowed and required here. So Let's see how to run two tasks in parallel and then await them. True concurrency at bottom level operations. If a sync context, only pseudo-concurrency. So only switched on await. That means we can increment shared var without locking. But cant assume same value before and after await.

```
Var task1= printanswertolife();
Var task2= printanswertolife();
Await task1;
Await task2;
Async Task <int> getanswertolife()
{
    _x++;
    Await Task.Delay(5000);
    Return 21 * 2;
}
```

Async lambda expr.

```
Func<Task> unnamed = async () =>
{
Await Task.Delay(1000);
Console.writeline("Foo");
};

Await namedmethod();
Await unnamed();
Mybutton.Click += async(sender, args) =>
```

```
{
Await Task.Delay(1000);
Mybutton.Content= "Done";
};
Mybutton.Click += buttonhandler;

Async void buttonhandler(object sender, eventargs args)
{
Await Task.Delay(1000);
Mybutton.Content= "Done";
};

Func<Task<int>> unnamed = async ()=>
{
Await Task.Delay(1000);
Return123;
};
Int answer = await unnamed();
```

Cancellation. Cancel is on cancellationtokensource. Most have builtin cancellation support.

```
Class  cancellationtoken
{
Public bool  iscancellationrequested { get; private set;}
Public void Cancel() { iscancellationrequested= true; }
Public void  throwifcancellationrequested()
{
If(iscancellationrequested)
Throw new operationcanceledexception();
}
}
Async Task Foo(cancellationtoken  cancellationtoken)
{
For(int i= 0; i < 10; i++)
{
Console.writeline(i);
Await Task.Delay(1000);
Cancellationtoken.throwifcancellationrequested();
}
}
Async Task Foo(cancellationtoken  cancellationtoken)
{
For(int i= 0; i < 10; i++)
{
Console.writeline(i);
Await Task.Delay(1000, cancellationtoken);
}
}
Var cancelsource= new cancellationtokensource();
Task.Delay (5000).continuewith(ant => cancelsource.Cancel());
//...

Var cancelsource= new cancellationtokensource(5000);
Try{ await Foo (cancelsource.Token); }
Catch(operationcanceledexception ex)
{ Console.writeline("Cancelled"); }
```

Can even use cancellation with Task.Wait. I.e. Sync methods but have to call cancel from another task. Infact you can give a time interval after which auto cancelled. Useful for timeouts.

Progress reporting. Thread safety issues. Iprogress and progress. Progress ctor.

```
Async Task Foo(Action<int> onprogresspercentchanged)
{
Return Task.Run(() =>
{
For(int i= 0; i < 1000; i++)
{
If(i % 10 == 0)  onprogresspercentchanged(i / 10);
// Do something compute-bound...
}
});
}
Action<int> progress = i=> Console.writeline (i+ "%");
Await Foo(progress);

Var progress= new Progress<int>(i => Console.writeline (i+ " %"));
Await Foo(progress);

Public interface iprogress<in T>
{
Void Report(T value);
}

Task Foo (iprogress<int>onprogresspercentchanged)
{
Return Task.Run (()=>
{
For(int i= 0; i < 1000; i++)
{
If(i % 10 == 0)
Onprogresspercentchanged.Report(i / 10);
// Do something compute-bound...
}
});
}
```

Task-based Async Pattern (TAP): A TAP method Returns a “hot” (running) Taskor Task¡tresult¿. Has an “Async” suffix (except for special cases such as task combinators). Is overloaded to accept a cancellation token and/or iprogress¡T¿if it supports cancellation and/or progress reporting. Returns quickly to the caller (has only a small initial synchronous phase). Does not tie up a thread if I/O-bound.

Task Combinators.

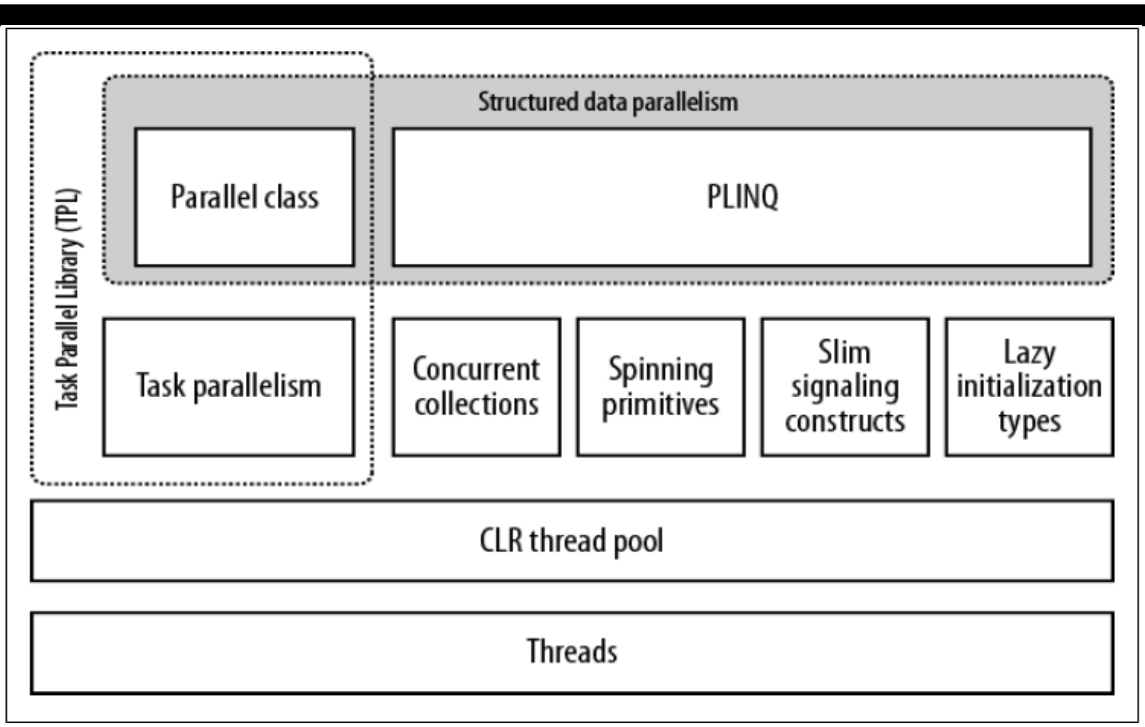
```
Task<int> winningtask  = await Task.whenany
Console.writeline  ("Done");
Console.writeline  (winningtask.Result); 4
(Delay1(), Delay2(), Delay3());
Int answer = await await Task.whenany(Delay1(), Delay2(), Delay3());
Task <string> task= someasyncfunc();
Task winner = await (Task.whenany(someoperation, Task.Delay(5000)));
If(winner != task) throw new timeoutexception();
String result= await task;
```

Chapter 36

Lecture 36

We can imlement timeout with whenany task combinator. On the other hand whenall waits for all. Difference is that should task1 fault, we never get to.

```
Int answer = await await Task.whenany(Delay1(), Delay2(), Delay3());
Task<string> task = someasyncfunc();
Task winner = await (Task.whenany(someoperation, Task.Delay(5000)));
If(winner != task) throw new timeoutexception();
String result= await task;
Await Task.whenall(Delay1(), Delay2(), Delay3());

Task task1 = Delay1(), task2 = Delay2(), task3 = Delay3();
Await task1; await task2; await task3;

Task task1 = Task.Run (()=>{ throw null;} );
Task task2 = Task.Run (()=>{ throw null;} );
Task all = Task.whenall (task1, task2);
Try{ await all;}
Catch
{
Console.writeline (all.Exception.innerexceptions.Count);
}
Task <int> task1 = Task.Run(()=>1);
Task <int> task2 = Task.Run(()=>2);
Int[] results= await Task.whenall(task1, task2);
Async Task<int> gettotalsize(string[] uris)
{
Ienumerable<Task<byte[]>> downloadtasks= uris.Select(uri =>
New(webclient).downloaddatataskasync(uri));
Byte[][] contents = await Task.whenall(downloadtasks);
Return contents.Sum(c => c.Length);
}
Async Task<int> gettotalsize(string[] uris)
{
Ienumerable<Task<int>> downloadtasks = uris.Select(async uri =>
(await new webclient().downloaddatataskasync(uri)).Length);
Int[] contentlengths = await Task.whenall (downloadtasks);
Return  contentlengths.Sum();
}
```



Task Parallel Library exploit multicore for real parallel tasks. 3 steps partition into small chunks, process, collate the results in thread-safe manner. Lock contention and lot of cumbersome code. Data parallelism vs. Task parallelism. Data parallelism easier and scales well. It also structured. Same place in code where parallelism starts and ends.

Concurrent collections also useful when you want a thread-safe collection three static methods in the Parallel class Parallel.Invoke is not a shortcut for creating many threads. Actually batches to use processors efficiently. Combining on the user. Following is thread-unsafe. Locking may make slow. Concurrentbag.

```
Public static void Invoke(params Action[] actions);
Parallel.Invoke (
() => new webclient().downloadfile ("http://www.linqpad.net", "lp.html"),
() => new webclient().downloadfile ("http://www.jaoo.dk", "jaoo.html"));
Var data = new List<string>();
Parallel.Invoke (
() => data.Add(new webclient().downloadstring("http://www.foo.com")),
() => data.Add(new webclient().downloadstring("http://www.far.com")));
Public static  parallelloopresult For(
Int frominclusive, int toexclusive, Action<int> body)
Public static  parallelloopresult foreach<tsource> (
Ienumerable<tsource> source, Action<tsource> body)
For(int i= 0; i < 100; i++)
Foo(i);
Parallel.For(0, 100, i=> Foo(i));
Parallel.For(0, 100, Foo);
Foreach (char c in "Hello, world")
Foo(c);
Parallel.foreach("Hello, world", Foo);
Var keypairs= new string[6];
Parallel.For (0, keypairs.Length,
I => keypairs[i] = RSA.Create().toxmlstring(true));
```

Next we discuss using loop counter, breaking out of loops, and using per-thread counters. Loop counters are easy with sequential. Use overloaded version. What is parallel loop state.

```
Int i = 0;
Foreach (char c in "Hello, world")
Console.writeline (c.tostring()+ i++);

Public static  parallelloopresult foreach<tsource> (
Ienumerable<tsource> source,
Action<tsource,parallelloopstate,long> body)

Parallel.foreach("Hello, world",(c, state, i)=>
{ Console.writeline(c.tostring()+ i); });

Public class  parallelloopstate
{
Public void Break();
Public void Stop();
Public bool isexceptional{ get;}
Public bool isstopped{ get;}
Public long? Lowestbreakiteration { get;}
Public bool  shouldexitcurrentiteration{ get;}
20          }
```

A normal break in parallel. Diff between Break and Stop. Break reaches atleast the sequential point. What about local value. Use overload.

```
Foreach (char c in "Hello, world")
If(c ==’,’)
Break;
Else
Console.Write(c);

// OUTPUT: Hello

Parallel.foreach("Hello, world",(c, loopstate)=>
{
If(c ==’,’)
Loopstate.Break();
Else
Console.Write(c);
});

// OUTPUT: Hlloe

Public static  parallelloopresult For<tlocal>(
Int frominclusive,
Int toexclusive,
Func<tlocal> localinit,
Func<int, parallelloopstate, tlocal, tlocal> body,
Action<tlocal> localfinally);

Object locker= new object();
Double total = 0;
Parallel.For(1,10000000,i =>{ lock(locker) total += Math.Sqrt(i); });
```

```
Object locker= new object();
Double  grandtotal = 0;
Parallel.For(1,10000000,() => 0.0,(i, state, localtotal)=>
Localtotal + Math.Sqrt(i),
Localtotal =>
{ lock(locker) grandtotal+= localtotal; }
);
```

Concurrent collections include concurrent stack, concurrent queue, concurrent bag, concurrent dictionary. They are optimized for concurrent. But also useful for thread-safe. Conventional outperform in all but highly concurrent situations. Thread-safe collection doesnt guarantee thread safe code. You enumerate, another modifies, you get com-bination of old and new. No concurrent version of List. Stack queue bag implemented with linked list but less mem efficient as a consequence. E.g. The following code is 3 times slower but not with reads. Also some atomic test and act.like trypop.

Iproducerconsumercollection¡T¿ has no need to lock. Adds tryadd and trytake methods. With stack, most recent, queue oldest, and bag whatever. Concurrentbag is an unordered collection. Its kind of linked list for each thread. Add has almost no contention.

Blockingcollection¡T¿ wait instead of returning false. Wrapper class. Also Let's you limit the size. Called bounded blocking collection. Ctor takes a iproducerconsumer and limit. Default is to make a queue. Can also give multiple collecitons and use addtoany takefromany. Can also takegetconsumingenumerable. Let's make a producer consumer queue.

```
Public class pcqueue : idisposable
{
    Blockingcollection<Action> _taskq = new blockingcollection<Action>();
    Public pcqueue(int workercount)
    {
        For (int i = 0; i < workercount; i++)
            Task.Factory.startnew(Consume);
    }
    Public void Enqueue(Action action) { _taskq.Add(action); }
    Void Consume()
    {
        Foreach (Action action in _taskq.getconsumingenumerable())
            Action();
    }
    Public void Dispose() { _taskq.completeadding(); }
}

Public class pcqueue : idisposable
{
    Blockingcollection<Task> _taskq = new blockingcollection<Task>();
    Public pcqueue (int workercount)
    {
        For (int i = 0; i < workercount; i++)
            Task.Factory.startnew (Consume);
    }
    Public Task Enqueue (Action action, cancellationtoken canceltoken
        = default (cancellationtoken))
    {
        Var task = new Task(action, canceltoken);
        _taskq.Add(task);
        Return task;
    }
```

```
    Public Task<tresult> Enqueue<tresult>(Func<tresult> func,
        Cancellationtoken canceltoken = default (cancellationtoken))
    {
        Var task = new Task<tresult>(func, canceltoken);
        _taskq.Add(task);
        Return task;
    }
    Void Consume()
    {
        Foreach (var task in _taskq.getconsumingenumerable())
            Try
            {
                If (!Task.iscanceled) task.runsynchronously();
            }
            Catch (invalidoperationexception) { }
    }
    Public void Dispose() { _taskq.completeadding(); }
}
```

Chapter 37

Lecture 37

In the last lecture, finished discussion on tasks and multithreading. Important part of event driven and visual pro-gramming. To remain responsive. The task parallel library. Parallel.Invoke, For, Foreach. Loop counter, break, stop, per-thread counter. Concurrent collections, stack, queue, bag, dictionary. Ended with a producer consumer queue with tasks.

Now we'll discuss event driven programming on the web and mobile. Client side programming and event-handling. Server side web programming in other courses.

Static web sites were there initially. Html. Hypertext markup language. Display information and thats it. Today websites reach interactivity of desktop applications. Because of javascript. Animation, interactivity, dynamic visual effects. E.g. Immediately give error msg when wrong data, give total when add to shopping cart, slideshow instead of image list, expand collapse information, popup tooltips. Immediate feedback. No delay like server side. No constant loading reloading. So feels like desktop programs.

E.g. You visited google maps. JS (javascript) in action. Zoom in zoom out. Prev. Map sites used reloading.

Javascript was introduced in 95 by netscape. About as old as web. But mostly hobby things like flies with mouse or messages that move like stock ticker. Many scripts but didnt work in all browsers, even crashed them often. JS has nothing to do with Java, originally named livescript but renamed to associate with the then hot Java. Initial interoperability problems in netscape and IE. Often added incompatible features. MS introduced jscript. Their version of JS for IE. These days mostly handled. Standardization. Some quirks left. Called ecmascript. The official standardization name. Refueled by high profile sites like google using JS in last decade. Now JS even used by some non-web scripting. Even flash actionscript based on it. Or write widgets, phone apps etc.

JS is a prog language. Can be hard for some. Also browser incompatibilities make testing difficult. Jquery is a JS library intended to make JS programming easier. Jquery solves JS complexity and browser incompatibilities. Can do things in single LOC that would take hundreds. Many advanced features come as jquery plugins. Used on millions of websites.

HTML: structural layer and CSS: presentation layer and JS: behavioral layer.

HTML has simple commands called tags. Doctype is of html5 here. Tells browser how to render the page. What standards. Five types of html in use: HTML 4.01 Transitional, HTML 4.01 Strict, XHTML 1.0 Transitional, XHTML 1.0 Strict, and HTML5. All current browsers understand them all. Starting and closing tags like XML. At least three tags. Html root tag, head tag containing title etc, and body tag containing all parts to be rendered in browser window. ¡p¿¡/p¿ is paragraph ¡strong¿¡/strong¿ is emphasis¡a href=”http:...”¿¡/a¿ is a hyperlink. XML attribute and value. Validating html means checking if all tags appropriately closed etc.

```
<!DOCTYPE  html >
<html>
<head>
<meta charset=utf-8>
<title>Hey, I am the title of this web page.</title>
</head >
<body>
```

```
Hey, I am some body text on this web page.
</body >
</html >
```

Originally there was only HTML. CSS is a formatting language. Html only to structure, so ¡h1¿ ¡h2¿ are both headings with diff imp ¡ul¿ is an unordered list. CSS adds design. CSS style is a rule telling web browser how to display an element. E.g. You can make CSS rule to make all ¡h1¿ tags 36 px tall, in courier font, and in orange. CSS can do more powerful stuff, like add border, change margins, and even control exact placement on web page. JS can add/remove/change CSS properties based on input or mouse clicks. Even animate from one property to another. Like yellow to red. Or animate across screen by changing position.

A single CSS style is a rule that tells how to format. Make "this" look like "that". Selector and declaration block. E.g. Selector can be headline, paragraph of text, photo etc. Declaration block can turn text blue, add red border around a paragraph, position the photo at center of page etc. E.g. P color: red; font-size: 1.5em; selector, declaration block has declarations. Each is a property value pair and then ";".

JS Let's a page re-act. Smart web forms. Let's user know when they miss important info, make elements appear or disappear, or move around a webpage. Even load new content from web server without reloading. More engaging and effective web sites.

Client side vs. Server side. Prog lang for the web browser. Alternate is a server prog lang. Php, .net, asp, cold fusion, ruby on rails, etc. They run on web server. Log of intelligence by accessing DB, process CC, send emails. Visitors wait until response comes. Client side lang can re-act immediately. Responsive. Other client side technologies are applet's, silverlight, flash. Often requires a plugin or start slow because of downloading. Sometimes even diff to see if flash or JS. Once yahoo maps was flash. Then re-written. Right click and see if About the Flash Player Ajax brings client-side server-side together. JS talks to server, downloads content, and update webpage. Google maps Let's you move to new areas. JS is a prog lang and can be used on server side. E.g. Node.js supports JS on server-side.

Compiled vs scripted languages (interpreted). JS interpreter is in web browser. Let's write a prog to ask visitor name, get response, and print a welcome msg. Web browser has layout or rendering engine (understanding HTML and CSS) and a JS interpreter. Tell web browser about JS in a ¡script¿¡/script¿ tag. Here is a script in html 4.01 and in html5.

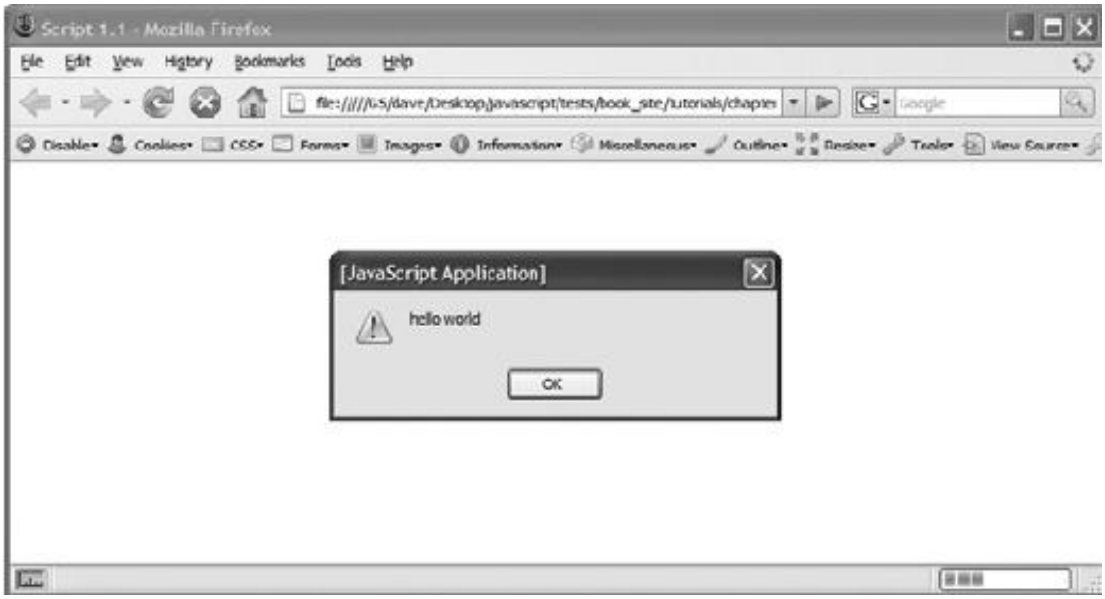
```
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01//EN"
"http://www.w3.org/TR/html4/strict.dtd">
<html>
<head>
<title>My Web Page</title>
<script type="text/javascript">
</script>
</head>
<!Doctype html>
<html>
<head>
<meta charset="UTF-8">
<title>My Web Page</title>
<script>
</script>
</head>
4.01//EN"  "http://www.w3.org/TR/html4/strict.dtd">
```

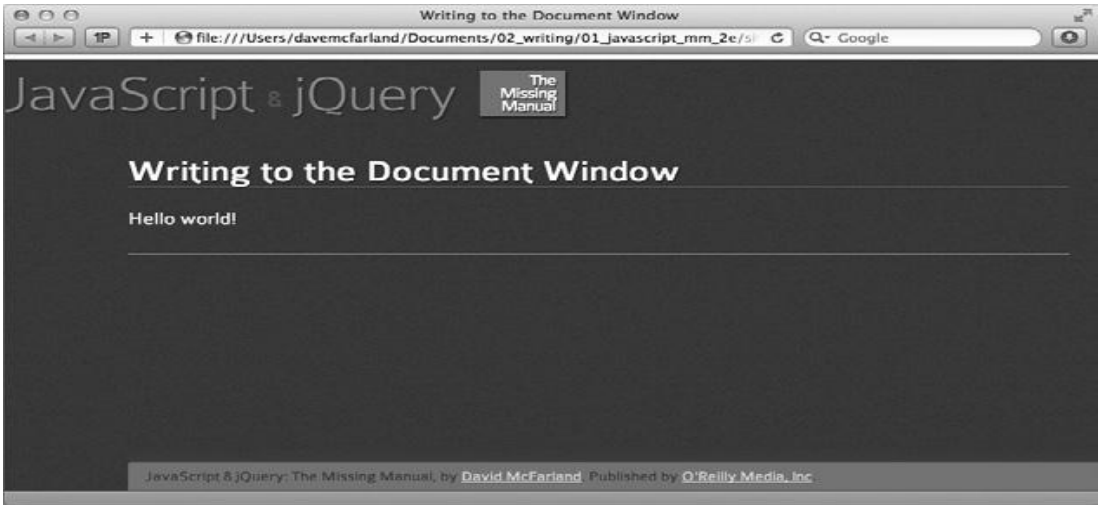
Usually in head section and at one place. But its ok to put it anywhere and in multiple tags. Script can also be placed after ¡/body¿ so script loaded after page displayed can also use external script files. Easy to share.

Separate tags if you want inline code AND “src” attrib for external file. Can and often use multiple external files.

```
<!Doctype html>
<html>
<head>
<meta charset="UTF-8">
<title>My Web Page</title>
<script src="navigation.js"></script>
</head>
<!Doctype html>
<html>
<head>
<meta charset="UTF-8">
<title>My Web Page</title>
<script src="navigation.js"></script>
<script src="slideshow.js"></script>
<script>Alert('hello world!');</script></head>
```

Let’s open hello.html in web browser.





```
<book = javascript \& jquery the Missing Manual page 30>
<script>
Document.write('<p>Hello world!</p>');
</script >
```

Before the web page because of placement. Web page appears after OK pressed. Use document.write and <script> tag anywhere.

```
<link href="../_css/site.css" rel="stylesheet">
<script src="../_js/jquery-1.6.3.min.js"></script>
<script>
$(function (){
$('body').hide().fadein(3000);
});
</script >
```

Lot of basic syntax like C++, C#. Create a variable using "var x", names begin with letter, $, or . Var days = ['Mon', 'Tues', 'Wed', 'Thurs', 'Fri', 'Sat', 'Sun']; alert(days[0]); var playlist = []; var prefs = [1, 223, 'www.oreilly.com', false]; prefs.push('test'); prefs.push('test',test2); arrays grow prefs.unshift('test'); prefs.unshift('test',test2); insert at start shift() gets/removes the first element. Queue using push/shift pop() removes last. Stack alert and prompt functions.

```
Var TAX = .08;
Function  calculatetotal(quantity, price){
Var total= quantity * price * (1+ TAX);
Var formattedtotal = total.tofixed(2);
Return  formattedtotal;
}

Var saletotal= calculatetotal(2,16.95);
Document.write('Total cost is: \$'+ saletotal);
```

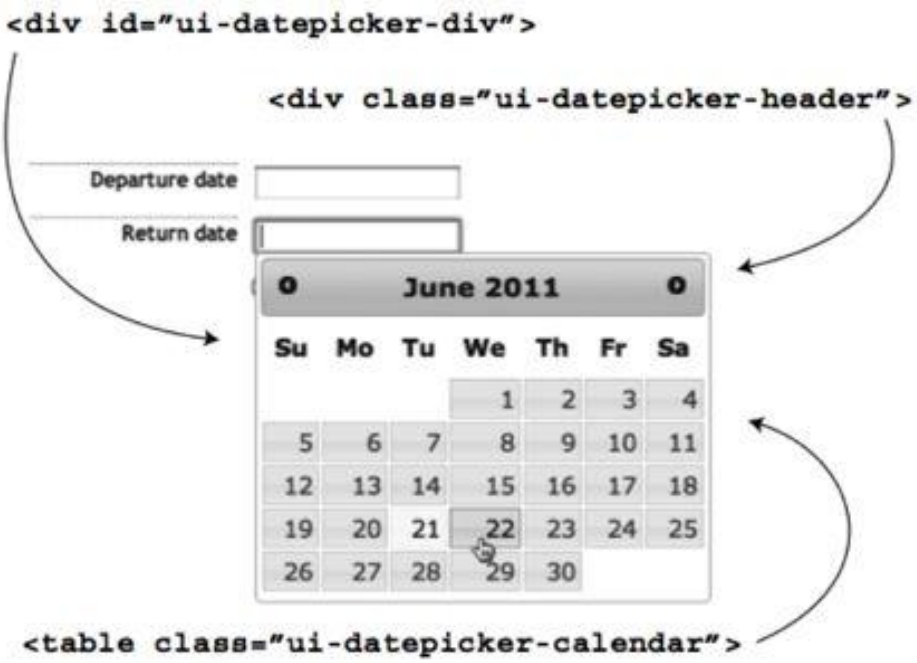
If, while, for, dowhile like C#. Function declarations. No types. Return value.

Let's discuss jquery now. Many JS programs select elements, add new content, hide and show content, modify tag's attributes, determine value of form fields, and react to user actions. The details complicated specially with browser interoperability. Libraries offer a set of functions to make these tasks easy. Only 30k compressed library size. Easy to learn. Used on millions of sites. Free. Dev community. Plugins !!! Where to get jquery.js. Use cdns dont need to host your own. Often cached. Google one is v popular.

```
<script src="http://ajax.aspnetcdn.com/ajax/jquery/jquery-1.6.3.min.js">
</script >

<script src="http://code.jquery.com/jquery-1.6.3.min.js"></script>

<script src="http://ajax.googleapis.com/ajax/libs/jquery/1.6.3/jquery.min.js">
</script >

<script src="js/jquery-1.6.3.min.js"></script>
<script>
\$(document).ready(function (){
// your programming goes here
});
</script >

\$(function (){
```

```
// your programming goes here
});// end ready
```

Second script tag for jquery prog. $(document).ready() waits until HTML of webpage loads and then runs the function. Browser processes in order and to work with elements you want them downloaded. Shortcut for ready function.

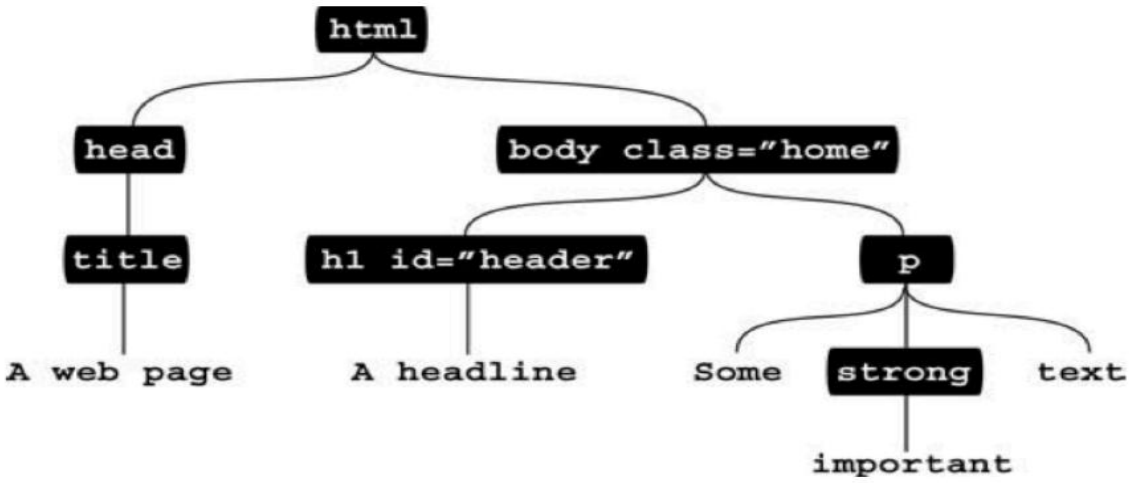


Dynamically changing webpage is key idea. Mouse over, click, detail info. E.g. Date picker is run by JS but itself is made of HTML / CSS. JS just makes presentation interactive. Two steps. Select sth, do sth with it. Do sth can be change prop, add/remove element, extra info, add/remove class attrib, or a combination of these.

Chapter 38

Lecture 38

HTML DOM is much like an XML DOM. JS provides ways to select e.g. Some browsers allow selecting by CSS sometimes not cross browser.




```
Document.getelementbyid('banner');
Document.getelementbytagname('a');
```

To select ¡a¿ tags with class navbutton, you have to select all ¡a¿ tags, then iterate and find the ones with the right class. In jquery $('selector') e.g. $('#banner') is the tag with id banner. $('#banner').html('¡h1¿javascript was here¡/h1¿');. Html is a jquery helper function.
Basic selectors are ID selectors, element selectors, class selectors.

```
<p id="message">Special message</p>
Var messagepara = document.getelementbyid('message');
Var messagepara = $('\#message');

Var linkslist= document.getelementsbytagname('a');
Var linkslist= $('a');

$('.submenu')
$('.submenu').hide();
```

Advanced selectors are

- Descendent selectors $('#navbar a') ,
- Child selectors $('body ¿ p') ,
- Adjacent sibling $('h2 + div') ,
- Attribute selectors $('img[alt]'), $('input[type="text"]'), $('a[href ="mailto:"]'), $('a[href$=".pdf"]'), $('a[href*="missingmanuals.com"]') , form element selectors later. Jquery filters are :even :odd $('.striped tr:even') :first :last :not $('a:not(.navbutton)'); :has $('li:has(a)') — diff from descendent :contains $('a:contains(Click Me!)') :hidden :visible $('div:hidden').show();

Jquery selection. Dont end up with DOM lists. Rather jquery equivalents automatic loops.$('#slideshowimg').hide();

Chaining functions...$('#popup').width(300).height(300);
$('#popup').width(300).height(300).text('Hi!').fadein(100

Let's see jquery functions to add content

Take this example

```
<div id="container">
<div id="errors">
<h2>Errors:</h2>
</div >
</div >
Alert(\$('\#errors').html());
$('#product101').replacewith(<p>Added to cart</p>');
$('\#errors').html('<p>There are four errors in this form</p>');
$('\#errors h2').text('No errors found');
$('\#errors').append('<p>There are four errors in this form</p>');
$('\#errors').prepend('<p>There are four errors in this form</p>');
$('\#username').after('<span class="error">User name required</span>');
$('\#popup').remove();
$('\#product101').replacewith(<p>Added to cart</p>');
$('a[href\^="http://"]').addclass('externallink');
<a href="http://www.oreilly.com/">
<a href="http://www.oreilly.com/" class="externallink">
```

Attributes can be manipulated with addclass removeclass toggleclass. And css. Var bgcolor = $('#main').css('background-color'); $('body').css('font-size', '200$('p.highlight').css('border', '1px solid black'); multiple css props can be changed together.

```
Var basefont = $('body').css('font-size');
Basefont = parseint(basefont,10);
$('body').css('font-size',basefont * 2);

$('#highlighteddiv').css('background-color','#FF0000','border','2px solid #FE0037');
```

For changing html attribute, css and addclass are just shortcuts general purpose attr() and removeattr() var image-File = $('#banner img').attr('src'); $('#banner img').attr('src','images/newimage.png'); $('body').removeattr('bgcolor');. Acting on each element in a selection. When you do want something special. Each() and anonymous function. Use "this" for current element as DOM obj. $(this) for current element as jquery selection.

```
Var imagefile= \$('\#banner img').attr('src');
\$('\#banner img').attr('src', 'images/newimage.png');
\$('body').removeattr('bgcolor');
\$('selector').each(function (){
// code goes in here
});
\$('a[href\^=http://]').each(function (){
Var extlink = \$(this).attr('href');
\$('\#biblist').append('<li>'+ extlink+ '</li >');
});
```

Events. Things happen to webpage. Page loading, mouse move, key press you respond to events. Mouse events: click, dblclick, mousedown, mouseup, mouseover, mouseout, mousemove. Doc/window events: load, resize, scroll, unload. Form events: submit, reset, change, focus, blur. Keyboard: keypress (over n over), keydown, keyup.

Step 1: select elements, step 2: assign an event, step 3: pass function to event.

```
$('\#menu').mouseover(function (){
$('\#submenu').show();
});// end mouseover
```

Ready() vs. Load event

```
\$(function(){
// do something on document ready
});
```

Jquery events. Hover. Toggle is like hover except worked on and off by clicks. Event object is passed to all functions handling events.

```
<script>
\$(document).ready(function (){
\$('html').dblclick(function (){
Alert('ouch');
});// end double click
\$('a').mouseover(function(){
Var message = "<p>You moused over a link</p>";
\$('.main').append(message);
});// end mouseover
\$('\#button').click(function(){
\$(this).val("Stop that!");
});// end click
});// end ready
</script >
\$('\#menu').hover(function (){
\$('\#submenu').show();
}, function (){
\$('\#submenu').hide();
});// end hover
```

Chapter 39

Lecture 39

Event properties. String.fromcharcode(evt.which) evt.preventdefault(); or return false; to stop normal behavior e.g. Links, form submit etc. Remove events $('.tabbutton').unbind('click'); default event bubbling and can stop it evt.stoppropagation(); generic way to bind events (click, mouseover etc. Special) $('#selector').bind('click', mydata, functionname); $('selector').bind('click', functionname); equivalent is $('selector').click(functionname); can bind multiple events

```
\$(document).click(function (evt){
Var xpos = evt.pagex;
Var ypos = evt.pagey;
```

Event property	Description
pageX	The distance (in pixels) of the mouse pointer from the left edge of the browser window.
pageY	The distance (in pixels) of the mouse pointer from the top edge of the browser window.
screenX	The distance (in pixels) of the mouse pointer from the left edge of the monitor.
screenY	The distance (in pixels) of the mouse pointer from the top edge of the monitor.
shiftKey	Is *true* if the shift key is down when the event occurs.
which	Use with the keypress event to determine the numeric code for the key that was pressed (see tip, next).
target	The object that was the "target" of the event—for example, for a *click()* event, the element that was clicked.
data	A jQuery object used with the *bind()* function to pass data to an event handling function (see page 177).

```
Alert('X:' + xpos+ ' Y:' + ypos);
});// end click


\$('selector').bind('click', functionname);
\$('\#selector').bind('click', mydata, functionname);
\$(document).bind('click keypress', function (){
\$('\#lightbox').hide();
});// end bind
\$('\#theelement').bind('click', function (){
// do something  interesting
});// end bind
\$('\#theelement').bind('mouseover', function (){
// do something else interesting
}
// end bind
\$('\#theelement').bind({
```

```
'Click'  : function()  {
// do something  interesting
}, // end click function
'Mouseover'  : function()  {
// do something  interesting
}; // end mouseover  function
});// end bind
```

FAQ example.

```
<script src="../_js/jquery-1.6.3.min.js"></script>
<script>
\$(document).ready(function()  {
\$('.answer').hide();
\$('\#main h2').toggle(
Function()  {
\$(this).next('.answer').fadein();
\$(this).addclass('close');
},
Function()  {
\$(this).next('.answer').fadeout();
\$(this).removeclass('close');
}
); //end toggle
}); 16  </script>
```

Jquery animations. $('element').fadeout('slow');. Fast normal slow or number of ms . Default 200 400 600. Fadein, fadeout, fadetoggle. Slidedown, slideup, slidetoggle.

Login slider example

```
\$(document).ready(function (){
\$('\#open').toggle(
Function (){
\$('\#login form').slidedown(300);
\$(this).addclass('close');
},
Function (){
\$('\#login form').fadeout(600);
\$(this).removeclass('close');
}
);// end toggle
});// end ready
```

Generic animate. Any numeric css prop. Border-left-width becomes borderleftwidth cuz JS doesnt understand hyphen even += -=.

```
\$('\#message').animate(
{
Left:'650px',
Opacity:.5,
Fontsize:'24px'
},
1500
);
\$('\#moveit').click(function(){
\$(this).animate(
{
Left:'+=50px'
},
1000);
});
```

Easing. Linear or swing $('#element').slideup(1000,'linear'); can event pass a function to run when the animation finishes

```
\$('\#element').slideup(1000,'linear');
\$('\#photo').fadein(1000, function(){
\$('\#caption').fadein(1000);
});
\$('\#photo').width(0).height(0).css('opacity', 0);
\$('\#caption').hide();
\$('\#photo').animate(
{
Width:'200px',
Height: '100px',
Opacity:1
},
1000,
Function() {
\$('\#caption').fadein(1000);
}
);// end animate
```

Can chain effects.

$('#photo').fadein(1000).delay(10000).fadeout(250);

Photo gallery example.

```
\$(’\#gallery a’).click(function(evt) {
Evt.preventdefault();
Var imgpath = \$(this).attr(’href’);
Var oldimage = \$(’\#photo img’);
Var newimage = \$(’<img src="’ + imgpath+’">’);
Newimage.hide();
\$(’\#photo’).prepend(newimage);
Newimage.fadein(1000);
Oldimage.fadeout(1000,function(){
\$(this).remove();
});// end fadeout
});// end click
\$(’\#gallery a:first’).click();
```

Forms.

```
<input name="quantity" type="text" id="quantity">
<input name="total" type="text" id="total">
Var unitcost = 9.95;
Var amount = \$(’\#quantity’).val();
Var total= amount * unitcost;
Total = total.tofixed(2);
\$(’\#total’).val(total);
```

Selector	Example	What it does
:input	$(':input')	Selects all input, textarea, select, and button elements. In other words, it selects all form elements.
:text	$(':text')	Selects all text fields.
:password	$(':password')	Selects all password fields.
:radio	$(':radio')	Selects all radio buttons.
:checkbox	$(':checkbox')	Selects all checkboxes.
:submit	$(':submit')	Selects all submit buttons.
:image	$(':image')	Selects all image buttons.
:reset	$(':reset')	Selects all reset buttons.
:button	$(':button')	Selects all fields with type *button*.
:file	$(':file')	Selects all file fields (used for uploading a file).
:hidden	$(':hidden')	Selects all hidden fields.

Submit event.

```
\$(document).ready(function(){
\$(’\#signup’).submit(function(){
If(\$(’\#username’).val() ==’’){
Alert(’Please supply a name in the Name field.’);
```

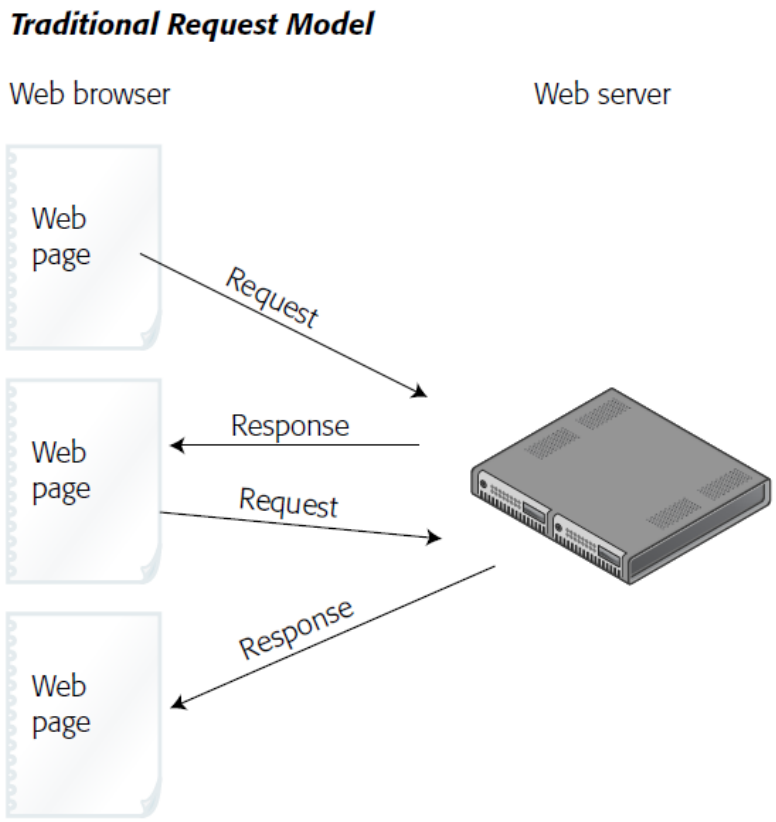
```
Return false;
}
});// end submit()
});// end ready()
```

Focus, blur, click, change (menus). React to choice from a list menu cant do everything at client. Page disappearing and reappearing. Ajax Let's webpage ask for info and update itself when it comes. Asynchronous JS and XML. Term coined in 2005 for interactive sites coming from google. Like google maps, gmail.

Chapter 40

Lecture 40

In Last Lecture, we talk about event object and its properties. Binding and unbinding events. Jquery animations, easing, chaining. FAQ, login slider, photo gallery. Forms and form selectors. Now the real power. Ajax. What can be done. Display NEW html content without reloading the page. Submit form and instantly display results. Login without leaving the page. E.g. Star rating widget. Browsing through database info like you scroll down on fb, twitter. Nothing radical. Except without loading a new page. Can achieve same with HTML and server side prog. Ajax make pages feel more responsive and desktop like.



JS, server-side programming, and web browser, all work together. Web browser: xmlhttprequest object. Makes ajax possible. Talks to web server and get response. JS: sends request, waits for response, process response, updates web page. Web server: receives request and responds as HTML, plain text, XML, JSON. Or application server for more complicated tasks. Need web server for ajax examples.

For talking to web server, we create xmlhttprequest (also called XHR in short) var newxhr = new XML-httprequest(); again browser incompatibilities call open to specify what kind of data and where it will go can GET or POST newxhr.open('GET', 'shop.php?Productid=34'); write a callback function it will remove, add, change el-ements send data newxhr.send(null); GET newxhr.send('q=javascript'); POST receive response callback invoked and XHR receives status, text response, and possibly an XML response

status = 200/304 all ok, 404 file not found,500 internal server error, 403 access forbidden responsetext has text of JSON or HTML, responsexml less commonly used .

The jquery simplifies all steps except that of changing the webpage simplest is load function which loads HTML into an area of web page e.g. Load news in a div from a web-server $('#headlines').load('todays news.html'); can only load from same site... Relative urls possible to add only a part $('#headlines').load('todays news.html #news');.

```
\$('\#headlines').load('todays_news.html');

\$('\#headlines').load('todays_news.html \#news');

\$('\#newslinks a').click(function (){
Var url = \$(this).attr('href');
\$('\#headlines').load(url+ ' \#newsitem');
Return false;
});
\$.get(url, data, callback);
\$.post(url, data, callback);
```

Get() and post(). Need server side to do anything else. Server may not return html e.g. Database records as xml or json. Jquery handles differences of GET and POST. No selector. Stand by themselves.

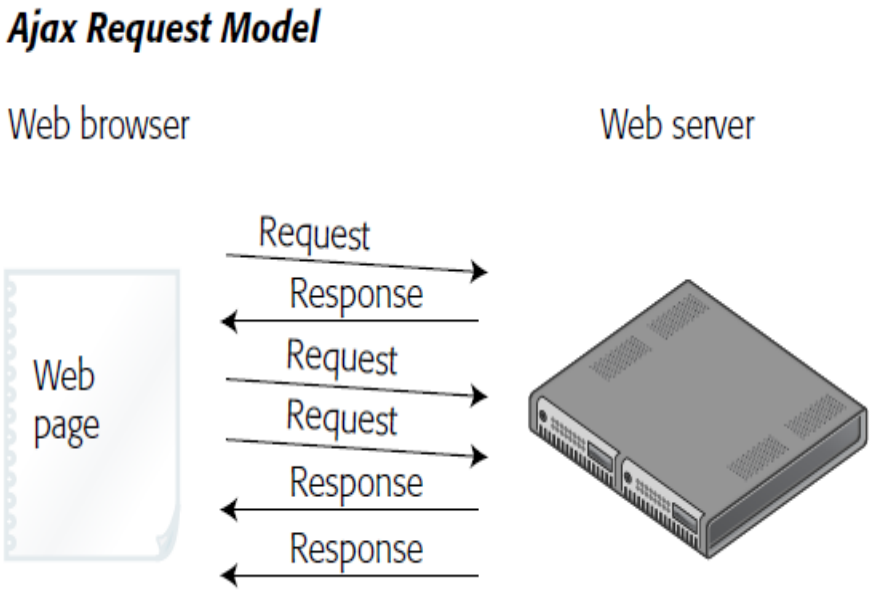
```
$.get('ratemovie.php', 'rating=5');
$.post('ratemovie.php', 'rating=5');
```

Formatting data. Can send a product number, entire form, signup. Format as query string of JS obj literal. URL http://www.chia-vet.com/prod- ucts.php?Prodid=18&sessid=1234. GET has limit. Often thousands of chars. $.get('ratemovie.php','rating=5'); $.post('ratemovie.php','rating=5'); $.post('ratemovie.php','rating=5&user=Bob'); 'favfood=Mac & Cheese' // incorrect 'favfood=Mac%20%26%20Cheese' // properly escaped var querystring = 'fav-Food=' + encodeuricomponent('Mac & Cheese'); $.post('foodchoice.php', querystring); better way is obj literal.

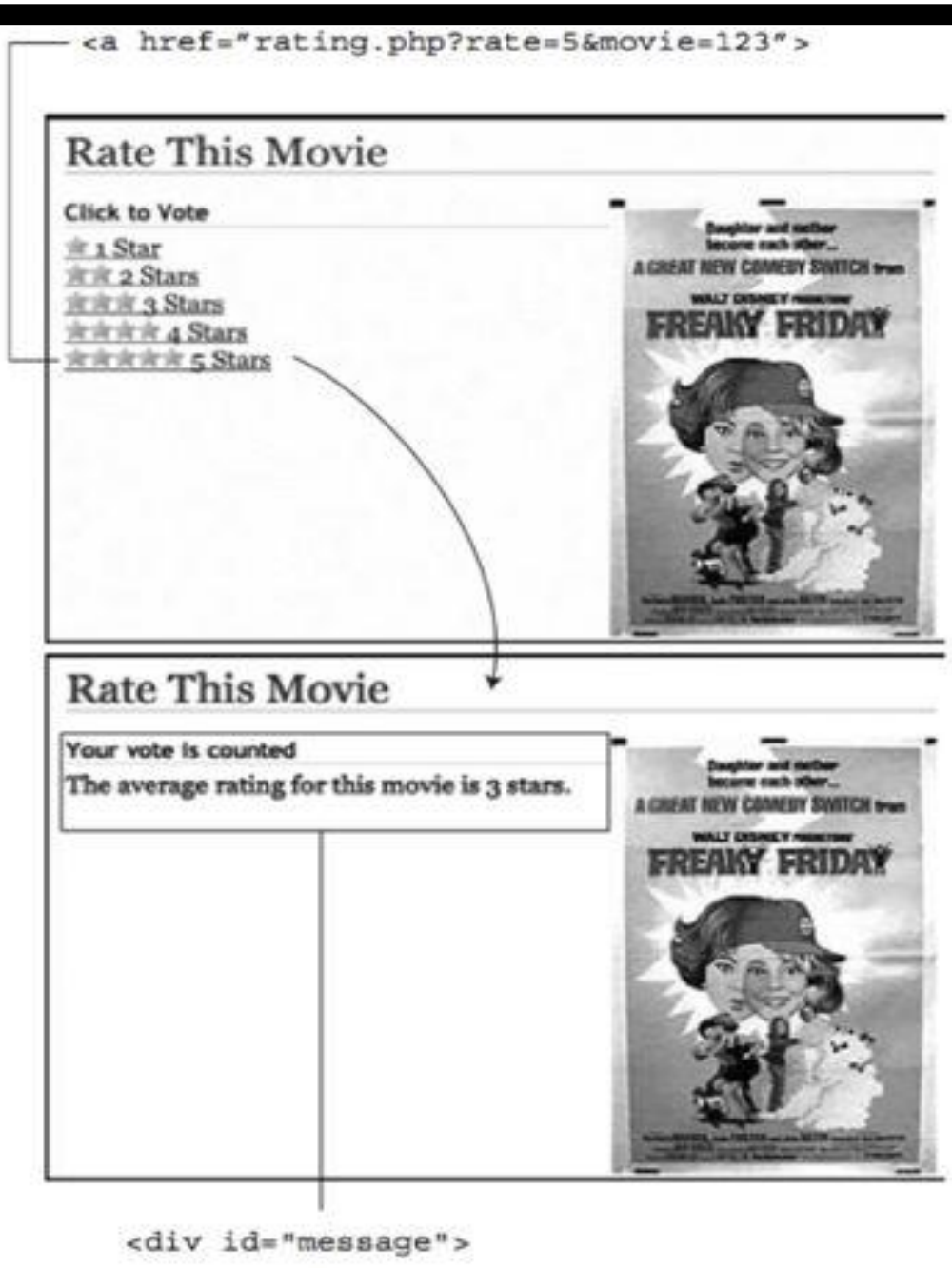
```
{
Name1:'value1',
Name2:'value2'
}

\$.post('rankmovie.php', { rating:5 });

Var data = { rating:5 };

\$.post('rankmovie.php', data);

Var data = {
Rating: 5,
User:'Bob'
}

\$.post('rankmovie.php', data);

Var data = \$.post('rankmovie.php',
{
Rating: 5,
User:'Bob' }
);// end post
```

Serialize using name/value of form elements var formdata = $('#login').serialize(); $.get('login.php',formdata,loginresults); processing data returned. Call back first arg is data. Servers often use XML of JSON. Second arg is string about status success. E.g. Movie rating.

```
Function  processresponse(data, status){
Var newhtml;
Newhtml = '<h2>Your vote is counted</h2>';
Newhtml += '<p>The average rating for this movie is ';
Newhtml += data + '.</p>';
\$('\#message').html(newhtml);
//
}
```




```
\$('\#message a').click(function() {
Var href=\$(this).attr('href');
Var querystring=href.slice(href.indexof('?')+1);
\$.get('rate.php', querystring, processresponse);
Return false;// stop the link
});

Function  processresponse(data) {
Var newhtml;
Newhtml = '<h2>Your vote is counted</h2>';
Newhtml += '<p>The average rating for this movie is ';
Newhtml += data + '.</p>';
\$('\#message').html(newhtml);
}
```



Error handler.

```
\$.get(url, data, successfunction).error(errorfunction);
\$.get('rate.php', querystring, processresponse).error(errorresponse);
Function  errorresponse(){
Var errormsg= "Your vote could not be processed right now.";
Errormsg += "Please try again later.";
\$('\#message').html(errormsg);
}
```

JS format. Method for exchanging data. JSON is JS so its quick n easy for JS. No XML like parsing. JSON is a JS obj literal. (MUST use quotations if names have spaces etc.)

```
{
Firstname: ’Frank’,
Lastname:’Smith’,
Phone:’503-555-1212’
}

{
’Firstname’:’Frank’,
’Lastname’: ’Smith’,
’Phone’: ’503-555-1212’
}
```

Server returns a string formatted like a JSON obj literal. Jquery getjson method. Callback will be a JSON object.

```
Var bday  = {
Person:  ’Raoul’,
Date:  ’10/27/1980’
};
Bday.person  //’Raoul’
Bday.date  //’10/27/1980’
```

Obj literals can be composed of other obj literals

```
Var data  = {
Contact1:  {
Firstname:  ’Frank’,
Lastname:  ’Smith’,
Phone:  ’503-555-1212’
},
Contact2:  {
Firstname:  ’Peggy’,
Lastname:  ’Jones’,
Phone:  ’415-555-5235’
}
}; 13
Data.contact1.firstname
\$.each(JSON, function  (name, value)  { 16  });
```

Like get but data passed to

\$.getjson(’contacts.php’,’limit=2’, processcontacts);

Chapter 41

Lecture 41

Objective-C introduces smalltalk style messaging in C. Early 1980s. Used for next computer. It is simple extension of C. To create an object, send it an alloc message. Nsmutablearray *arrayinstance = [nsmutablearray alloc];. Initialize it. [arrayinstance init];. Can combine. Nested message send. Nsmutablearray *arrayinstance = [[nsmutablearray alloc] init];. Message = [receiver selector arguments]. E.g. Add obj to array. [arrayinstance addobject:anotherobject];.

Another message you can send to mutablearray. Replaceobjectsinrange:withobjectsfromarray:range:. Pairing of labels and arguments a feature of objective-C. In other languages, Arrayinstance.replace object sinrange with objects from array range (anotherarray, anotherrange); In objective-C. [arrayinstance replaceobjectsinrange:arange withobjectsfromar-ray:anotherarray range:anotherrange];.Destroy using [arrayinstance release];. Should also arrayinstance = nil;. Otherwise dangling. But sending message to nil is ok. Nil is like null.

```
Int main (int argc, const char* argv[])
{
Nsautoreleasepool *pool=[[nsautoreleasepool  alloc] init];
Nsmutablearray *items= [[nsmutablearray  alloc] init];
[items addobject:@"One"];
[items addobject:@"Two"];
[items addobject:@"Three"];
[items insertobject:@"Zero" atindex:0];
For(int i= 0;i< [items count]; i++){
Nslog(@"%@",[items  objectatindex:i]);
}
[items release];
Items= nil;
[pool drain];
Return0;
}
```

Nsstring. [items addobject:@”One”]; @ is shortcut for creating nsstring. Nsstring *mystring = @”Hello, World!”;. Int len = [mystring length];. Len = [@”Hello, World!” Length];. Mystring = [[nsstring alloc] initwithstring:@”Hello, World!”];. Len = [mystring length];.Nslog. Format string. Int a = 1; float b = 2.5; char c = ’A’; nslog(@”Integer: Integer: 1 Float: 2.5 Char: A Nsarray and nsmutablearray. Also nsdictionary and nsset. Holds references. Cannot hold primitives and C structures. Can call [array count]. Int numberofobjects = [array count]; [array insertobject:object atindex:numberofobjects]; cant add beyond end. Exception. [array addobject:[nsnull null]]; // otherwise cannot hold nil. Nsstring *object = [array objectatindex:0];

Subclassing. Root class of entire hierarchy. Nsobject. Objective-C keywords start with @. Instance variables.

```
#import <uikit/uikit.h>
@interface  Possession: nsobject
{
}
@end
#import  <Foundation/Foundation.h>
@interface  Possession: nsobject
```

```
{
Nsstring  *possessionname;
Nsstring  *serialnumber;
Int valueindollars;
Nsdate  *datecreated;
}
@end


#import  <Foundation/Foundation.h>
@interface  Possession: nsobject
{
Nsstring  *possessionname;
Nsstring  *serialnumber;
Int valueindollars;
Nsdate  *datecreated;
}
- (void)setpossessionname:(nsstring  *)str;
- (nsstring  *)possessionname;

- (void)setserialnumber:(nsstring  *)str;
- (nsstring  *)serialnumber;
- (void)setvalueindollars:(int)i;
- (int)valueindollars;
- (nsdate  *)datecreated;
@end 17

#import "Possession.h"
@implementation  Possession
// Getter
- (nsstring  *)possessionname
{
// Returna pointer to the object this Possession calls  its possessionname
Return  possessionname;
}
// Setter
-  (void)setpossessionname:(nsstring  *)newpossessionname 29 {
// Change the instance  variable to point at another string,
// this Possession will now call this new string its possessionname
Possessionname= newpossessionname;
}
// Createa new Possession  instance
Possession *p=[[Possession alloc] init];
// Set possessionname to a new nsstring
[p setpossessionname:@"Red Sofa"];
// Get the pointer of the Possession's possessionname
Nsstring *str= [p possessionname];
// Print that object
Nslog(@"%@", str); // This would print "Red Sofa"
```

Getter setters can be made like above example. Let's see instance methods (and overriding).

```
- (nsstring*)description
{
Nsstring *descriptionstring=
[[nsstring alloc] initwithformat:@"\%@(\%@): Worth\$\%d, recorded on\%@",
```

```
Possessionname,
Serialnumber,
Valueindollars,
Datecreated];
Return  descriptionstring;
}
```

Initializers start with init naming convention. Id = any object. Every object has isa pointer to class and thats how methods are called. Like vtable. Self and super. Other initializers.

```
- (id)initwithpossessionname:(nsstring *)name
Valueindollars:(int)value
Serialnumber:(nsstring*)snumber;
- (id)initwithpossessionname:(nsstring *)name
Valueindollars:(int)value
Serialnumber:(nsstring*)snumber
{
// Call the superclass's designated  initializer
Self=[super init];
// Did the superclass's designated  initializer  succeed?
If(self){
// Give the instance  variables  initial values
[self setpossessionname:name];
[self setserialnumber:snumber];
[self setvalueindollars:value];
Datecreated=[[nsdate alloc] init];
}
// Return the address of the newly initialized  object
Return self;
}
- (id)init
{
Return[self initwithpossessionname:@"Possession"
Valueindollars:0
Serialnumber:@""];
}
```

Class methods

```
@interface  Possession: nsobject
{3 nsstring *possessionname;
Nsstring *serialnumber;
Int valueindollars;
Nsdate*datecreated;
}
+ (id)randompossession;
- (id)initwithpossessionname:(nsstring *)name
Valueindollars:(int)value
Serialnumber:(nsstring*)snumber;
+ (id)randompossession
{
// Create an array of three adjectives
Nsarray *randomadjectivelist=[nsarray  arraywithobjects:@"Fluffy",
@"Rusty",
@"Shiny", nil];
// Create an array of three nouns
Nsarray *randomnounlist=[nsarray  arraywithobjects:@"Bear",
@"Spork",
```

```
@"Mac", nil];
Int adjectiveindex= rand()\%[randomadjectivelist  count];
Int nounindex= rand()\% [randomnounlist  count];
Nsstring *randomname=[nsstring  stringwithformat:@"\%@ \%@",
[randomadjectivelist  objectatindex:adjectiveindex],
[randomnounlist  objectatindex:nounindex]];
Int randomvalue= rand()\%100;
Nsstring *randomserialnumber= [nsstring  stringwithformat:@"\%c\%c\%c\%c\%c",
'0'+  rand()\% 10,
'A'+ rand()\%26,
'0'+  rand()\% 10,
'A'+ rand()\%26,
'0'+  rand()\% 10];
// Once again, ignore the memory problems with this method
Possession *newpossession=
[[self alloc] initwithpossessionname:randomname
Valueindollars:randomvalue
Serialnumber:randomserialnumber];
Return  newpossession;
}
```

Obj-C is dynamically typed. Nsmutablearray *items = [[nsmutablearray alloc] init]; [items dosomethingweird]; 2009-07-19 01:34:53.602 randompossessions[25326:10b]. *** -[nscfarray dosomethingweird]: unrecognized selector sent to instance 0x104b40. Objective-C has try-catch. Usually for runtime errors that the programmer made better for loop.

Properties, simplified accessors.

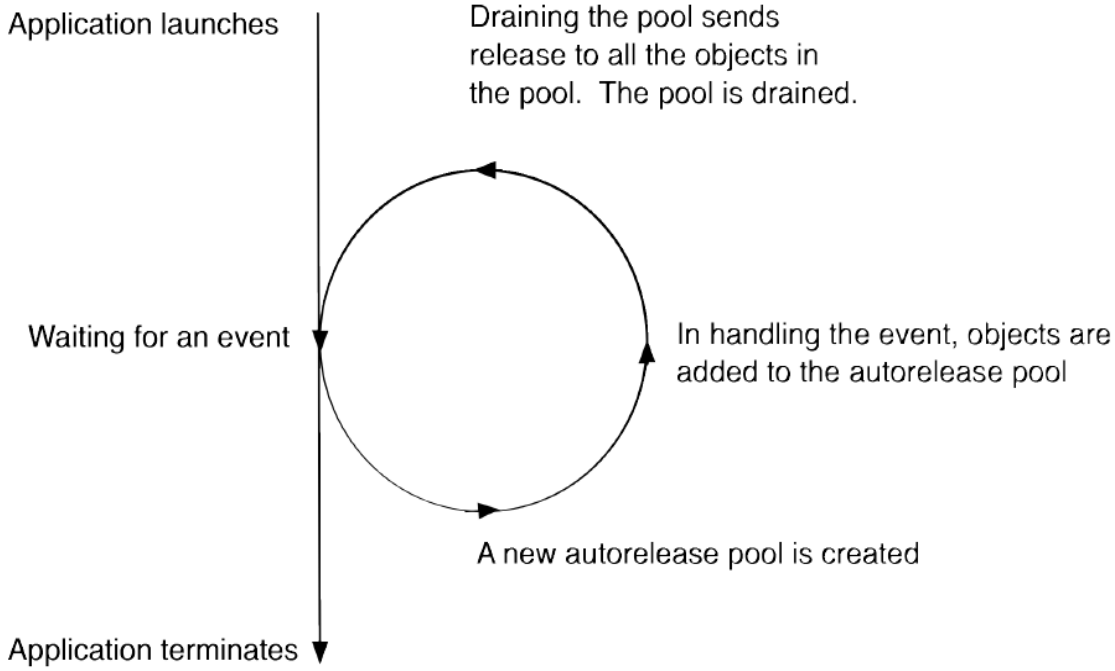
```
@interface  Possession: nsobject
{
Nsstring *possessionname;
Nsstring *serialnumber;
Int valueindollars;
Nsdate*datecreated;
}
+ (id)randompossession;
- (id)initwithpossessionname:(nsstring *)name
Valueindollars:(int)value
Serialnumber:(nsstring*)snumber;
- (id)initwithpossessionname:(nsstring *)name;
@property  nsstring*possessionname;
@property  nsstring*serialnumber;
@property int valueindollars;
@property  nsdate *datecreated;
@end
@implementation  Possession
@synthesize  possessionname, serialnumber, valueindollars, datecreated;
@property(nonatomic, copy) nsstring*possessionname;
@property(nonatomic, copy) nsstring*serialnumber;
- (void)setpossessionname:(nsstring *)str
{
Id t = [str copy];
[possessionname  release];
Possessionname = t;
}
```

Copy and mutablecopy.

Chapter 42

Lecture 42

Alloc and dealloc methods. Manual reference counting. Obj knows owner count retaincount. Retain and release methods. Should you release a created object that is returned ?. Want to say don't release but i don't want to be the owner. Autorelease. Added to nsautoreleasepool. Nsobject *x = [[[nsobject alloc] init] autorelease];.




```
Nsobject *x = [[[nsobject alloc] init] autorelease];
- (nsstring*)description
{
Nsstring *descriptionstring=
[[nsstring alloc] initwithformat:@"\%@(\%@): Worth \$\%d, Recorded on \%@",
Possessionname,
Serialnumber,
Valueindollars,
Datecreated];
Return  [descriptionstring  autorelease];
}
 - (nsstring  *)description
{
Return  [nsstring  stringwithformat:@"\%@  (\%@): Worth \$\%d, Recorded on  \%@",
Possessionname,
Serialnumber,
Valueindollars,
Datecreated];
}
- (void)setpossessionname:(nsstring *)str
{
```

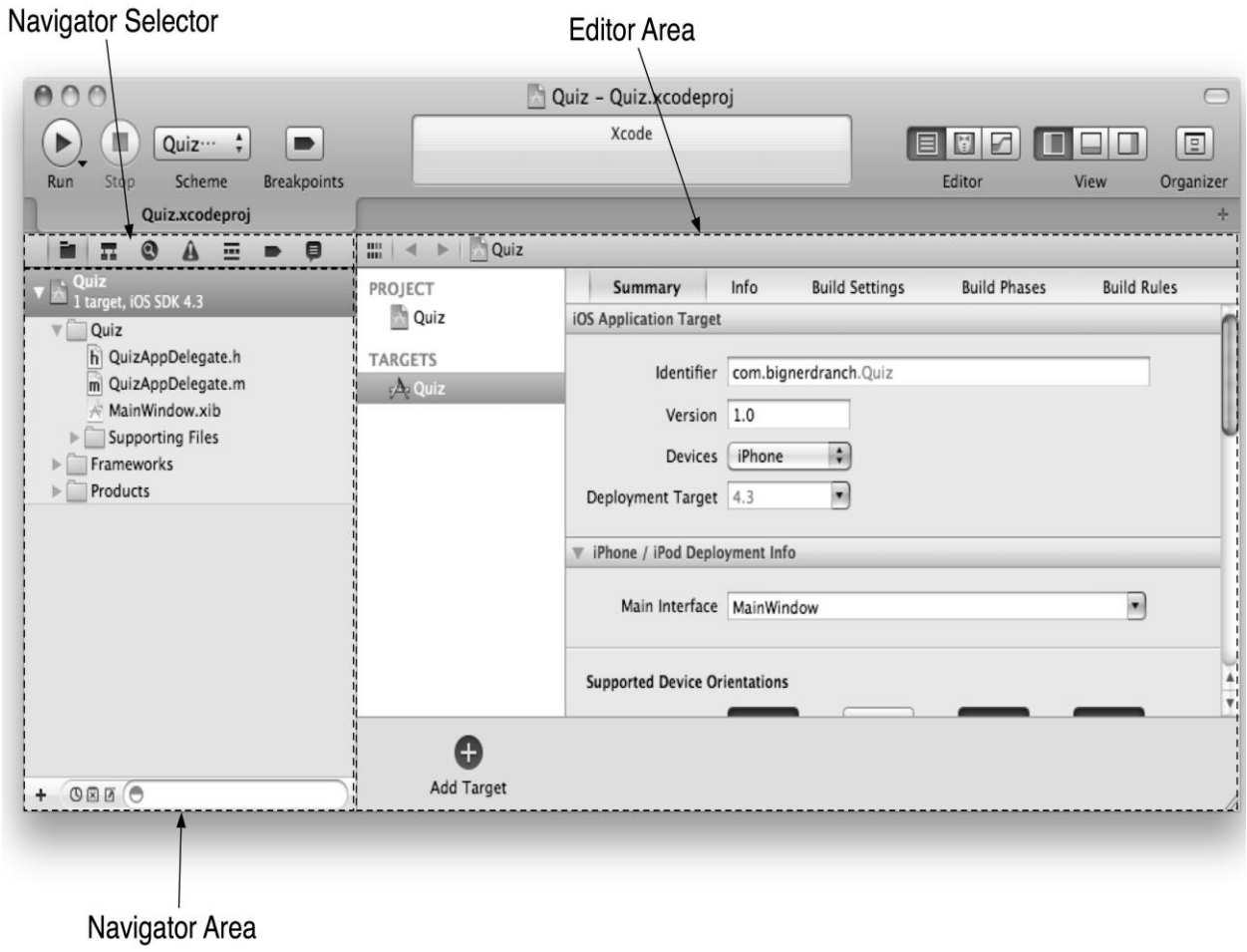
```
[str retain];
[possessionname  release];
Possessionname = str;
}
- (void)dealloc
{
[possessionname  release];
[serialnumber  release];
[datecreated  release];
[super dealloc];
}
```

Retain count rules. Init, new, copy in name. Assume you own. Any other means. Assume in autorelease. If you dont own and want to make sure, call retain. No longer need and own than release or autorelease. When 0 count, dealloc called.

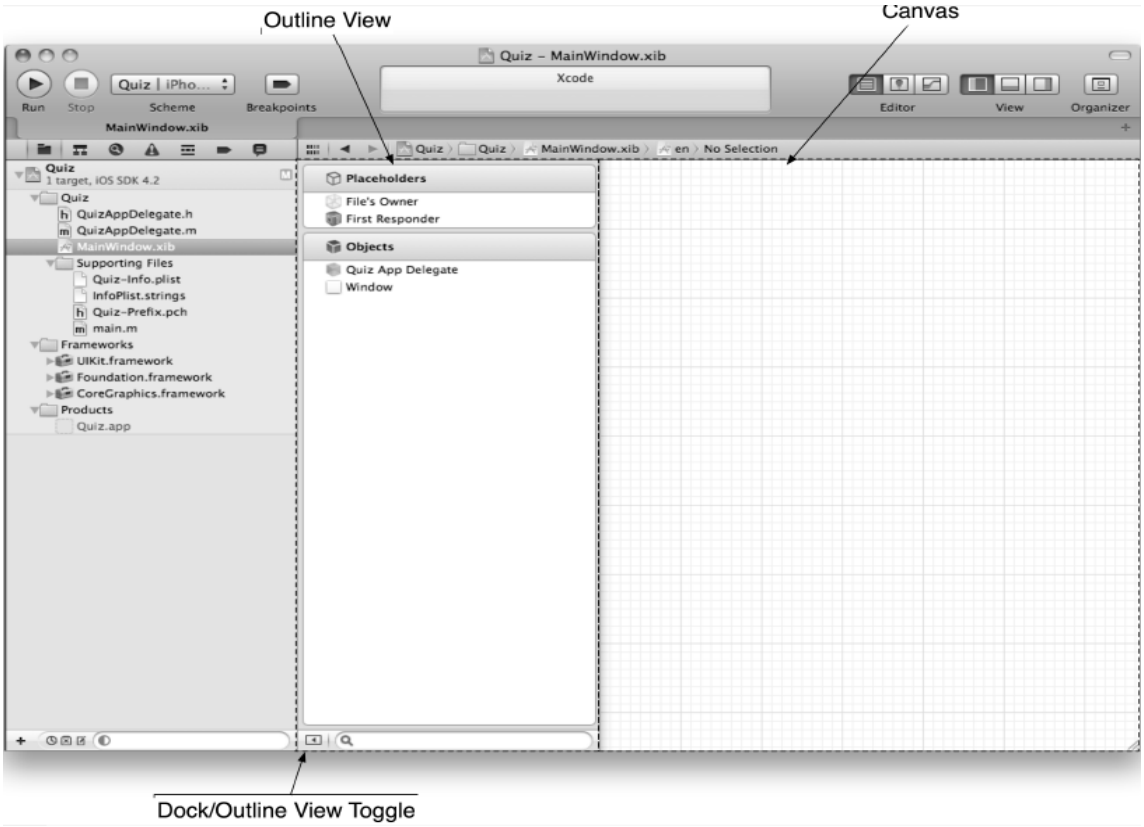
Protocols i.e. Interfaces.

Last lecture about ajax. Really learned how similar event driven programming is in JS than wpf. What about mobile. Same paradigm, different language. Same concepts, different incarnation. Ios programming. Learn objective-C. Event driven programming. Well see examples but you may not be able to try them. Need a mac computer and Xcode, teh ios simulator. Let's make a simple app.

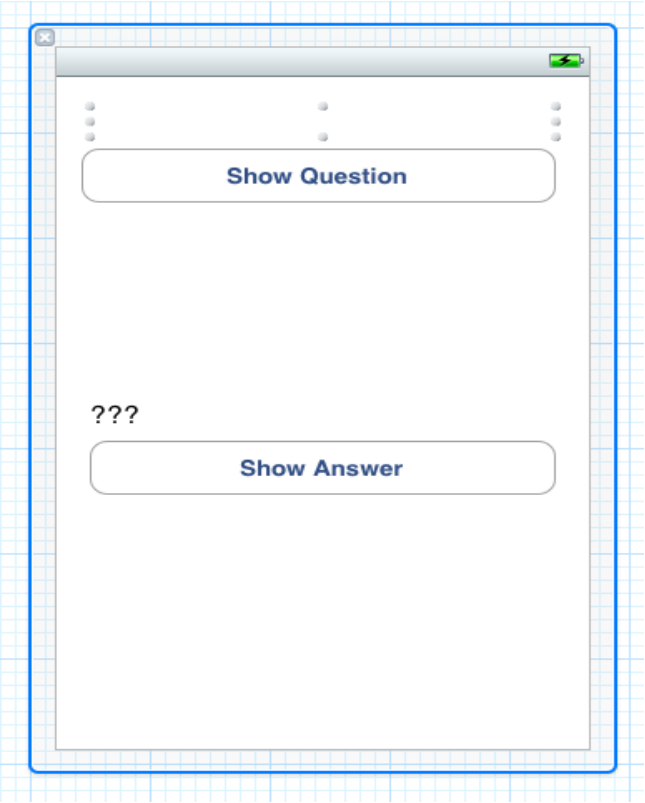
A quiz showing a question and revealing the answer. Create new project (window based applica.). Name it.



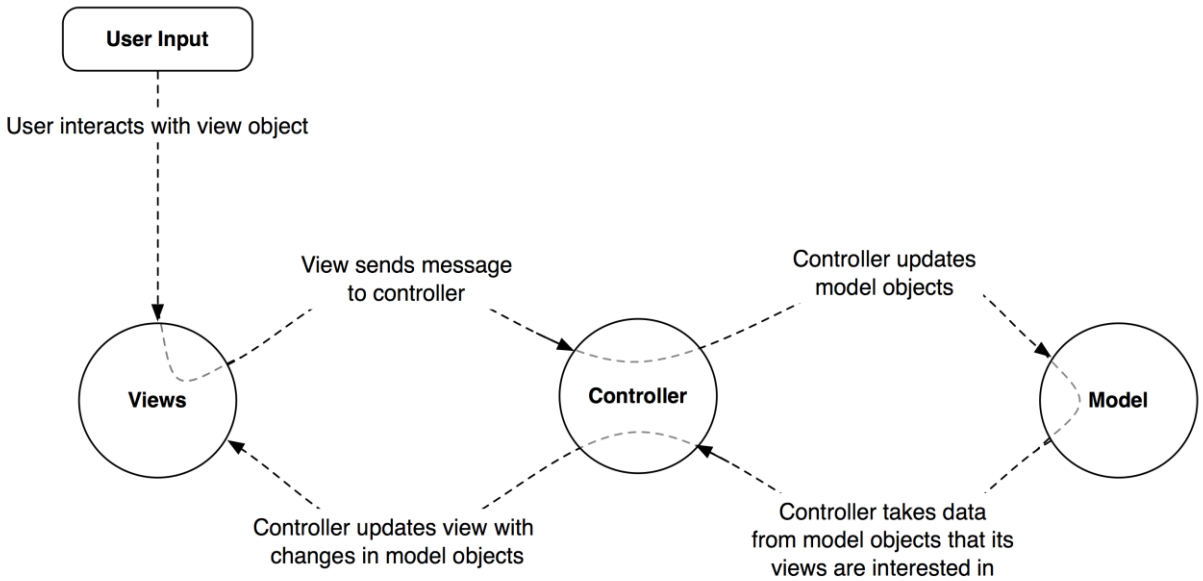
Similar to visual studio. The interface in xml. Xib file. Compiled to a nib file. Ios application a directory containing executables and resources. Sounds familiar.

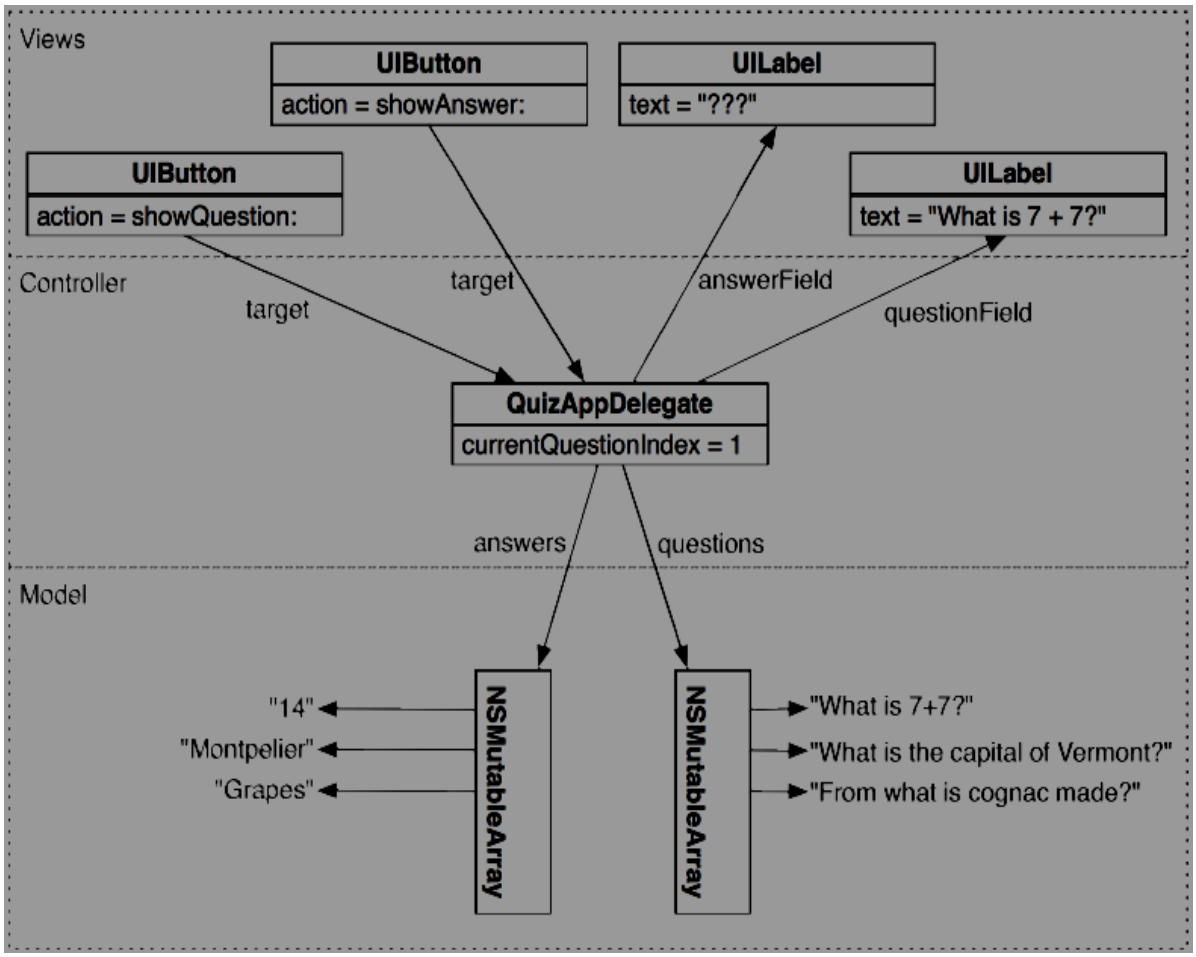


Xib editing. Select window to add controls. Drag buttons. Give them names.



Mvc pattern. View objects. Visible things. Uiview subclasses. Model objects. Hold data and know nothing about interface. Often use standard containers. Controllers keep things in sync.





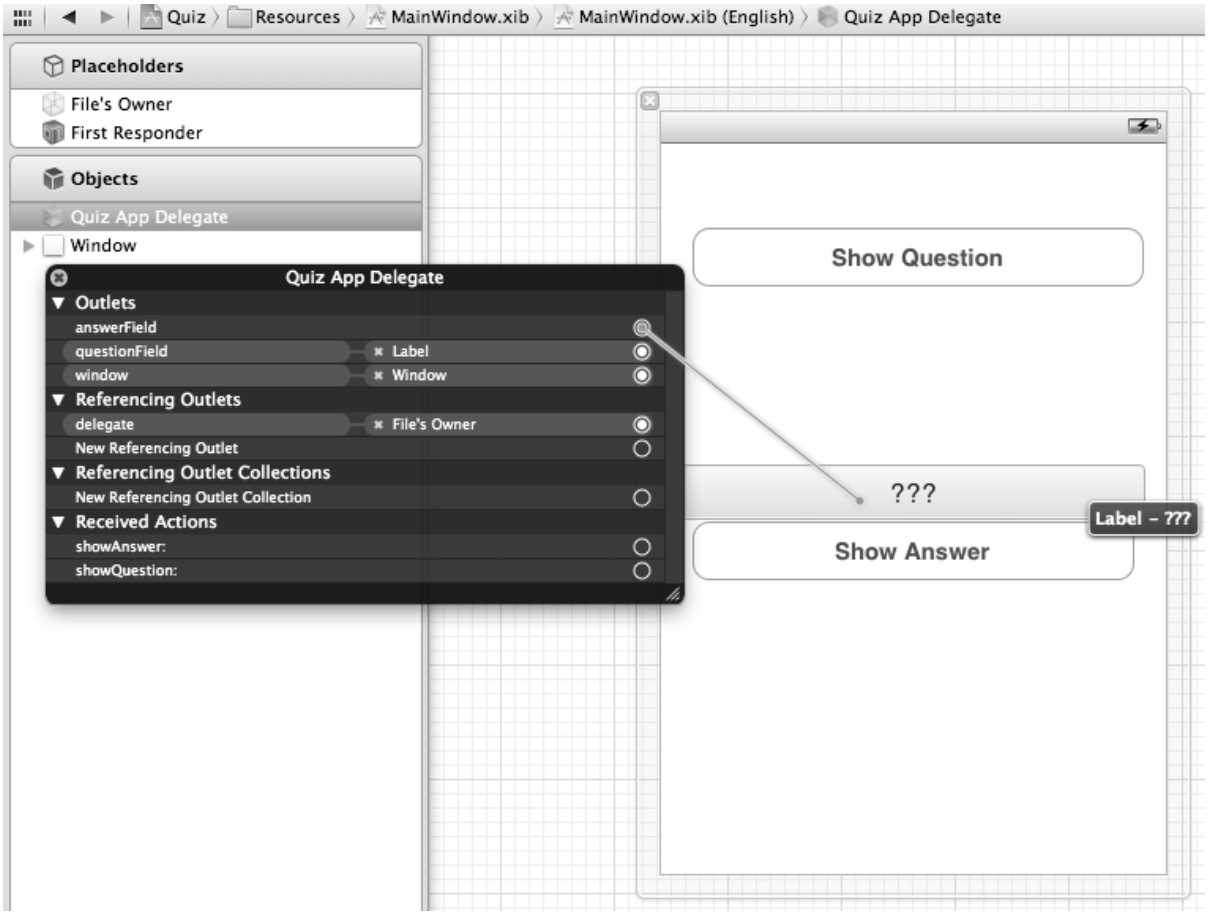
Iboutlet, ibaction.

```
@interface  quizappdelegate: nsobject
{
Int currentquestionindex;
// The model objects
Nsmutablearray  *questions;
Nsmutablearray  *answers; 7  // The view objects
<uiapplicationdelegate>

Iboutlet  uilabel *questionfield;

Iboutlet  uilabel *answerfield;

}

@property(nonatomic, retain) iboutlet  uiwindow *window;

- (ibaction)showquestion:(id)sender;

- (ibaction)showanswer:(id)sender;
```

```
@end
```

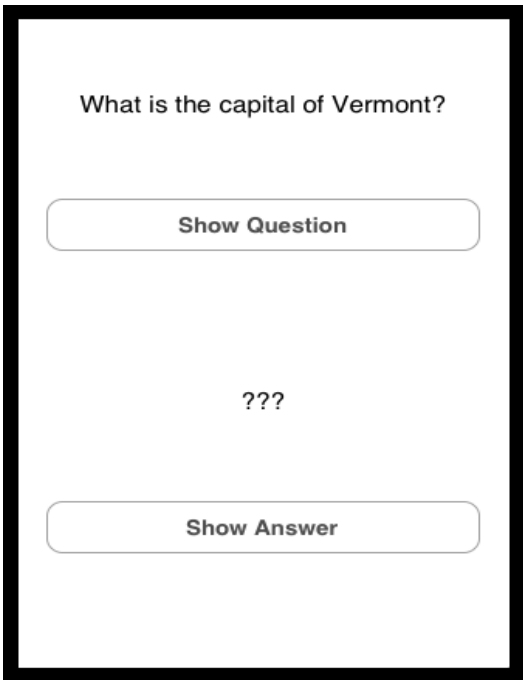
Setting connections. From the obj with pointer to the obj you want that pointer to point at. Control-drag from target to object. Check connection in connection inspector.

```
@interface  quizappdelegate: nsobject <uiapplicationdelegate>
{
Int currentquestionindex;
// The model objects
Nsmutablearray *questions;
Nsmutablearray *answers;
// The view objects
Iboutlet  uilabel *questionfield;
Iboutlet  uilabel *answerfield;
}
@property(nonatomic, retain) iboutlet  uiwindow *window;

- (ibaction)showquestion:(id)sender;
- (ibaction)showanswer:(id)sender;
@end
```




```
@implementation  quizappdelegate
- (id)init
{
// Call the init method  implemented by the superclass
Self  =  [super init]; 6  if(self){
// Create two arrays and make the pointers point to them
Questions= [[nsmutablearray  alloc] init];
Answers= [[nsmutablearray  alloc] init];
// Add questions and answers to the arrays
[questions  addobject:@"What is7+ 7?"];
[answers  addobject:@"14"];
[questions  addobject:@"What is the capital of Vermont?"];
[answers  addobject:@"Montpelier"];
[questions  addobject:@"From what is cognac made?"];
[answers  addobject:@"Grapes"];
}
// Return the address of the new object
Return self;
}
- (ibaction)showquestion:(id)sender
{
// Step to the next question
Currentquestionindex++;
// Am I past the last question?
If (currentquestionindex == [questions count]){
// Go back to the first question
```

```
Currentquestionindex= 0;
}

// Get the string at that index in the questions array
Nsstring *question= [questions  objectatindex:currentquestionindex];
// Log the string to the console
Nslog(@"displaying  question:%@", question);
// Display the string in the question field
[questionfield  settext:question];
// Clear the answer field
[answerfield  settext:@"???"];
}
- (ibaction)showanswer:(id)sender
{
// What is the answer to the current  question?
Nsstring *answer= [answers  objectatindex:currentquestionindex];
// Display it in the answer field
[answerfield  settext:answer];
}
```

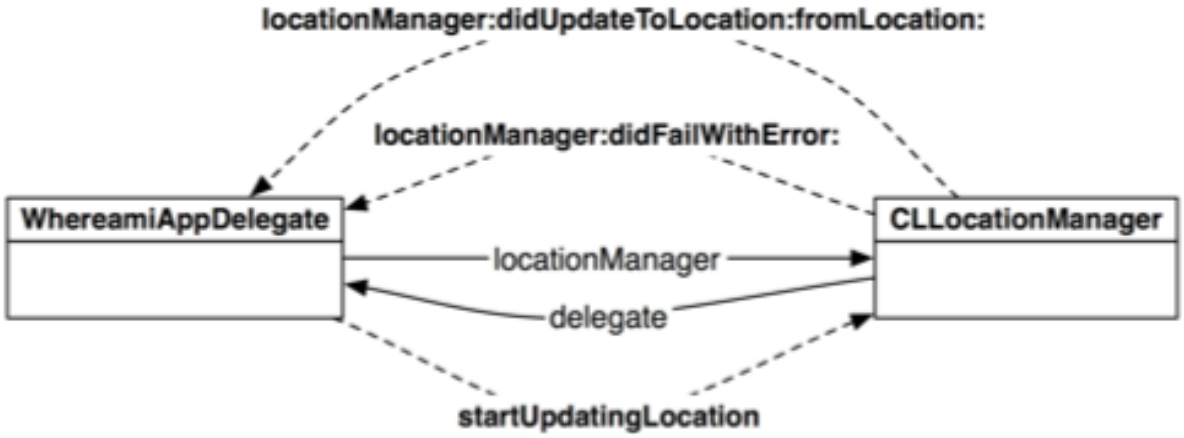


Chapter 43

Lecture 43

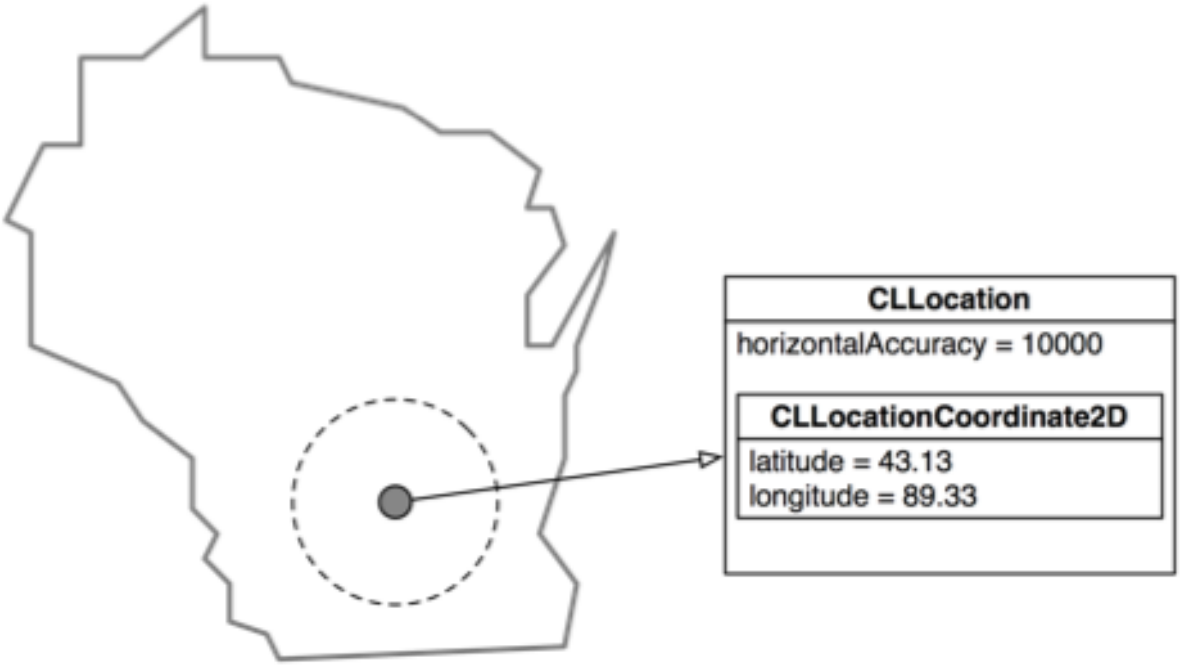
In Last Lecture, we discussed objective C memory management rules, Wrote our first iphone app: a quiz app, xib and nib files and interface editor, MVC pattern, iboutlet ibaction, Connection Inspector, then wrote init, showquestion, and showanswer.

Core Location Framework. Let's make a whereami application. Classes that enable finding geographical position. Distancefilter and desiredaccuracy properties.




```
#import  <uikit/uikit.h>
#import  <corelocation/corelocation.h> 3
@interface  whereamiappdelegate: nsobject
{ <uiapplicationdelegate>
Cllocationmanager *locationmanager;
}
@property(nonatomic, retain) iboutlet  uiwindow *window;
@end
- (BOOL)application:(uiapplication*)application
Didfinishlaunchingwithoptions:(nsdictionary *)launchoptions
{
// Create location  manager object
Locationmanager=[[cllocationmanager  alloc] init];
// We want all results from the location  manager
[locationmanager  setdistancefilter:kcldistancefilternone];
// And we want it to be as accurate as possible
// regardless of how much time/power it takes
[locationmanager  setdesiredaccuracy:kcllocationaccuracybest];
// Tell our manager to start looking for its location  immediately
[locationmanager  startupdatinglocation];
// This line may say self.window, don't worry about that
[[self window] makekeyandvisible];
Return YES;
}
```

Delegation. Locationmanager:didupdatetolocation:fromlocation: sent to delegate which we want



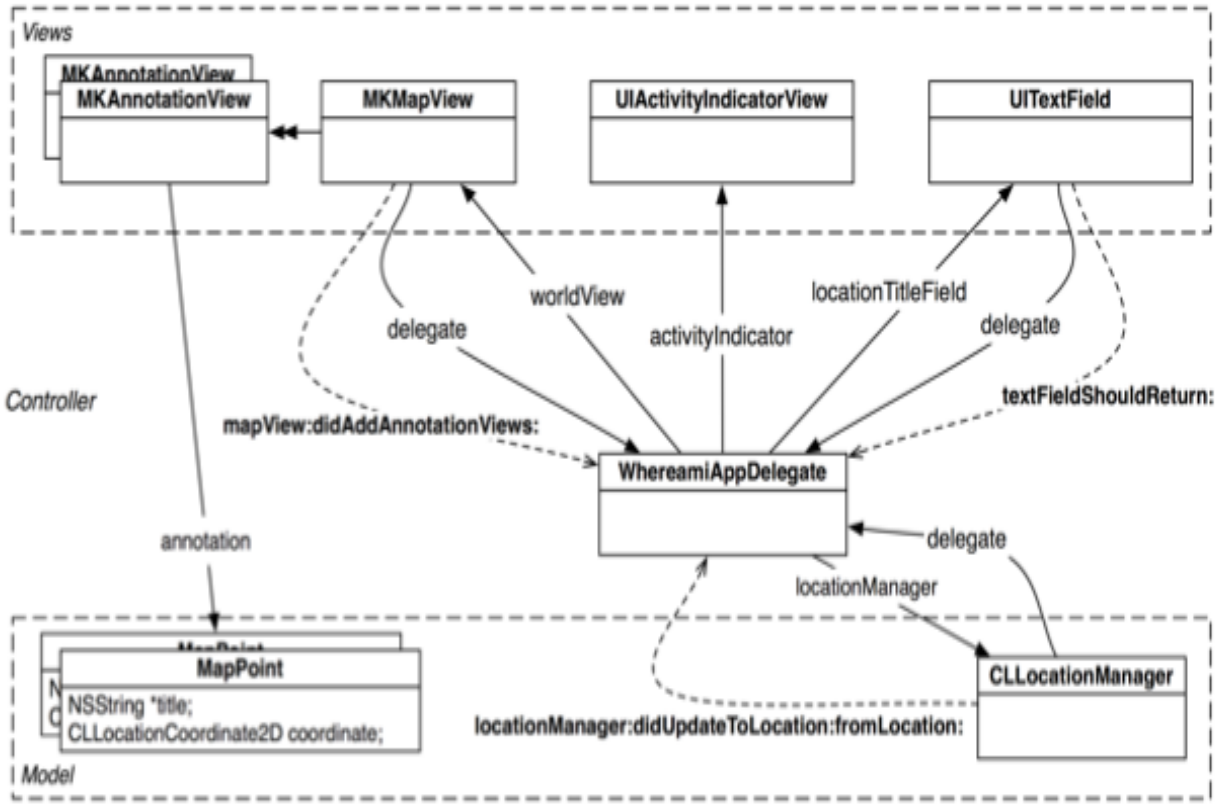
whereamiap-pdelegate to be.

```
- (void)locationmanager:(cllocationmanager *)manager
Didupdatetolocation:(cllocation *)newlocation
Fromlocation:(cllocation *)oldlocation
{
Nslog(@"%@", newlocation);
}
- (void)locationmanager:(cllocationmanager *)manager
Didfailwitherror:(nserror *)error
{
Nslog(@"Could not find location:%@", error);
}
```

Delegation is a design pattern. An OO approach to callbacks. Allows callback methods to share data. A delegatecan only be sent messages specified in its protocol. For every object that can have a delegate, there is a corresponding protocol.

```
@protocol  cllocationmanagerdelegate <nsobject>
@optional
- (void)locationmanager:(cllocationmanager *)manager
Didupdatetolocation:(cllocation *)newlocation
Fromlocation:(cllocation *)oldlocation;
- (void)locationmanager:(cllocationmanager *)manager
Didupdateheading:(clheading *)newheading;
- (BOOL)locationmanagershoulddisplayheadingcalibration:(cllocationmanager*)manager;
- (void)locationmanager:(cllocationmanager *)manager
Didenterregion:(clregion *)region;
- (void)locationmanager:(cllocationmanager *)manager
Didfailwitherror:(nserror *)error;
```

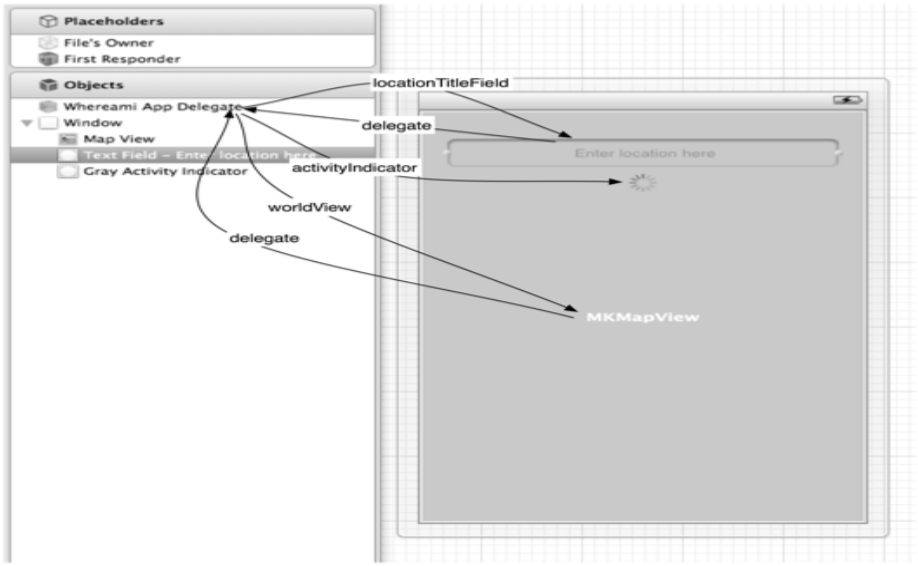
```
@end

 (void)finishedfindinglocation:(cllocation *)newlocation

EL updatemethod=@selector(locationmanager:didupdatetolocation:fromlocation:);
F ([[self delegate] respondstoselector:updatemethod]){
/ If the method is implemented, then we send the message.
[self delegate] locationmanager:self
Idupdatetolocation:newlocation
Romlocation:oldlocation];
Interface  whereamiappdelegate: nsobject
Uiapplicationdelegate, cllocationmanagerdelegate>
 (void)dealloc

F([locationmanager  delegate]== self)
Locationmanager  setdelegate:nil];
Locationmanager  release];
Window release];
Super dealloc];
```



Protocols are like interfaces in other langs i.e. No implementation. Protocols for delegations delegate protocols. Can be used to ask things from the delegate or inform of updates. Can be @optional methods. By default required. Here all were optional. Every object implements respondstoselector: something like this (above). Class has to declare protocols it implements like other langs. No warning anymore.

Memory and delegation. Delegates are usually controllers. Delegates never retained. Retain cycle. Assign attrib. Weak reference.

Let's display a map of current location.



Several instances of mkannotationview appear as icons on the mkmapview. An mkmapview displays the map and the labels for the recorded locations. A uiactivityindicatorview indicates that the device is working and not stalled. A uitextfield allows the user to input text to label the current location on the map.

Drag the map view onto uiwindow.

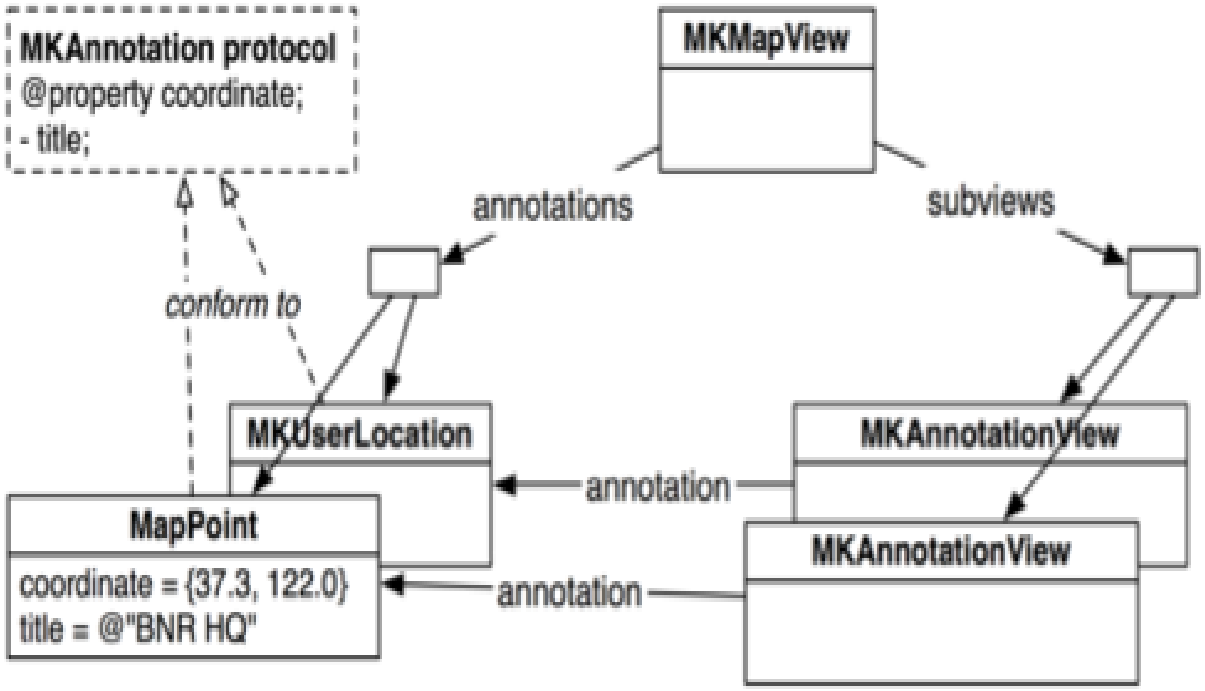
```
#import <mapkit/mapkit.h>

@interface  whereamiappdelegate: nsobject
<uiapplicationdelegate, cllocationmanagerdelegate>
{
Cllocationmanager *locationmanager;
Iboutlet  mkmapview*worldview;
Iboutlet  uiactivityindicatorview *activityindicator;
Iboutlet  uitextfield *locationtitlefield;
}
@property(nonatomic, retain) iboutlet  uiwindow *window;
@end
```

All we do is set showsuserlocation property of mkmapview and it will show users location.

```
- (BOOL)application:(uiapplication*)application
Didfinishlaunchingwithoptions:(nsdictionary *)launchoptions
{
Locationmanager=[[cllocationmanager  alloc] init];
[locationmanager  setdelegate:self];
[locationmanager  setdistancefilter:kcldistancefilternone];
[locationmanager  setdesiredaccuracy:kcllocationaccuracybest];
//[locationmanager  startupdatinglocation];
[worldview  setshowsuserlocation:YES];
// This line may say self.window, don't worry about that
[[self window] makekeyandvisible];
Return YES;
}
- (void)mapview:(mkmapview *)mv didupdateuserlocation:(mkuserlocation*)u
{
Cllocationcoordinate2d loc= [u coordinate];
Mkcoordinateregion  region= mkcoordinateregionmakewithdistance(loc,250,250);
[worldview  setregion:region animated:YES];
}
```

Too big the dot on world map. Want to zoom in. When to send a zoom in message. Instead mkmapview has delegate.

```
#import <Foundation/Foundation.h>
#import <corelocation/corelocation.h>
#import <mapkit/mapkit.h>
@interface  mappoint: nsobject <mkannotation>
{

Nsstring *title;
Cllocationcoordinate2d  coordinate;
}
//A new designated  initializer for instances of mappoint
- (id)initwithcoordinate:(cllocationcoordinate2d)c title:(nsstring *)t;
// This isa required  property from mkannotation
```




```
@property(nonatomic, readonly) cllocationcoordinate2d  coordinate;
// This is an optional  property from mkannotation
@property(nonatomic, copy) nsstring*title;
@end
```

Let's add an annotation. Mkannotation protocol Let's create a new mappoint class.

```
#import "mappoint.h"
@implementation  mappoint
@synthesize coordinate, title;
- (id)initwithcoordinate:(cllocationcoordinate2d)c title:(nsstring*)t
{
Self=[super init];
If(self){
Coordinate= c;
[self settitle:t];
}
Return self;
}
- (void)dealloc
{
[title release];
[super dealloc];
}
@end
```

In xib file set text fields delegate to be the instance of whereamiappdelegate. This methods from uitextfielddelegate.

```
@interface  whereamiappdelegate: nsobject
<uiapplicationdelegate, cllocationmanagerdelegate,
Mkmapviewdelegate, uitextfielddelegate> 6
- (BOOL)textfieldshouldreturn:(uitextfield  *)tf{
// This method isn't implemented yet- but will be soon.
```

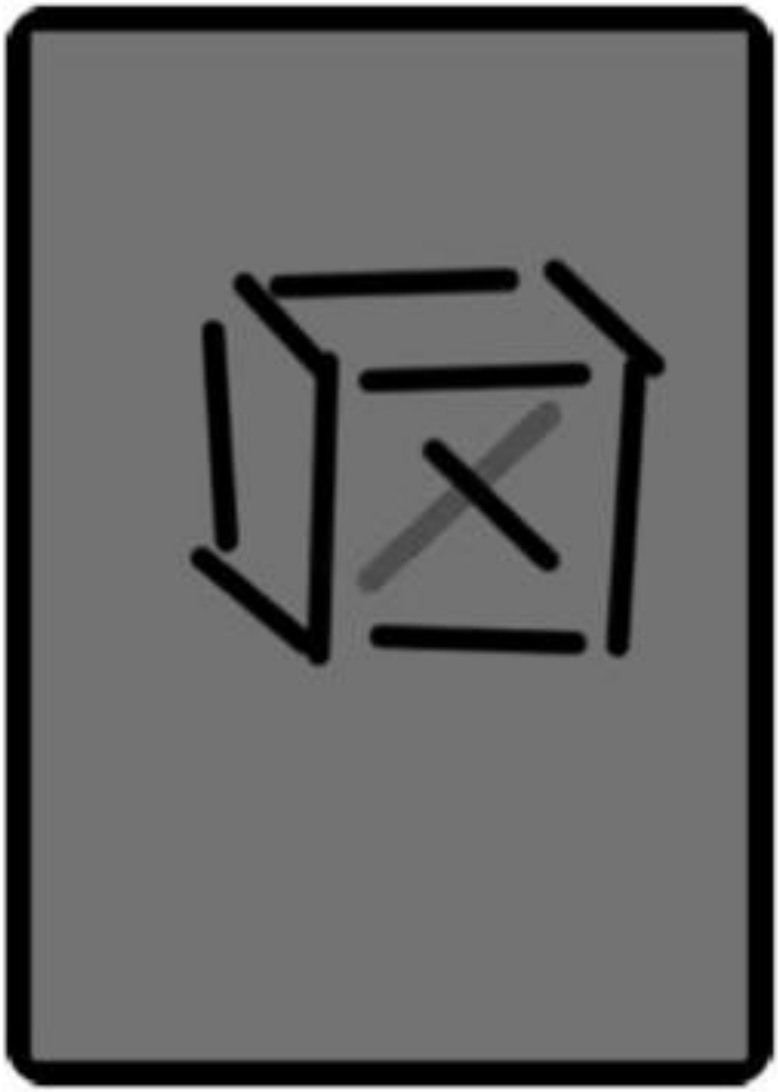
```
[self findlocation];
[tf resignfirstresponder];
Return YES;
} 14
@interface  whereamiappdelegate: nsobject
<uiapplicationdelegate, cllocationmanagerdelegate,
Mkmapviewdelegate, uitextfielddelegate>
{
Cllocationmanager  *locationmanager;
Iboutlet  mkmapview  *worldview;
Iboutlet  uiactivityindicatorview  *activityindicator;
Iboutlet  uitextfield  *locationtitlefield;
}
@property  (nonatomic, retain) iboutlet  uiwindow  *window;
- (void)findlocation;
- (void)foundlocation:(cllocation  *)loc;
@end 28
#import "whereamiappdelegate.h"
#import "mappoint.h"
@implementation  whereamiappdelegate 33
- (void)findlocation
{
[locationmanager  startupdatinglocation];
[activityindicator  startanimating];
[locationtitlefield  sethidden:YES];
} 40
- (void)foundlocation:(cllocation  *)loc
{
Cllocationcoordinate2d  coord=  [loc coordinate];
// Create an instance of mappoint with the current data
Mappoint  *mp=  [[mappoint alloc] initwithcoordinate:coord
Title:[locationtitlefield text]];
// Add it to the map view
[worldview  addannotation:mp];
// mkmapview  retains its annotations, we can release
[mp release]; 51  // Zoom the region to this location means we can implement
Mkcoordinateregion  region= mkcoordinateregionmakewithdistance(coord, 250, 250);
[worldview  setregion:region animated:YES];
[locationtitlefield  settext:@""];
[activityindicator  stopanimating];
[locationtitlefield  sethidden:NO];
[locationmanager  stopupdatinglocation];
} 59
- (void)locationmanager:(cllocationmanager  *)manager
Didupdatetolocation:(cllocation  *)newlocation
Fromlocation:(cllocation  *)oldlocation
{
Nslog(@"\%@", newlocation); 65  // How many seconds ago was this new location  created?
Nstimeinterval t= [[newlocation  timestamp] timeintervalsincenow];
// cllocationmanagers will return the last found location of the
// device first, you don’t want that data in this case.
// If this location was made more than 3 minutes ago, ignore it.
If(t<-180){
// This is cached data, you don’t want it, keep looking
Return;
}
[self foundlocation:newlocation];}
```

Chapter 44

Lecture 44

Let’s discuss touch events. Touch events are hallmark of mobile devices Let’s make a drawing app like brushes. A finger or fingers touches the screen - (void)touchesbegan:(nsset *)touches withevent:(uievent *)event; a finger or fingers move across the screen (This message is sent repeatedly as a finger moves.) - (void)touchesmoved:(nsset *)touches withevent:(uievent *)event;

A finger or fingers is removed from the screen - (void)touchesended:(nsset *)touches withevent:(uievent *)event; a system event, like an incoming phone call, interrupts a touch before it ends - (void)touchescancelled:(nsset *)touches withevent:(uievent *)event;



Events added to event queue. A uitouch object created and tracked for a finger. Set of uitouches passed. One per finger. Only the moving, begining, or ending event passed. Let’s make our app.

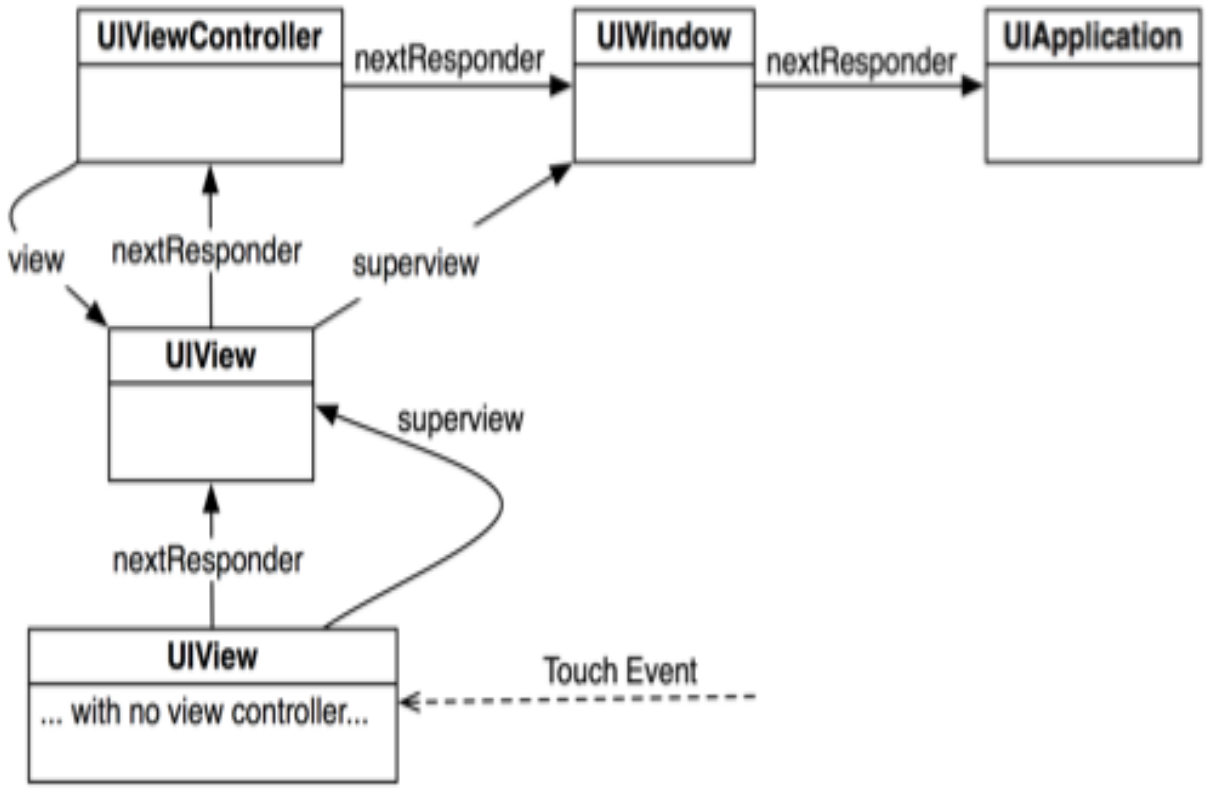
```
#import <Foundation/Foundation.h>
@interface Line: nsobject{
Cgpoint begin;
Cgpoint end;
}
@property(nonatomic) cgpoint begin;
@property(nonatomic) cgpoint end;
@end
#import "Line.h"
@implementation Line
@synthesize begin, end;
@end
#import <Foundation/Foundation.h>
#import <uikit/uikit.h>
@interface  touchdrawview: uiview
{
Nsmutabledictionary*linesinprocess;
Nsmutablearray *completelines;
}
- (void)clearall;
@end
```

Use IB to set the view to default

```
#import "touchdrawview.h"
#import "Line.h"
@implementation  touchdrawview
- (id)initwithcoder:(nscoder*)c
{
Self = [super  initwithcoder:c];
If(self){
Linesinprocess= [[nsmutabledictionary  alloc] init];
Completelines= [[nsmutablearray  alloc] init];
[self  setmultipletouchenabled:YES];
}
Return self;
}
- (void)dealloc
{
[linesinprocess  release];
[completelines  release];
[super dealloc];
}
- (void)clearall
{
// Clear the collections
[linesinprocess  removeallobjects];
[completelines  removeallobjects];
// Redraw
[self setneedsdisplay];
}
- (void)drawrect:(cgrect)rect
{
Cgcontextref  context= uigraphicsgetcurrentcontext();
Cgcontextsetlinewidth(context, 10.0);
Cgcontextsetlinecap(context, kcglinecapround);
```

```
// Draw complete lines in black
[[uicolor  blackcolor] set];
For(Line *line in completelines){
Cgcontextmovetopoint(context,[line begin].x,[line begin].y);
Cgcontextaddlinetopoint(context,[line end].x,[line end].y);
Cgcontextstrokepath(context);
}
// Draw lines in process in red
[[uicolor  redcolor] set];
For(nsvalue *v in linesinprocess){
Line *line=[linesinprocess  objectforkey:v];
Cgcontextmovetopoint(context,[line begin].x,[line begin].y);
Cgcontextaddlinetopoint(context,[line end].x,[line end].y);
Cgcontextstrokepath(context);
}
}
- (void)touchesbegan:(nsset *)touches
Withevent:(uievent *)event
{
For(uitouch *t in touches){
// Is this a double tap?
If([t tapcount]> 1){
[self clearall];
Return;
}
// Use the touch object (packed in an nsvalue) as the key
Nsvalue *key= [nsvalue  valuewithpointer:t];
// Create a line for the value
Cgpoint loc=[t locationinview:self];
Line *newline=[[Line alloc] init];
[newline  setbegin:loc];
[newline setend:loc];
// Put pair in dictionary
[linesinprocess  setobject:newline forkey:key];
// There isa memory leak in this method
// You will find it using Instruments in the nextchapter
}
}
- (void)touchesmoved:(nsset *)touches
Withevent:(uievent *)event
{
// Update  linesinprocess with moved touches
For(uitouch *t in touches){
Nsvalue *key= [nsvalue  valuewithpointer:t];
// Find the line for this touch
Line *line=[linesinprocess  objectforkey:key];
// Update the line
Cgpoint loc=[t locationinview:self];
[line setend:loc];
}
// Redraw
[self setneedsdisplay];
}
- (void)touchesended:(nsset *)touches
Withevent:(uievent *)event
{
[self endtouches:touches];
```

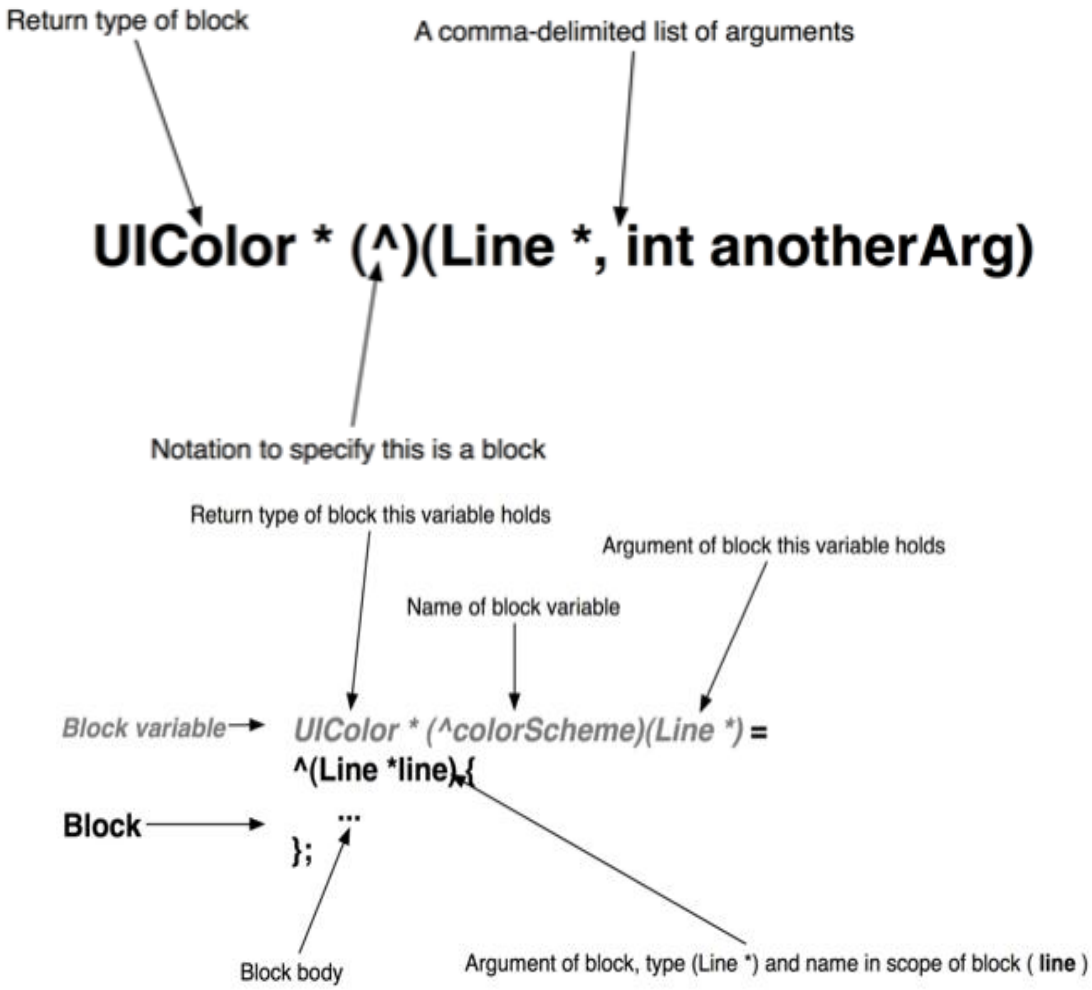
```
}
 - (void)touchescancelled:(nsset*)touches
 Withevent:(uievent *)event
 {
 [self endtouches:touches];
 }
 - (void)endtouches:(nsset *)touches
 {
 // Remove ending touches from dictionary
 For(uitouch *t in touches){
 Nsvalue *key=[nsvalue  valuewithpointer:t];
 Line *line=[linesinprocess  objectforkey:key];
 // If this is a double tap, 'line' will be nil,
 // so make sure not to add it to the array

 If(line){
 [completelines  addobject:line];
 [linesinprocess  removeobjectforkey:key];
 }
 }
 // Redraw
 [self setneedsdisplay];
}
```



Let's use Blocks in ios 4.0 to change colors based on angle and length.

```
@interface Line: nsobject{
Cgpoint begin;
Cgpoint end;
Uicolor  *color;
}
@property  (nonatomic) cgpoint begin;
@property  (nonatomic) cgpoint end;
@property  (nonatomic, retain) uicolor  *color;
@end 11 @implementation Line
@synthesize color;
- (id)init
{
Self=  [super init];
If  (self){ 17  [self setcolor:[uicolor  blackcolor]];
}
Return self;
}
- (void)dealloc
{
```

```
[color release];
[super dealloc];
}
- (void)drawrect:(cgrect)rect{
Cgcontextref  context= uigraphicsgetcurrentcontext();
Cgcontextsetlinewidth(context, 10.0);
Cgcontextsetlinecap(context, kcglinecapround);
//[[uicolor  blackcolor] set];
For(Line *line in completelines){
[[line color] set];
Cgcontextmovetopoint(context,[line begin].x,[line begin].y);
Cgcontextaddlinetopoint(context,[line end].x,[line end].y);
Cgcontextstrokepath(context);
}
[[uicolor  redcolor] set];
For(nsvalue *v in linesinprocess){
Line *line=[linesinprocess  objectforkey:v];
Cgcontextmovetopoint(context,[line begin].x,[line begin].y);
Cgcontextaddlinetopoint(context,[line end].x,[line end].y);
Cgcontextstrokepath(context);
}
}
```



```
- (void)transformlinecolorswithblock:(uicolor* (^)(Line *))colorforline
{
For(Line *l in completelines){
Uicolor *c= colorforline(l);
[l setcolor:c];
}
[self setneedsdisplay];
}

- (void)colorize
{
// Vertical means more red, horizontal means more green,
// longer means more blue
//A block variable named colorscheme is created here:
Uicolor* (^colorscheme)(Line*)=^(Line*l){
// Compute delta between begin and end points
// for each component
Float dx=[l end].x- [l begin].x;
Float dy=[l end].y- [l begin].y;
// If dx is near zero, red= 1, otherwise, use slope
Float r = (fabs(dx)< 0.001?1.0: fabs(dy/ dx));
// If dy is near zero, green=1, otherwise, use inv. Slope
Float g = (fabs(dy)< 0.001?1.0: fabs(dx/ dy));
// blue= length over 300
Float b = hypot(dx, dy)/300.0;
Return[uicolor  colorwithred:r green:g blue:b alpha:1];
};
// Pass this colorscheme block to the method
// that will iterate over every line and assign
// the computed color to that line
[self  transformlinecolorswithblock:colorscheme];
}
```

Colors when you shake.

```
- (BOOL)canbecomefirstresponder
{
Return YES;
}
- (void)didmovetowindow
{
[self becomefirstresponder];
}
- (void)motionbegan:(uieventsubtype)motion
Withevent:(uievent *)event
{
[self colorize];
}
```

Blocks capture variables like anonymous functions in C#. Inline block objects(values copied). Blocks are an alternate to callbacks. Blocks are used for GCD. Kind of like tasks.

Grand central dispatch. Dispatch queues pools of threads managed.

```
- (void) docalculation{
/* Do your calculation here */
}
- (void) calculationthreadentry{
```

```
@autoreleasepool{
Nsuinteger  counter= 0;
While ([[nsthread  currentthread] iscancelled] == NO){
[self docalculation];
Counter++;
If (counter >= 1000){
Break;
}
}
}
}
- (BOOL)   application:(uiapplication*)application
Didfinishlaunchingwithoptions:(nsdictionary *)launchoptions{
/* Start the thread*/
[nsthread  detachnewthreadselector:@selector(calculationthreadentry)
Totarget:self
Withobject:nil];
Self.window = [[uiwindow alloc] initwithframe:
[[uiscreen  mainscreen] bounds]];
Self.window.backgroundcolor = [uicolor  whitecolor];
[self.window  makekeyandvisible];
Return YES;
}
```

Calling back on UI thread

```
Dispatch_queue_t  mainqueue = dispatch_get_main_queue();
Dispatch_async(mainqueue, ^(void) {
[[[uialertview alloc] initwithtitle:@"GCD"
Message:@"GCD is amazing!"
Delegate:nil
Cancelbuttontitle:@"OK"
Otherbuttontitles:nil, nil] show];
});
```

```
Void (^printfrom1to1000)(void) = ^{
Nsuinteger  counter= 0;
For(counter = 1;
Counter <= 1000;
Counter++){
Nslog(@"Counter = %lu- Thread = %@",
(unsigned long)counter,
[nsthread  currentthread]);
}
};
Dispatch_queue_t  concurrentqueue =
Dispatch_get_global_queue(DISPATCH_QUEUE_PRIORITY_DEFAULT, 0);
Dispatch_sync(concurrentqueue, printfrom1to1000);
Dispatch_sync(concurrentqueue, printfrom1to1000);
```

Chapter 45

Lecture 45

Let's download an image asynchronously.

```
Dispatch_queue_t  concurrentqueue =
Dispatch_get_global_queue(DISPATCH_QUEUE_PRIORITY_DEFAULT,0);
Dispatch_async(concurrentqueue,^{
__block uiimage*image= nil;
Dispatch_sync(concurrentqueue,^{
/* Download the image here*/
});
Dispatch_sync(dispatch_get_main_queue(),^{
/* Show the image to the user here on the main queue*/
});
});
- (void) viewdidappear:(BOOL)paramanimated{
Dispatch_queue_t  concurrentqueue =
Dispatch_get_global_queue(DISPATCH_QUEUE_PRIORITY_DEFAULT,0);
Dispatch_async(concurrentqueue,^{
__block uiimage*image= nil;
Dispatch_sync(concurrentqueue,^{
/* Download the image here*/
/* ipad's image from Apple's website. Wrap it into two
Lines as the URL is too long to fit into one line*/
Nsstring*urlasstring=@"http://images.apple.com/mobileme/features"\
"/images/ipad_findyouripad_20100518.jpg";
NSURL*url=[NSURL  urlwithstring:urlasstring];
Nsurlrequest*urlrequest=[nsurlrequest  requestwithurl:url];
Nserror*downloaderror=nil;
Nsdata*imagedata=[nsurlconnection
Sendsynchronousrequest:urlrequest
Returningresponse:nil
Error:\&downloaderror];
If(downloaderror==nil\&\&
Imagedata!=nil){
Image=[uiimage  imagewithdata:imagedata];
/* We have the image. We can use it now*/
}
Else if(downloaderror!=nil){
Nslog(@"Error happened=\%@",downloaderror);
} else{
Nslog(@"No data could get downloaded from the URL.");
}
});
Dispatch_sync(dispatch_get_main_queue(),^{
/* Show the image to the user here on the main queue*/
If(image!=nil){
/* Create the image view here*/
Uiimageview*imageview=[[uiimageview alloc]
Initwithframe:self.view.bounds];
/* Set the image*/
[imageview  setimage:image];
/* Make sure the image is not scaled  incorrectly*/
```

```
[imageview  setcontentmode:uiviewcontentmodescaleaspectfit];

/* Add the image to this view controller’s view */
[self.view addsubview:imageview];
} else{
Nslog(@"Image isn’t downloaded. Nothing to display.");
}
});
});
}
```

Generate 10k random numbers if needed. Read 10K random numbrs, sort n display. Uitableview.

```
- (nsstring*) filelocation{
/* Get the document folder(s)*/
Nsarray *folders =
Nssearchpathfordirectoriesindomains(nsdocumentdirectory
Nsuserdomainmask,
YES);
/* Did we find anything?  */
If([folders count]  == 0){
Return nil;
}
/* Get the first folder  */ 12  nsstring *documentsfolder = [folders  objectatindex:0];

/* Append the file name to the end of the documents path */

Return[documentsfolder
```

```
Stringbyappendingpathcomponent:@"list.txt"];

}

- (BOOL) hasfilealreadybeencreated{

BOOL result = NO;

Nsfilemanager*filemanager = [[nsfilemanager  alloc] init];

If([filemanager  fileexistsatpath:[self filelocation]]){

Result= YES;

}
Return result;
}

Dispatch_queue_t  concurrentqueue =
Dispatch_get_global_queue(DISPATCH_QUEUE_PRIORITY_DEFAULT,0);
/* If we have not already saved an array of 10,000
Random numbers to the disk before, generate these numbers now
And then save them to the disk in an array*/
Dispatch_async(concurrentqueue, ^{
Nsuinteger  numberofvaluesrequired = 10000;
If([self  hasfilealreadybeencreated] == NO){
Dispatch_sync(concurrentqueue, ^{
Nsmutablearray *arrayofrandomnumbers =
[[nsmutablearray  alloc] initwithcapacity:numberofvaluesrequired]
Nsuinteger  counter= 0;
For(counter = 0;
Counter < numberofvaluesrequired;
Counter++){
Unsigned int randomnumber =
Arc4random()% ((unsigned int)RAND_MAX + 1);
[arrayofrandomnumbers  addobject:
[nsnumber  numberwithunsignedint:randomnumber]];
}

/* Now let's write the array to disk */
[arrayofrandomnumbers  writetofile:[self filelocation]
Atomically:YES];
});
}
__block  nsmutablearray *randomnumbers = nil;
/* Read the numbers from disk and sort them in an
Ascending  fashion */57 dispatch_sync(concurrentqueue, ^{
/* If the file has now been created, we have to read it*/
If([self  hasfilealreadybeencreated]){
Randomnumbers= [[nsmutablearray  alloc]
Initwithcontentsoffile:[self filelocation]];
/* Now sort the numbers */
[randomnumbers  sortusingcomparator:
^nscomparisonresult(id obj1, id obj2) {
Nsnumber *number1 = (nsnumber*)obj1;
Nsnumber *number2 = (nsnumber*)obj2;
```

```
Return[number1 compare:number2];
}];
}
});
Dispatch_async(dispatch_get_main_queue(), ^{
If([randomnumbers count]> 0){
/* Refresh the UI here using the numbers in the
Randomnumbers  array */
}
});
});
```

Dispatch after dispatches after a delay. Timers. Dependencies. Group of tasks.

Course Overview:

Where did we start from. Where to go from here.

- Started with handling multiple input sources in C++, discovered message loop refactored, understood events, event processing, source, target, event object downloaded VS and C# language and its features, examples of OO programming in C#.
- Discussed delegates, events, exception handling, attributes, collections. Worked with XML docs. Wpf history and xaml. Property elements and markup extensions. Mixing xaml and procedural code.
- Discussed logical and visual trees. Dependency properties. Change notifications. Prop val. Inheritance. Attached properties. Sizing, positioning, and transforming elements.
- Discussed transforms, panels stackpanel, wrappanel, canvas, dockpanel, grid. Content overflow, clipping, scaling, scrolling.
- Discussed Events, input events, attached events, touch events (manipulation... High level). Commands, persisting and restoring.
- Covered resources. Binary and logical. Static vs dynamic logical resources. Data binding, binding object, binding markup extension. Binding to collection, implicit datacontext. Datatemplates and value converters. Customizing collection view. Sorting, filtering, grouping, navigating. Data providers. Xml and object data providers.
- Concurrency and threads. Captured variables. Synchronization context and tasks. Continuations. Task completion source. Sync vs async.
- Course grained vs fine grained sync. Async wait keywords in C# 5.0. Parallelism. Cancellation and progress reporting. Task combinator and task parallel library. Parallel.Invoke, For, foreach. Concurrent collections.
- JS history and jquery library. HTML CSS JS. Client side vs server side. Dom. Jquery selectors and filters. Changing attributes and elements. Events and animations. Ajax. Xmlhttprequest, get put load. JSON. Mobile development. Objective C history. Call syntax and OO concepts. Properties. Retain count and memory management. Xib, nib, iboutlet, ibaction, interface builder.
- Button events and a QA app. Protocols and delegates. Location and map kits. Touch events and blocks. GCD and multithreading.

