VS 2010 og .NET framework 4.0

Captator
Tlf: 8620 4242
www.captator.dk

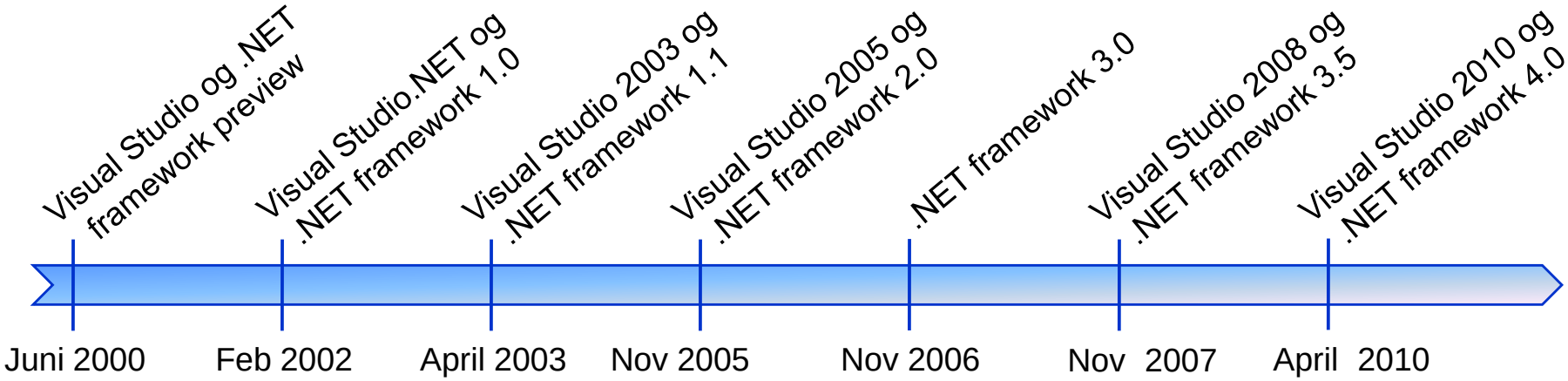
Henrik Lykke Nielsen
Softwarearkitekt, Microsoft Regional Director for Denmark
lykke@captator.dk
Mobil: 2237 3311

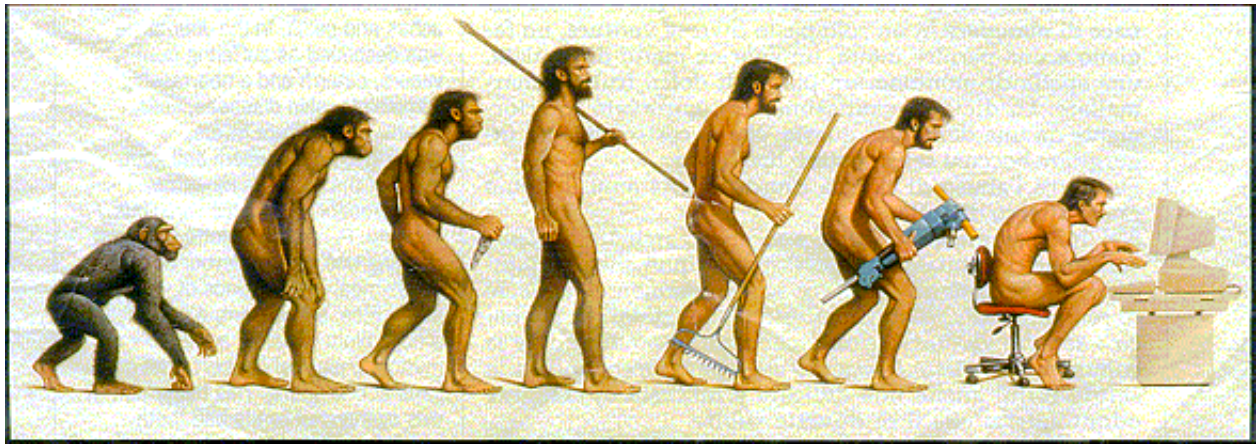
Agenda

- **Sprog features i C# 4.0 og VB 10.0**
  - Nyheder i C# (som VB allerede har)
  - Nyheder i VB (som C# allerede har)
  - Generisk co- og kontravarians (nyt i både C# og VB)
  - Den dynamiske type (C#)
  - COM support
  - ExpandoObject, DynamicObject
- **Visual Studio features festivitas**
- **.NET framework features**
  - Diverse framework nyheder
  - Task Parallel Library
- **Visual Studio Versioner**

Visual Studio og .NET frameworket

- **En platform i udvikling**





Navngivne argumenter

C# Captator

- Betydningen af et argument kan være uklar
- Navngivning af argumenter kan hjælpe på dette

```
int result1 = String.Compare("ABC", "abc", true);
int result2 = String.Compare("ABC", "abc", ignoreCase: true);
```

- Kan sendes i vilkårlig orden

```
int result3 = String.Compare(ignoreCase: true, strB: "abc",
    strA: "ABC");
```

- Kun navngivne argumenter må følge efter et navngivet argument

```
int result4 = String.Compare(strB: "abc", "ABC", ignoreCase: true);
```

✘ Kompiler fejl

Optionelle argumenter

C# Captator

- **Ofte bruges overloads til at implementere, at et argument er optionelt**

```
string UseOverloads(IEnumerable<string> strings)
{
  return UseOverloads(strings, "|");
}

string UseOverloads(IEnumerable<string> strings, string delimiter)
{
  /* ... */
}
```

- **Argumenter kan nu være optionelle**
- **Defineres ved angivelse af default konstant værdi**

```
public static string UseOptional(IEnumerable<string> strings,
                                 string delimiter = "|")
{
  /* ... */
}
```

Optionelle argumenter

C# Captator

- **En optional parameter kan udelades – følgende kald giver samme resultat**

```
string result1 = UseOptional(new string[] { "abc", "123" }, "|");

string result2 = UseOptional(new string[] { "abc", "123" });
```

```
string result2 = UseOptional(new string[] { "abc", "123" },);
```

```
string MainWindow.UseOptional(IEnumerable<string> strings, [string delimiter = "|"])
```

- **Alle efterfølgende parametre skal også være defineret som værende optionelle**
  - Metoder uden optionelle argumenter foretrækkes ved valg af overload
- **ref og out parametre kan ikke have default værdier**

Automatic properties

VB Captator

- **En automatic property er en kompakt måde at definere en property med standard implementation (direkte offentliggørelse af et privat backing-field)**

```
Public Property Number42 As Integer = 42
```

Intialisering af propertyen

Svarer til den expandede property:

```
Private _number As Integer = 42

Public Property Number() As Integer
  Get
    Return _number
  End Get
  Set(ByVal value As Integer)
    _number = value
  End Set
End Property
```

Expander en automatic property ved at:

- Stå i linjen lige efter.
- Skriv ”get”.
- Tryk Enter.

Collection initializers

VB Captator

- **Allerede eksisterende array initialisering**

```
Dim primesArray As Integer() = {2, 3, 5, 7, 11, 13, 17}
```

- **Collectionen skal implementere System.Collections.Generic.ICollection<T>**
  - Kalder Add(T)

```
Dim primtal As New List(Of Integer) From {2, 3, 5, 7, 11, 13, 17}

Dim ænder As New List(Of Person) From {
  New Person() With {.Navn = "Anders And", .Adresse = "Andeby"},
  New Person() With {.Navn = "Andersine And", .Adresse = "Andeby"},
  New Person() With {.Navn = "Fætter Vims", .Adresse = "Andeby"}}

Dim primesArrayList As New ArrayList From {2, 3, 5, 7, 11, 13, 17}

Dim værdier = New Dictionary(Of Integer, String) From
  {{0, "Første"}, {1, "Anden"}}
```

Argument til Add-metoden

Implicit line continuation

VB Captator

- **I mange situationer kan den eksplicitte line continuation character undlades**
  - Efter et komma
  - Efter en `(` `{` `<%=` eller før en `)` `}` `%>`
  - Efter operatorer: `&` `=` `+=` `<>` `And` `Is` o.s.v.
  - Efter member qualifiere (`.`) i de fleste situationer
  - Før og efter query operatorer (LINQ)
  - Efter `In` i en Foreach statement
  - Efter `From` i en collection initializer
  - m.m.

Embedded expression i en XML literal

```
Dim a,
  b As Integer

Dim tal As Double =
  System.
  Math.
  PI +
  (
  System.Math.E
  )
```

Multiline lambda udtryk

VB Captator

- **Lambda udtryk kan være på flere linjer**

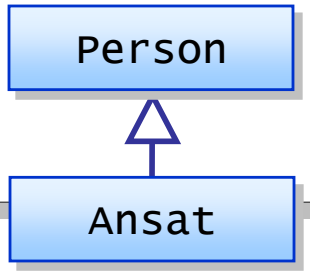
```
Dim beregnVærdi =
  Function(a As Double, b As Double, c As Double) As Double
    Dim resultat As Double
    resultat = a + b
    resultat += c
    Return resultat
  End Function

Dim værdi As Double = beregnVærdi(1, 2, 3)
```

```
Dim gørEtEllerAndet =
  Sub()
    Dim resultat As Double
    resultat = værdi ^ 2
    MsgBox(resultat.ToString())
  End Sub

gørEtEllerAndet()
```

Generisk covarians – problem i C# 3.0

Captator

```
public string GetNavne(IEnumerable<Person> personer)
{
  foreach (Person p in personer) { /* ... */ }
}
```



- **Muligt:**

```
IEnumerable<Person> personer =
  new List<Person> { new Person("Rip"), new Person("Rap") };

string personNavne = GetNavne(personer);
```

- **Da IEnumerable<Ansat> ikke nedarves fra IEnumerable<Person> giver følgende til gengæld kompilerfejl i C# 3.0**

covarians

```
IEnumerable<Ansat> ansatte =
  new List<Ansat> { new Ansat("Rip"), new Ansat("Rap") };

string ansatNavne = GetNavne(ansatte);
```

✘ Kompiler fejl

Problematisk array covarians

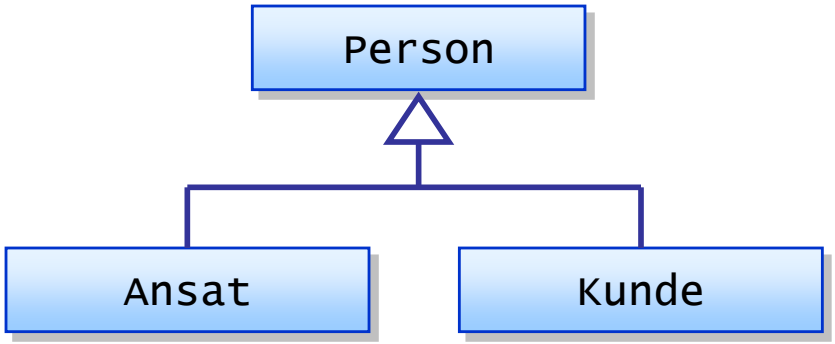
- **Problematisk <u>eksisterende</u> implementation af array covarians i C#**

```
Person[] personArray = new Ansat[10];

personArray[0] = new Ansat("Arne Ansat");
personArray[1] = new Person("Per Person");
personArray[2] = new Kunde("Kurt Kunde");
```

Accepterer alle personer

✘ Runtime fejl

- **personArray er erklæret til at kunne indeholde vilkårlige personer, men det bagvedliggende array accepterer kun ansatte**



Generisk covarians – løsningen i C# 4.0

Captator

- **For at udnytte covarians for et interface**
  - skal den generiske type angives med *out* i interfacet

```
System.Collections.Generic.IEnumerable<out T>
```

  - må den generiske type T kun indgå i outputværdier

```
System.Collections.Generic.IEnumerator<T> GetEnumerator()
```

- **Hermed er covarians mulig**

```
public string GetNavne(IEnumerable<Person> personer)
{
  foreach (Person p in personer) { /* ... */ }
}
```

```
IEnumerable<Ansat> ansatte =
    new List<Ansat> { new Ansat("Rip"), new Ansat("Rap") };

string ansatNavne = GetNavne(ansatte);
```

covarians

Generisk kontravarians – problem i C# 3.0

Captator

```
int System.Collections.Generic.IComparer<T>.Compare(T x, T y)
```

```
public int Sammenlign(IComparer<Ansat> comparer, Ansat a1, Ansat a2)
{ return comparer.Compare(a1, a2); }
```

```
class PersonComparer: IComparer<Person>
{
  int IComparer<Person>.Compare(Person x, Person y)
  { return x.Navn.CompareTo(y.Navn); }
}
```

Mangelfuld implementation

- **En personComparer kan sammenligne vilkårlige personer og burde dermed også kunne sammenligne ansatte – men ikke i C# 3.0**

```
IComparer<Ansat> ansatComparer = new AnsatComparer();
int result3 = SammenlignAnsatte(ansatComparer, ansat, ansat);

IComparer<Person> personComparer = new PersonComparer();
int result4 = Sammenlign(personComparer, ansat, ansat);
```

✗ Kompiler fejl

kontravarians

Generisk kontravarians – løsningen i C# 4.0

Captator

- **For at udnytte kontravarians for et interface**
  - skal den generiske type angives med *in* i interfacet

```
System.Collections.Generic.IComparer<in T>
```

  - må den generiske type T kun indgå i <u>indgående</u> parametre

```
int Compare(T x, T y)
```

- **Hermed er kontravarians mulig**

```
public int Sammenlign(IComparer<Ansat> comparer, Ansat a1, Ansat a2)
{
  return comparer.Compare(a1, a2);
}
```

```
int result4 = Sammenlign(personComparer, ansat, ansat);
```

kontravarians

Generisk co- og kontravarians

- **Kun interfaces og delegates kan benytte generisk co- og kontravarians**
- **Overtrædes *in* og *out* medfører det kompilerfejl**
- **Varians virker kun ved reference typecasts – ikke ved boxing**
- **Både in og out typer kan være repræsenteret**

```
System.Func<in T,out TResult>
```

  - T bliver kun brugt i indgående parametre
  - TResult bliver kun brugt udgående
- **En propertys type**
  - er output-safe hvis der *kun* er en get-accessor
  - er input-safe, hvis der *kun* er en set-accessor

Den dynamiske type

- **C#s dynamiske type kan med fordel bruges, når den konkrete type ikke kendes på kompilerings-tidspunktet (som ved reflektion)**

```
public static T GetPropertyValue<T>(object obj, string propertyName)
{
  PropertyInfo property = obj.GetType().GetProperty(propertyName);
  return (T) property.GetValue(obj, null);
}
```

```
private object GetObject() {
  return "The Dynamic Duo";
}
```

```
int len1 = GetString().Length;

object text = GetObject();
int len2 = GetPropertyValue<int>(text, "Length");

dynamic d = GetObject();
int len3 = d.Length;
```

Den dynamiske type

- **dynamic er en C# type, der ikke er underlagt statisk typecheck – typen findes ikke på runtime**
- **Kald af ikke-eksisterende members vil resultere i run-time fejl (RuntimeBinderException)**
- **Kompileres til System.Object**

```
dynamic d;

d = "The Dynamic Duo";
int len = d.Length;

d = new Person();
d.Navn = "Batman";
string navn = d.Navn;

d.SayHi();
(local variable) dynamic d

string eksistererEj = d.Adresse;
```

Disassembler

```
.method private hidebysig instance void ButtonDynamic_Click(object sende
{
  .maxstack 9
  .locals init (
    [0] object d,
    [1] int32 len,
    [2] string navn,
    [3] string eksistererEj,
```

Tooltip

✘ Runtime fejl

Den dynamiske type

Captator

- **Der er implicit konvertering *til* en dynamisk type**

```
dynamic i1 = 42;
```

- **Der er implicit konvertering *fra* en dynamisk type**

```
int i2 = i1;
```

- **Konvertering til en forkert type vil resultere i run-time fejl (RuntimeBinderException)**

```
string s = i1;
```

✘ Runtime fejl

- **Hvis et argument eller en parameter til en metode er dynamic, sker overload resolution på run-time**

Den dynamiske type

Captator

- **dynamic kan bruges**
  - som type for members, lokale variable, parametre
  - i generiske typer etc.

```
var elements = new List<dynamic>();
elements.Add(42);
dynamic element = elements[0];
System.Type isListOfObject = typeof(List<dynamic>);
```

  - sammen med *is* og *as* operatorerne

```
dynamic x1 = 17;
int x2 = 42;

bool isTrue = x1 is dynamic;
bool isAlsoTrue = x1 is int;

dynamic is17 = x1 as dynamic;
dynamic is42 = x2 as dynamic;
```

Vil altid være true hvis x1 != null

Den dynamiske type

- **På kompileringstidspunktet**
  - ændres typen dynamic til System.Object
  - genereres kode der baseret på kaldsinformationen identificerer operationen samt kalder den
- **På runtimetidspunktet**
  - hvis objektet er et COM objekt benyttes IDispatch
  - hvis objektet implementerer IDynamicObject kaldes dette
  - hvis objektet er et almindeligt .NET objekt benyttes Microsoft.CSharp.RuntimeBinder til at finde og binde til den korrekte metode
  - hvis metoden ikke kan findes kastes exception

Den dynamiske type

Captator

- **Dynamiske opslag kan ikke finde extension metoder**
- **Lambda udtryk kan ikke være argument til metodekald på dynamiske variable**

```
public class Person
{
  public void Call(System.Action action)
  {
    action();
  }
}
```

```
dynamic d = new Person();

d.Call(() => MessageBox.Show("I am calling you!"));
```

✗ Kompiler fejl

- **Men det kan en typestærk delegate godt**

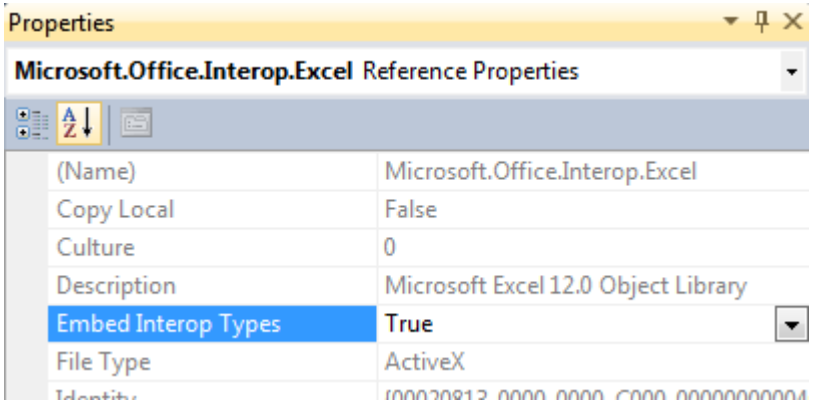
```
d.Call((Action)delegate { MessageBox.Show("I am calling you!"); });
```

Den dynamiske type

- **Normalt ikke et godt alternativ hvis man *kan* arbejde typestærkt**
- **Dog kan den bruges, hvis man ønsker at undgå eksplicitte typecasts**
  - Giver kortfattet men mere usikker kode
- **Hvis man foreacher over en dynamic, antages den at implementere IEnumerable**
- **WPF understøtter databinding til dynamiske objekter**
- **Kan bruges ved f.eks. anonyme view klasser**

COM support

- **Microsoft specifikke udvidelser af C#**
- **Kald af COM metoder må kalde ref parametre by value (uden at angive ref keywordet i kaldet)**
  - Kompileren laver en temporær variabel
- **Kan embedde (linke) anvendte interop typer fra PIAer i klient assemblyen**
  - Fylder mindre
  - Versionering/duck-typing



- **COM-metode, der returnerer et object, vil, hvis komponenten linkes (ikke refereres), i stedet returnere dynamic – slipper for typecasts**

COM support

- **C# 3.0 understøtter ikke indexed properties**

```
var excelApp = new Excel.Application();
Excel.Range targetRange = excelApp.get_Range("A1", Type.Missing);
```

- **Men i C# 4.0 understøttes direkte brug af COM komponenters indexed properties**

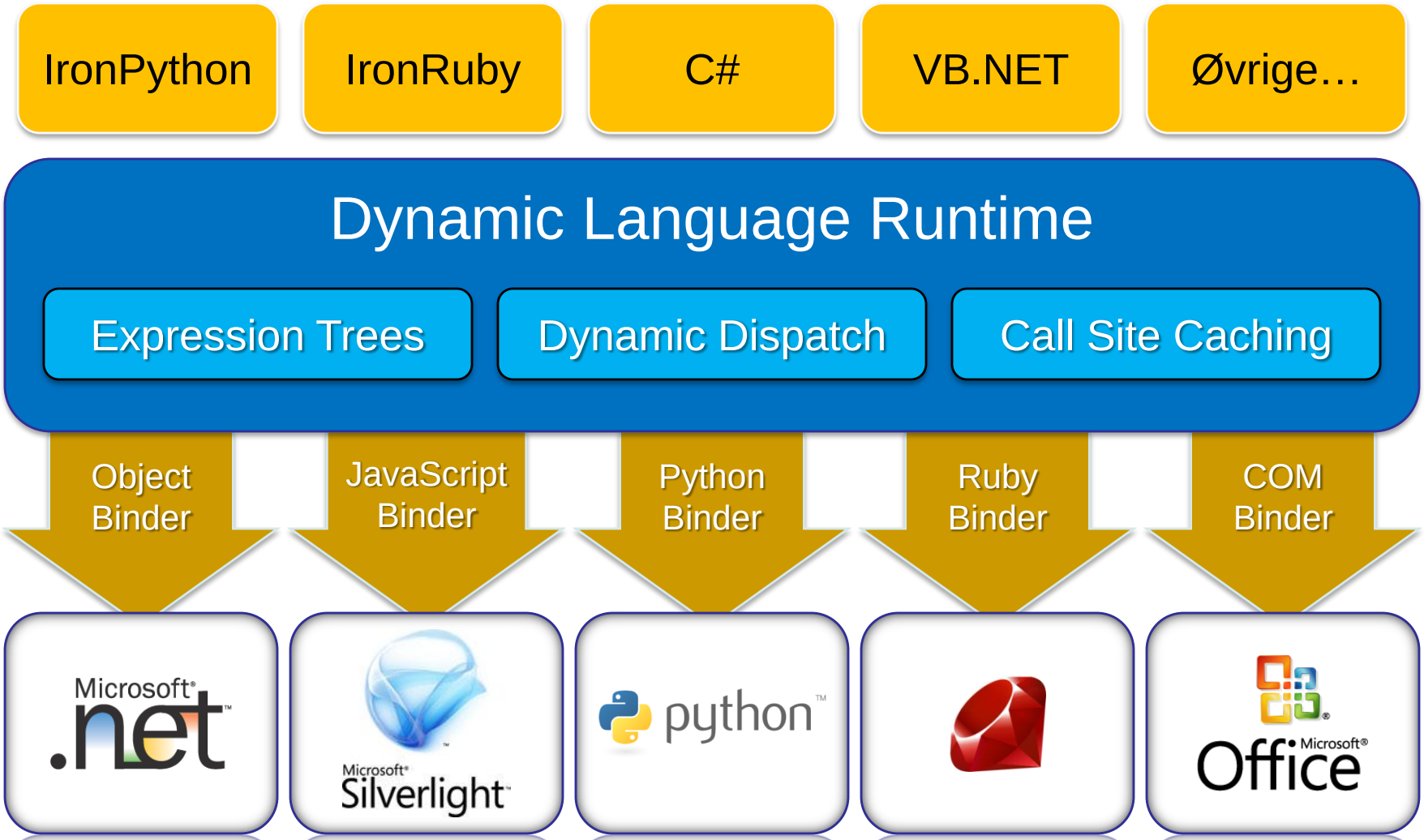
```
var excelApp = new Excel.Application();
Excel.Range targetRange = excelApp.Range["A1"];
```

- **Indexed properties understøttes dog ikke *generelt* i C#**

DLR'en baserer sig på CLR'en

- **CLR: Common Language Runtime**
  - Fælles platform for statiske sprog
- **DLR: Dynamic Language Runtime**
  - Fælles platform for dynamiske sprog
  - Formål:
    - Understøtte dynamiske sprog i .NET
    - Give statiske sprog dynamiske funktioner
  - Hurtigere kald af dynamiske members end ved almindelig reflektion

Dynamiske sprog baseret på .NET



ExpandoObject

- **System.Dynamic.ExpandoObject i System.Core.dll**
- **Implementerer:**
  - IDynamicMetaObjectProvider
  - IDictionary<string,object>
    - IEnumerable<KeyValuePair<String,Object>>
  - INotifyPropertyChanged
- **Properties kan tilføjes, fjernes og enumereres**
  - Gemmes i et dictionary
- **Kan bruges som et alternativ til et traditionelt Dictionary**

ExpandoObject

- **Properties kan tilføjes**

```
dynamic and = new System.Dynamic.ExpandoObject();

and.Navn = "Anders And";
and.Alder = 42;
and.Adresse = "Paradisæblevej 111";

string s = and.Navn + " " + and.Alder;

and.Kæreste = new System.Dynamic.ExpandoObject();
and.Kæreste.Navn = "Andersine And";

and.Kæreste.Niecer = new List<dynamic>();
and.Kæreste.Niecer.Add(new System.Dynamic.ExpandoObject());
and.Kæreste.Niecer[0].Navn = "Kylle";
and.Kæreste.Niecer.Add(new System.Dynamic.ExpandoObject());
and.Kæreste.Niecer[1].Navn = "Pylle";
and.Kæreste.Niecer.Add(new System.Dynamic.ExpandoObject());
and.Kæreste.Niecer[2].Navn = "Rylle";

int antalNiecer = and.Kæreste.Niecer.Count;
```

ExpandoObject

- **Properties kan enumereres**

```
dynamic expando = new System.Dynamic.ExpandoObject();

string s = "";

foreach (KeyValuePair<string,object> property in
                                (IDictionary<string, object>)expando)
{
  s += property.Key + "=" + property.Value.ToString();
}
```

- **Properties kan fjernes**

```
((IDictionary<string, object>)expando).Remove("Adresse");
```

DynamicObject

- **System.Dynamic.DynamicObject i System.Core.dll**
- **Implementerer IDynamicMetaObjectProvider**
- **Kan implementere custom memberhåndtering**
- **Eksempler på interessante members**
  - TryGetMember, TrySetMember
  - TryGetIndex, TrySetIndex
  - TryInvoke, TryInvokeMember
  - GetDynamicMemberNames
    - Bruges af debuggeren, *kan* bruges af egen kode

DynamicObject - eksempel

Captator

- **DataRowObject: Indpakning af DataRow med property-tilgang til columns**

Klassisk DataRow

```
string adresse1 = (string) row["Adresse"];

string adresse2 = dynaRow.Adresse;
```

DynamicObject

```
public class DataRowObject : System.Dynamic.DynamicObject
{
  public DataRowObject(System.Data.DataRow row) {
    _row = row;
  }

  private System.Data.DataRow _row;

  // ...
```

DataRowObject

```
System.Data.DataTable table = CreateTable();
System.Data.DataRow row = table.Rows[0];

dynamic dynaRow = new DataRowObject(row);
```

Klientkode

DynamicObject - eksempel

Captator

- **Property-tilgang til columns**

Klientkode

```
dynaRow.Adresse = "Paradisæblevej 111";
string dynaString = dynaRow.Id.ToString() + " - " + dynaRow.Navn;
```

- **Implementation:**

```
override bool TrySetMember(SetMemberBinder binder, object value)
{
  _row[binder.Name] = value;
  return true;
}
```

DataRowObject

```
override bool TryGetMember(GetMemberBinder binder,
                                                  out object result)
{
  result = _row[binder.Name];
  return true;
}
```

DataRowObject

- **binder.Name kan naturligvis mappes/modificeres**

DynamicObject - eksempel

Captator

- **Indexeret-tilgang til columns**

```
dynaRow[0] = 117;
int id1 = dynaRow[0];
```

Klientkode

- **Implementation:**

```
override bool TryGetIndex(GetIndexBinder binder, object[] indexes,
                                                  out object result)
{
  result = _row[(int)indexes[0]];
  return true;
}
```

DataRowObject

```
override bool TrySetIndex(SetIndexBinder binder, object[] indexes,
                                                  object value)
{
  _row[index] = value;
  return true;
 }
```

DataRowObject

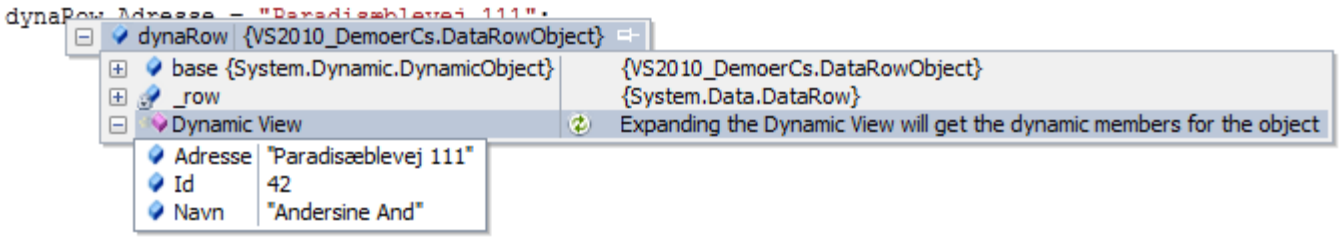
- ***indexes* type(r) kan varieres og overloades frit**

DynamicObject - eksempel

- **Opremsning af membernavne**
  - Skal implementeres for at få debuggersupport

```
override IEnumerable<string> GetDynamicMemberNames()
{
  var names = new System.Collections.Generic.List<string>();

  foreach (System.Data.DataColumn column in _row.Table.Columns) {
    names.Add(column.ColumnName);
  }
  return names;
}
```

DataRowObject

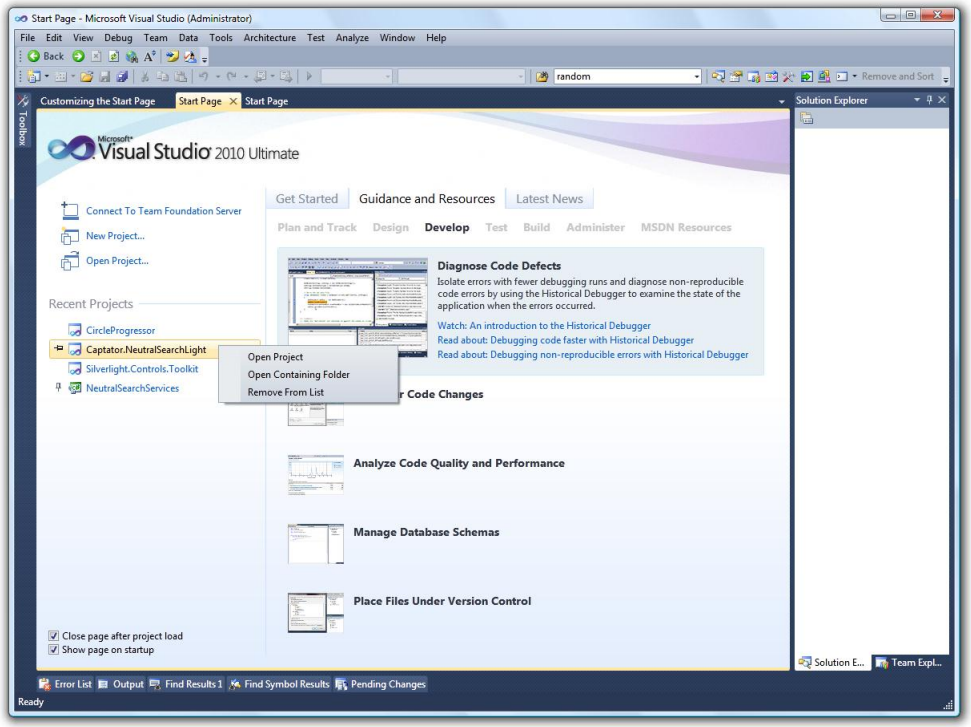


Klientkode

```
IEnumerable<string> names =
    ((DynamicObject)dynaRow).GetDynamicMemberNames()
```

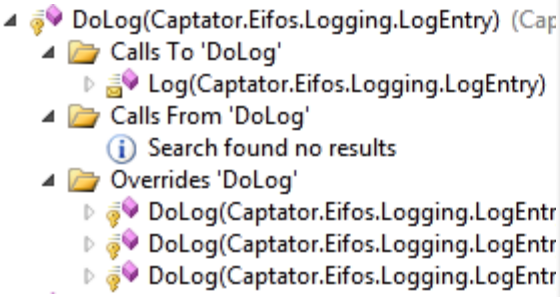
Start Page

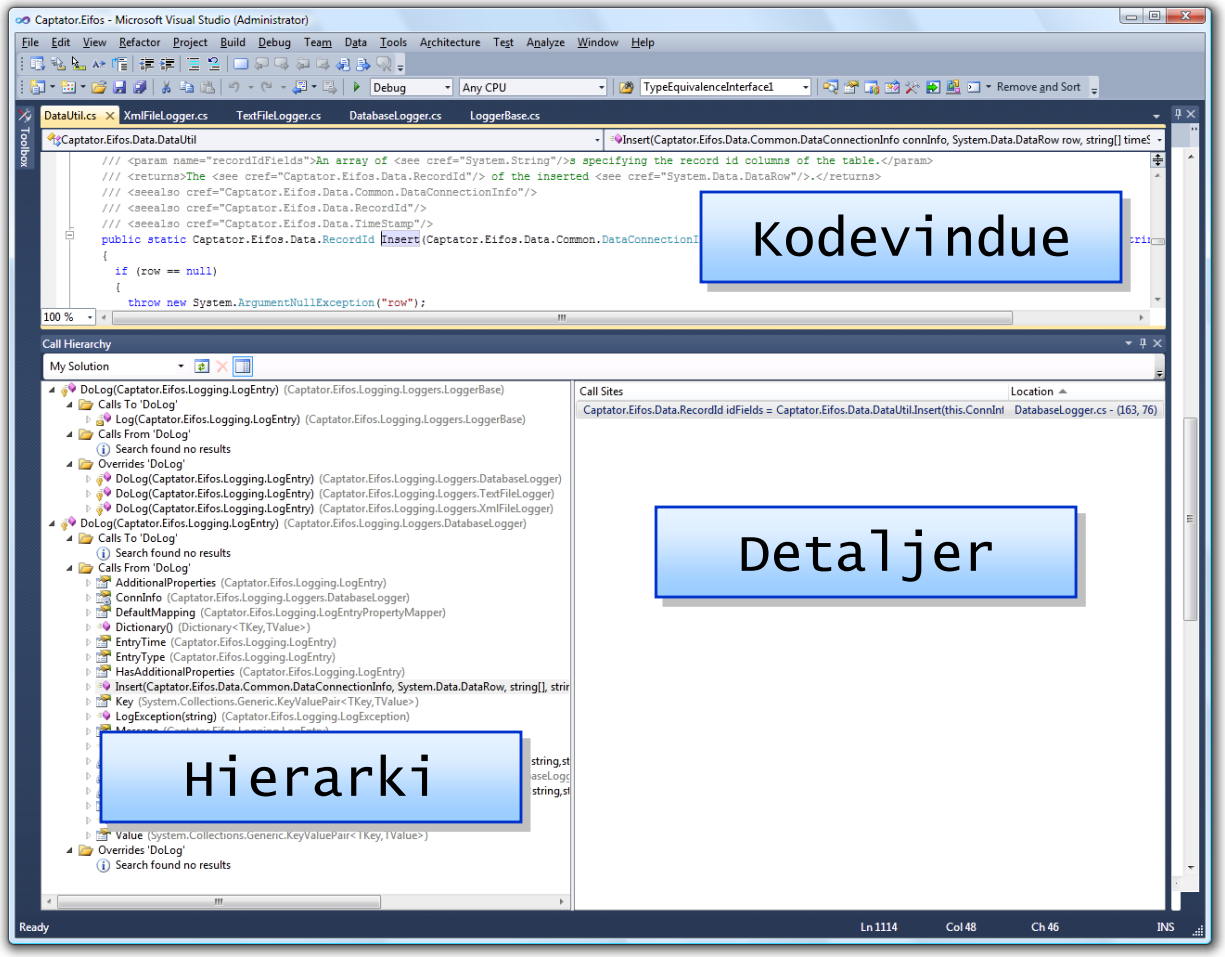
- **Pin og remove recent projects**
- **Kan ændres**
- **Defineret i XAML: StartPage.xaml, StartPage.csproj i *user\Documents*\Visual Studio 10\StartPages**



Call Hierarchy

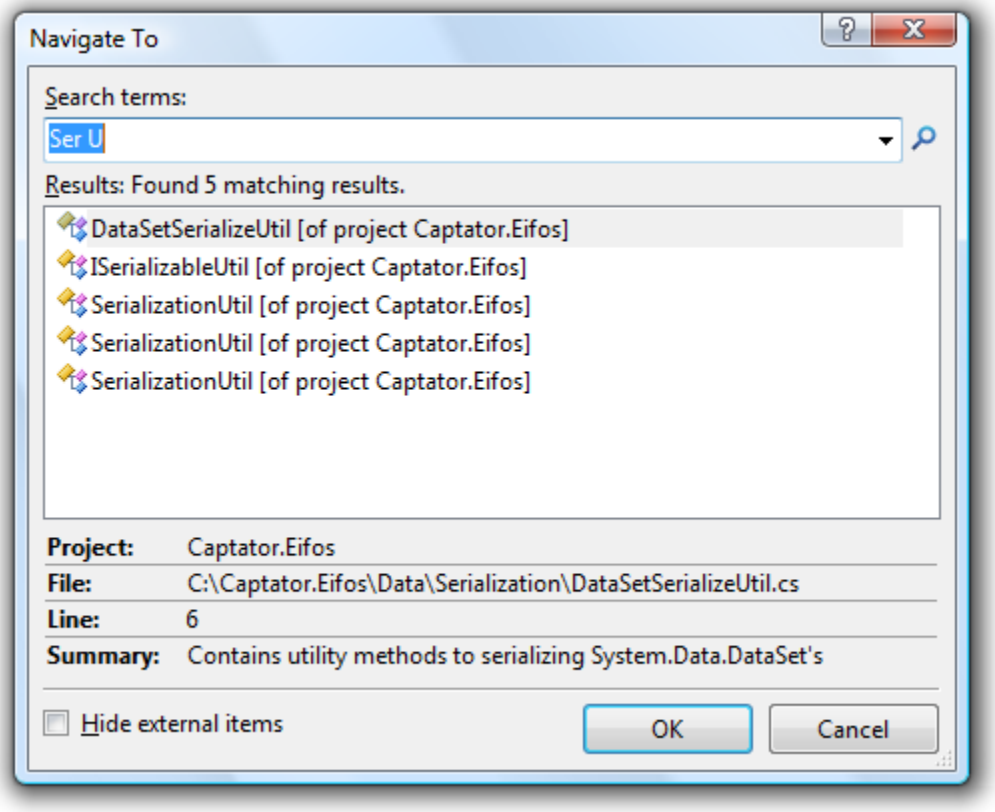
- **Viser kaldshierarkiet for en member**
  - Kald til og fra memberen samt overstyringer af den





Navigate To

- **Søger efter symboler udfra delstrenge**
  - Finder ikke lokale variable eller namespaces
  - Hvis søgestrengen indeholder uppercase tegn søges casesensitivt
  - Space fungerer som *and*
  - Akronymsøgning (f.eks. SU)
- Shortcut: ctrl+,
- Dobbeltklik eller OK går til symbolet



Reference Highlighting

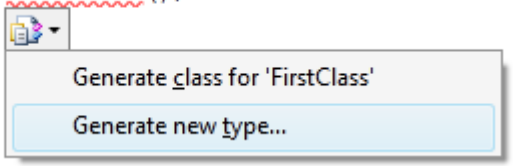
- **Valg af symbol highlighter alle forekomster af symbolet i samme fil**
- **Næste forekomst: ctrl+shift+pilned**
- **Forrige forekomst: ctrl+shift+pilop**

```
public static string EnsureEndingString(string source, string endsWith)
{
  if(source == null)
  {
    throw new System.ArgumentNullException("source");
  }

  if(endsWith == null)
  {
    throw new System.ArgumentNullException("endsWith");
  }

  if(source.EndsWith(endsWith))
  {
    return source;
  }
  else
  {
    return source + endsWith;
  }
}
```

Generate From Usage

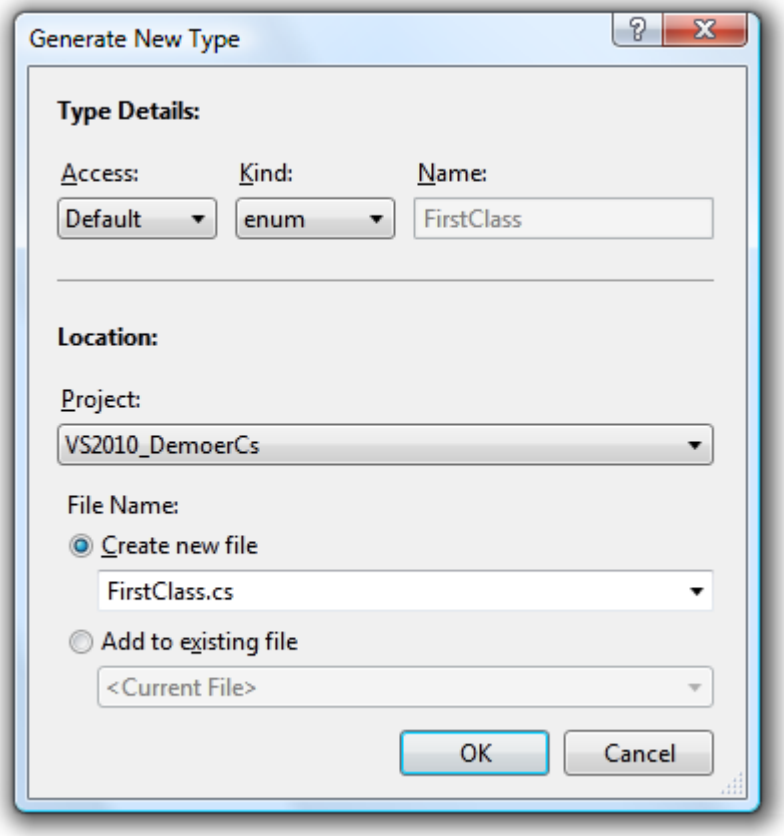
- **Kan generere en stub for en type eller konstruktør udfra en anvendelse**

```
var x = new FirstClass();
```



- **Kan allerede generere members med parametererklæringer**

```
using System;

namespace VS2010_DemoerCs
{
  class FirstClass
  {
    private int initialValue;

    public FirstClass(int initialValue)
    {
      // TODO: Complete member initialization
      this.initialValue = initialValue;
    }

    public int Value { get; set; }

    internal void TheMethod(int newValue, string parameter)
    {
      throw new NotImplementedException();
    }
  }
}
```

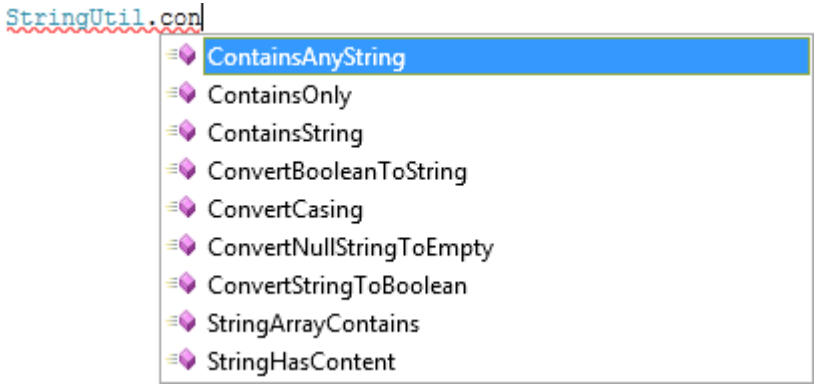


Completion og suggestion modes

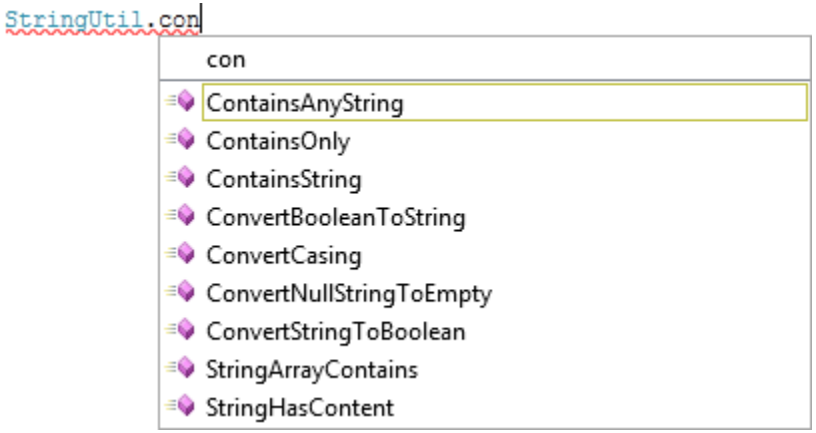
Captator

- **Skifter imellem de to modes med ctrl+alt+space**

Completion mode



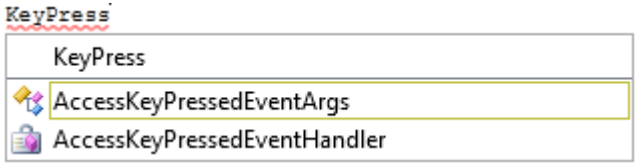
Suggestion mode

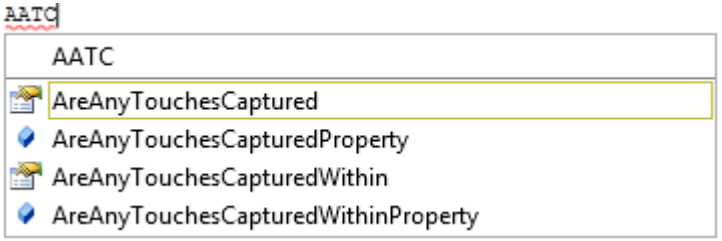


  - I completion mode indsættes den valgte identifier ved et identifier-afsluttende-tastetryk
  - I suggestionmode skal der trykkes på *tab* eller identifieren skal vælges med pilop/pilned

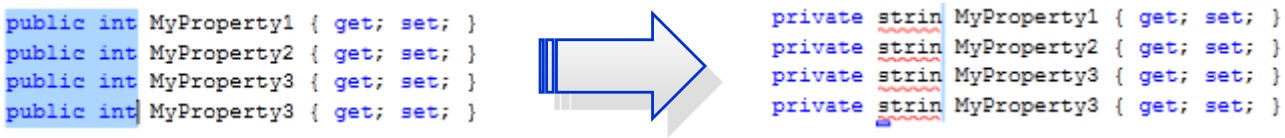
Intellisense

- **Intellisense**
  - Understøtter delstrengs- og akronymfiltrering



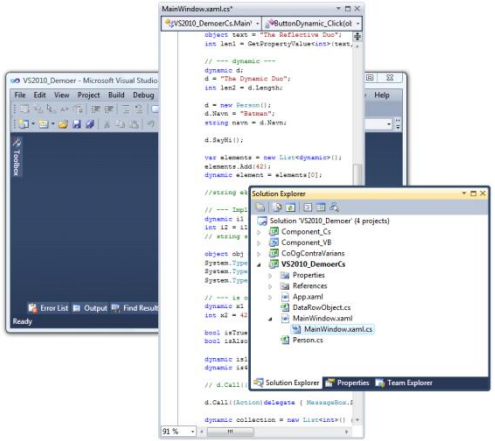


- **Box-selection**
  - VS2008 feature
    - Et kode-rektangel (box) kan vælges med *alt*-selektion
  - VS2010 feature
    - Hvis der tastes/pastes ind i en box selektion, erstatter det indsatte det valgte på hver linje (gentages)
    - En box med en bredde på nul karakterer kan bruges til indsættelse (uden at erstatte eksisterende tekst)

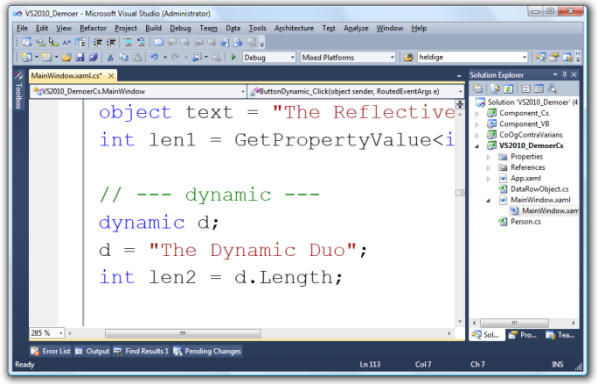


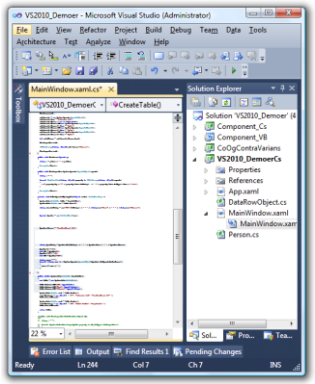
Diverse nye Visual Studio funktioner

- **Vinduer kan flyttes overalt på desktoppen**
  - Også til anden monitor



- **Kodevinduer kan zommes med ctrl+mousewheel**





- **References dialogen:**
  - Den senest valgte tab vises initielt

Debugging

- "Hover over"-quickwatch-tooltips kan pinnes

```
var watch = new System.Diagnostics.Stopwatch();
watch.Start();

int justAnInt32 = 0;
for (int index = 0; index < loopCount; index++)
{
  justAnInt32++;
}

watch.Stop();
long justAnInt32Time = watch.ElapsedMilliseconds;

watch.Restart();

System.Numerics.BigInteger aBiggie = 0;
for (int index = 0; index < loopCount; index++)
{
  aBiggie++;
}
```

watch.Elapsed.TotalSeconds 129.8707474

index 42

justAnInt32 10000000

Type a comment here

Unpin from source

justAnInt32Time 31

index 42

Hvilken værdi er dette helt præcist?

loopCount 10000000

Denne værdi ændrer sig tilsyneladende slet ikke ?!?

- Kan kommenteres og flyttes

BigInteger og Complex

Captator

- **System.Numerics.BigInteger (System.Numerics.dll)**
  - Vilkårlig stor integer
  - Mange af de sædvanlige matematiske og bitvise operationer
  - Struct, immutable – pas på med performance!
  - Konvertering til og fra bytearrays

```
var number = System.Numerics.BigInteger.Pow(42, 117);

byte[] bytes = number.ToByteArray();

var newNumber = new System.Numerics.BigInteger(bytes);
```

  - Pas på med ekstern håndtering af byte arrayet (studér formatet!)
- **System.Numerics.Complex (System.Numerics.dll)**

Tuple

- System.Tuple (mscorlib.dll)
- Immutable collection med fast størrelse
- Kan indeholde forskellige stærkt-typede objekter
- Create factory-metode – standard op til 7 elementer

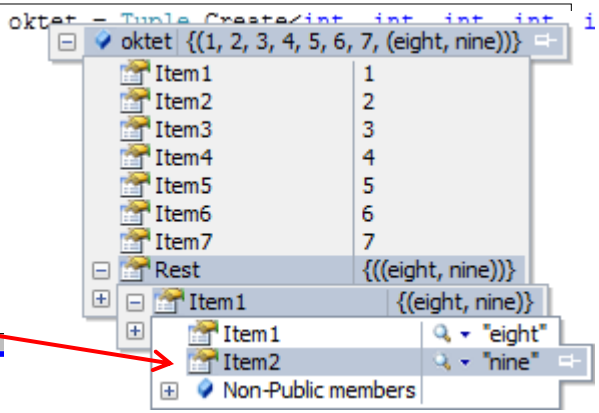
```
var trio = Tuple.Create<int, int, string>(17, 42, "fjel");
int first = trio.Item1;
trio = new Tuple<int, int, string>(trio.Item1, trio.Item2, "fejl");

var trio2 = Tuple.Create(17, 42, "fejl");
bool isFalse = trio == trio2;
```

Typeinferens

Reference sammenligning

- Mere end 7 elementer via nesting

```
var oktet = Tuple.Create<int, int, int, int,
int, int, int, Tuple<string, string>>
(1, 2, 3, 4, 5, 6, 7,
Tuple.Create<string,string>("eight", "nine"));

string isNine = oktet.Rest.Item1.Item2;
```



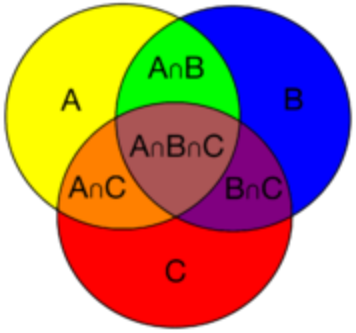
Diverse nyheder i frameworket

- **Func og Action delegatetyper med op til 16 parametre**
- **Nye filsystem enumereringer:**
  - (Directory|DirectoryInfo).EnumerateDirectories, (Directory|DirectoryInfo).EnumerateFiles, Directory.EnumerateFileSystemEntries, DirectoryInfo.EnumerateFileSystemInfos, File.ReadLines
- **<enum>.HasFlag metode og Enum.TryParse**

```
BindingFlags binding = BindingFlags.Public | BindingFlags.Instance;

bool isTrue = binding.HasFlag(BindingFlags.Instance);
bool isFalse = binding.HasFlag(BindingFlags.Static);

System.Reflection.BindingFlags result;

bool succeeded = Enum.TryParse<BindingFlags>("Public", out result);
bool failed = Enum.TryParse<BindingFlags>("Private", out result);
```

SortedSet

- **SortedSet**
  - Sorteret collection – dubletter ignoreres
  - Almindelige collection members
  - Inkluderer mængdeoperationer
    - ExceptWith, GetViewBetween, IntersectWith, IsProperSubsetOf, IsSubsetOf, IsSupersetOf, Overlaps, RemoveWhere, SetEquals, SymmetricExceptWith, UnionWith, Max, Min

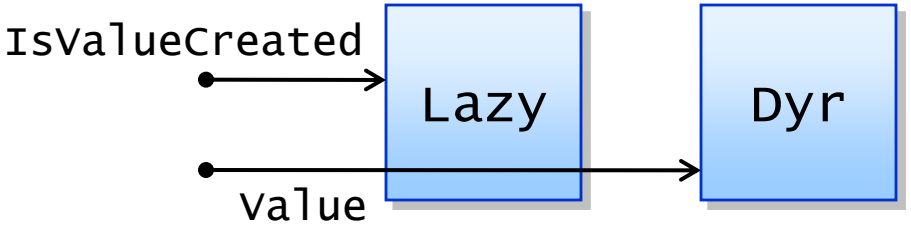



```
var mængde = new SortedSet<string>() { "q", "w", "r", "t", "y" };
mængde.Add("q");

SortedSet<string> delmængde = mængde.GetViewBetween("q", "t");

var mængde2 = new SortedSet<string>() { "t", "y", "k", "m", "æ" };
mængde.IntersectWith(mængde2);
int is2 = mængde.Count;
bool isTrue = mængde.IsProperSubsetOf(mængde2);
```

Lazy

- System.Lazy<T> (mscorlib.dll)
- Implementerer lazy instansiering

```
var dovendyr = new Lazy<Dyr>();

bool isFalse = dovendyr.IsValueCreated;

Dyr dyret = dovendyr.Value;
MessageBox.Show("Dovendyret hedder nu: " + dyret.Navn);

bool isTrue = dovendyr.IsValueCreated;
```



- Kan fodres med en instansieringsfunktion

```
var dovendyr = new Lazy<Dyr>(() => new Dyr("Ole Lukøje"));
```

Task Parallel Library

Captator

- System.Threading, System.Threading.Tasks
- Tasks håndterer trådallokeringen
- Asynkron eksekvering

```
Task<string> task = Task<string>.Factory
    .StartNew(s => GetString(s), "1");

task.ContinueWith(t => ShowText(task.Result));
```

- Synkroniseringen af tasks – eksempel

```
Task<string> task1 = Task<string>.Factory
    .StartNew(s => GetString(s), "1");
Task<string> task2 = Task<string>.Factory
    .StartNew(s => GetString(s), "2");

Task.Factory.ContinueWhenAll(new Task[] { task1, task2 }, tasks =>
{
  foreach (Task<string> task in tasks) {
    ShowText(task.Result);
  }
});
```

Parallel-klassen

- .Invoke kalder en række operationer parallelt

```
Tasks.Parallel.Invoke(() => Calculate(17), () => Calculate(42));
```

- .For definerer et antal gennemløb for hver af hvilke, der parallelt kaldes en operation

```
Tasks.Parallel.For(1, 11, (d) => Calculate(d));
```

- .ForEach kalder parallelt en operation for hvert element i sourcen (collection etc.)

```
var doubles = new List<double>() { 17, 42 };

Tasks.Parallel.ForEach<double>(doubles, (d) => Calculate(d));
```

Parallel LINQ (PLINQ)

Captator

- **Fungerer tilsvarende LINQ to Objects**
- **System.Linq.ParallelEnumerable**
- **Members (langt de fleste er extension metoder)**
  - Aggregate, All, Any, AsEnumerable, AsOrdered, AsParallel, AsSequential, AsUnordered, Average, Cast, Concat, Contains, Count, DefaultIfEmpty, Distinct, ElementAt, ElementAtOrDefault, Empty, Equals, Except, First, FirstOrDefault, ForAll, GetHashCode, GetType, GroupBy, GroupJoin, Intersect, Join, Last, LastOrDefault, LongCount, Max, Min, OfType, OrderBy, OrderByDescending, Range, Repeat, Reverse, Select, SelectMany, SequenceEqual, Single, SingleOrDefault, Skip, SkipWhile, Sum, Take, TakeWhile, ThenBy, ThenByDescending, ToArray, ToDictionary, ToList, ToLookup, ToString, Union, Where, WithCancellation, WithDegreeOfParallelism, WithExecutionMode, WithMergeOptions, Zip

Parallel LINQ (PLINQ)

Captator

- **Man *vælger* at eksekvere queries parallelt**

```
var numbers = Enumerable.Range(17, 42);

var query2 = from number in numbers.AsParallel()
             where IsEven(number)
             select number;
```

- **PLINQ bruger som default alle processorer**
  - WithDegreeOfParallelism kan begrænse antallet
- **ForAll har tilknyttet en continuation til hvert element**

```
string s = "";
var numbers = Enumerable.Range(17, 42);

var query = from number in numbers.AsParallel()
            where IsEven(number)
            select number;

query.ForAll((item) => s+= item.ToString() + " ");
```

Microsoft Visual Studio 2010 Ultimate

- IntelliTrace™
- UML Modeling
- Architecture Explorer
- Logical Class Designer
- Load Testing

Test and Lab Manager

- Test Case Management
- Manual Testing
- Test Record & Playback
- Layer Diagram
- Web Testing

Microsoft Visual Studio 2010 Premium

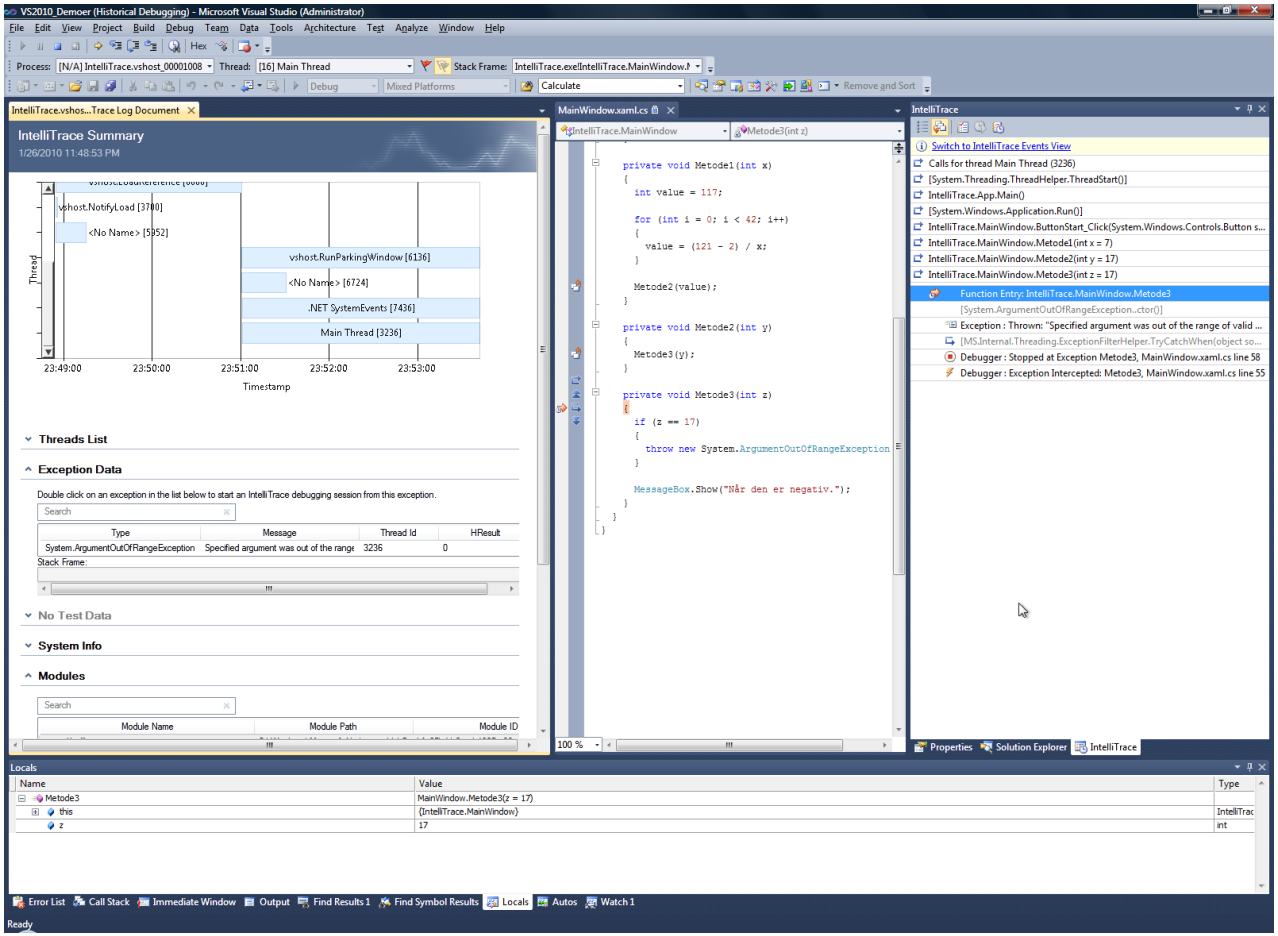
- UI Test Automation
- Performance Profiling
- Code Coverage
- Database Change Mgmt
- Database Unit Testing
- Test Impact Analysis
- Static Code Analysis
- Code Metrics
- Database Deployment
- Test Data Generation

Microsoft Visual Studio 2010 Professional

- Silverlight Tools
- SharePoint Development
- Web Development
- Generate from Usage
- New WPF Editor
- Multi-core Development
- Cloud Development
- Windows Development
- Office Development
- Customizable IDE

IntelliTrace

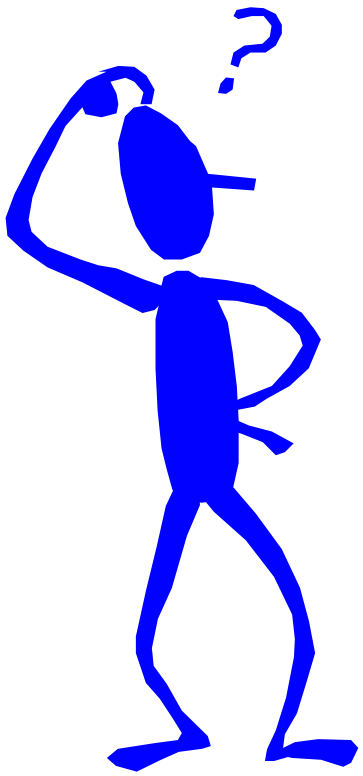
- "Baglæns debugging" under almindelig debugging
- Debugging i gemte TraceDebugging filer (.tdlog-filer)



Hvilke yderligere nyheder er der?

- Visual Studio Team System 2010 - TFS
- MEF
- Code contracts
- Memorymapped files
- ClickOnce, WCF og WF
- ASP.NET
- WPF
- Office
- VS add-ins
- Silverlight designer
- Entity Frameworket
- ... samt sikkert en hel masse andet...

Spørgsmål



www.captator.dk

nyheder, artikler, information, ...