



## What's New In C# 4 ?



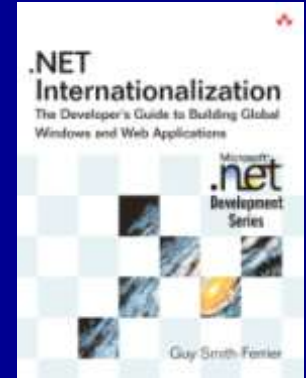
Guy Smith-Ferrier

[guy@guysmithferrier.com](mailto:guy@guysmithferrier.com)

Blog: <http://www.guysmithferrier.com>

# About...

- Author of .NET Internationalization
  - Visit <http://www.dotneti18n.com> to download the complete source code
- The .NET Developer Network
  - <http://www.dotnetdevnet.com>
  - Free user group for .NET developers, architects and IT Pros based in Bristol



# Agenda

- Dynamically Typed Objects
- Optional And Named Parameters
- Improvements To COM Support
- Co- and contra-variance
- Beyond C# 4

# Current Trends In Development Languages

- Concurrency
- Declarative Languages
- Dynamic Languages

# Dynamic Type

```
dynamic myInt = 1;  
myInt = "Hello world";
```

```
object myInt = 1;  
myInt = "Hello world";
```

```
CultureInfo cultureInfo;  
  
dynamic myVar = 1033;  
cultureInfo = CultureInfo.GetCultureInfo(myVar);  
  
myVar = "en-US";  
cultureInfo = CultureInfo.GetCultureInfo(myVar);  
  
System.Console.WriteLine(cultureInfo.Name);
```

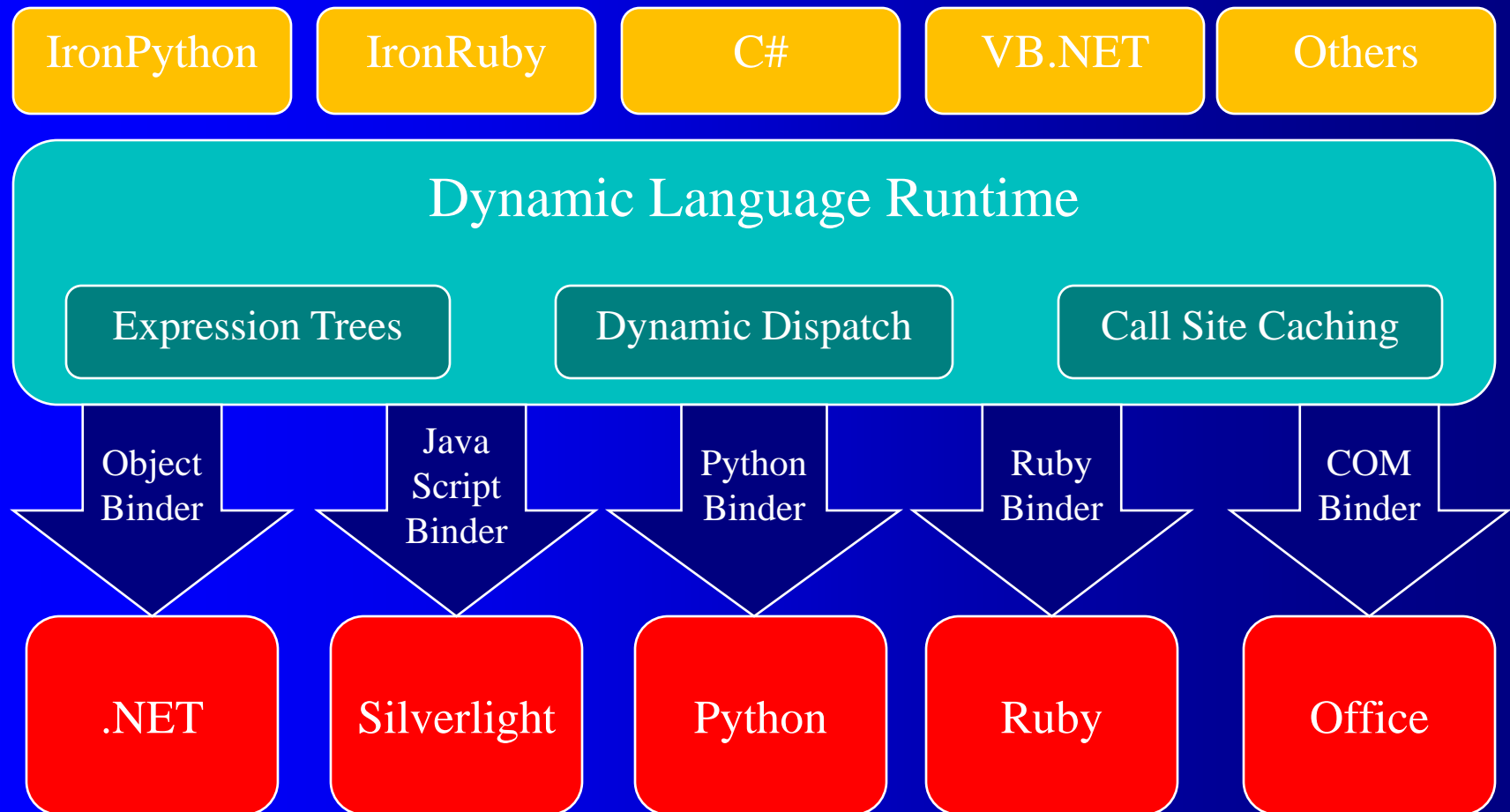
# Consuming JavaScript Dynamic Types From Silverlight (1 of 2)

```
public partial class Page1 : Page
{
    HtmlWindow htmlWindow = HtmlPage.Window;
    public void Init()
    {
        htmlWindow.Invoke("CreateMap");
    }
    void Customers_SelectionChanged(
        object sender, SelectionChangedEventArgs args)
    {
        Customer customer = customers.SelectedItem as Customer;
        HtmlDocument htmlDocument = HtmlPage.Document;
        htmlDocument.SetProperty("Title", customer.Name);
        htmlWindow.Invoke("UpdateMap",
            customer.Latitude, customer.Longitude);
    }
}
```

# Consuming JavaScript Dynamic Types From Silverlight (2 of 2)

```
public partial class Page1 : Page
{
    dynamic htmlWindow = HtmlPage.Window.AsDynamic();
    public void Init()
    {
        htmlWindow.CreateMap();
    }
    void Customers_SelectionChanged(
        object sender, SelectionChangedEventArgs args)
    {
        Customer customer = customers.SelectedItem as Customer;
        dynamic htmlDocument = HtmlPage.Document.AsDynamic();
        htmlDocument.Title = customer.Name;
        htmlWindow.UpdateMap(customer.Latitude, customer.Longitude);
    }
}
```

# Support For Dynamic Languages In The .NET Framework 4.0



# Implementing IDynamicObject

## The Goal

```
static void Main(string[] args)
{
    dynamic resource =
        new DynamicallyTypedResXResource("Resource1.resx");

    Console.WriteLine(resource.HelloWorld);
}
```

```
static void Main(string[] args)
{
    dynamic resource =
        new DynamicallyTypedDbResource("Resource1");

    Console.WriteLine(resource.HelloWorld);
}
```

# Implementing IDynamicObject

## The Solution

```
public class DynamicallyTypedResXResource: DynamicObject
{
    Dictionary<string, object> dictionary = new Dictionary<string, object>();

    public DynamicallyTypedResXResource(string resXFilename)
    {
        ResXResourceReader reader = new ResXResourceReader(resXFilename);
        IEnumerator enumerator = reader.GetEnumerator();
        while (enumerator.MoveNext())
        {
            DictionaryEntry entry = (DictionaryEntry) enumerator.Current;
            dictionary.Add(entry.Key.ToString(), entry.Value);
        }
    }

    public override object GetMember(GetMemberAction action)
    {
        return dictionary[action.Name];
    }
}
```

"HelloWorld"

# Benefits Of Dynamic Typing

- Program against dynamically typed languages using the same syntax as for static languages
  - e.g. JavaScript from Silverlight
  - Write code in .NET that traditionally has been written in other languages (e.g. JavaScript)
- Program against dynamic structures using the same syntax as for static structures
  - e.g. XML, DataSets

# Optional Parameters (1 of 2)

```
static void JoinUserGroup(string userGroupName, bool isFree,  
bool hasGreatSpeakers, bool hasSwag)  
{  
}
```

```
static void JoinUserGroup(string userGroupName, bool isFree,  
bool hasGreatSpeakers)  
{  
}  
static void JoinUserGroup(string userGroupName, bool isFree)  
{  
}  
static void JoinUserGroup(string userGroupName)  
{  
}
```

# Optional Parameters (2 of 2)

```
static void JoinUserGroup(  
    string userGroupName,  
    bool isFree = true,  
    bool hasGreatSpeakers = true,  
    bool hasSwag = true)  
{  
}  
  
static void Main(string[] args)  
{  
    JoinUserGroup("DotNetDevNet", true, true, true);  
    JoinUserGroup("DotNetDevNet", true, true);  
    JoinUserGroup("DotNetDevNet", true);  
    JoinUserGroup("DotNetDevNet");  
}
```

# Named Parameters

```
static void Main(string[] args)
{
    JoinUserGroup("NxtGen", isFree: false);

    JoinUserGroup("DotNetDevNet",
        hasSwag: true, isFree: true, hasGreatSpeakers: true);
}
```

```
static void Main(string[] args)
```

```
{
```

```
    JoinUserGroup("DotNetDevNet", hasSwag: true, h
```

```
}
```

```
void Program.JoinUserGroup(string groupName, [bool  
isFree], [bool hasGreatSpeakers], [bool hasSwag])
```

◆ hasGreatSpeakers:

◆ isFree:

# Improved COM Support

- Dynamic Type
- Optional and named parameters
- Support for COM indexed properties
- ref modifier is now optional
- Type Equivalence and Type Embedding (NoPIA)
- Improvements in COM event handling

# Dynamic Types And COM

- When you import a type library in .NET 4 you can choose whether to import types as object (.NET 1 to 3.5) or dynamic (.NET 4)

```
// C# 1 to 3.5
((Excel.Range)excel.Cells[1, 1]).value2 = "Hello";
Excel.Range range = (Excel.Range)excel.Cells[1, 1];
```

```
// C# 4 (type library imported with dynamic types)
excel.Cells[1, 1].value = "Hello";
Excel.Range range = excel.Cells[1, 1];
```

# Optional and Named Parameters And COM

```
// C# 1 to 3.5  
Excel.Chart chart = (Excel.Chart)  
excel.Application.Charts.Add(Type.Missing, excel.ActiveSheet,  
Type.Missing, Type.Missing);
```

```
// C# 4  
Excel.Chart chart =  
excel.Application.Charts.Add(After: excel.ActiveSheet);
```

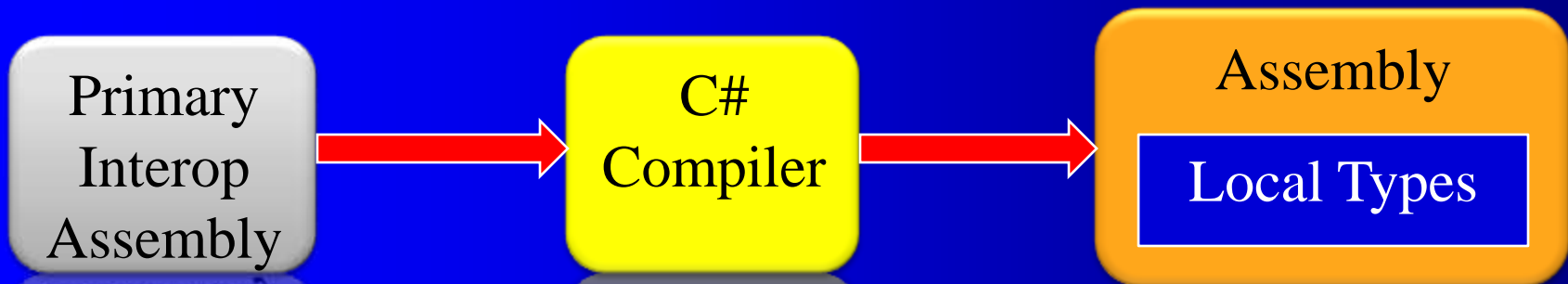
# ref Modifier Is Optional

```
// C# 1 to 3.5  
var missing = Type.Missing;  
word.Documents.Add(ref missing, ref missing, ref missing,  
    ref missing);
```

```
// C# 4  
word.Documents.Add();
```

# Type Embedding (1 of 2)

- Types from COM type libraries can be optionally embedded in the assembly
  - No Primary Interop Assembly (PIA) is required
  - Partial 'local' versions of the types are copied into the assembly



# Type Embedding (2 of 2)

- Only types from interop assemblies can be embedded
  - Assemblies must have the following attributes:-

```
[assembly:Guid()]  
[assembly:ImportedFromTypeLib()]
```

- Only metadata is embedded
  - interfaces, delegates, enums, simple structs
  - Classes cannot be embedded

# Type Equivalence

- Interfaces that have the same GUID are treated as equivalent types by the CLR

# Benefits Of Type Embedding And Type Equivalence (NoPIA)

- Smaller deployment footprint
  - Office 2007 PIA footprint is 6.3Mb
- Develop against one version of Office (e.g. 2003) and run against a different version (e.g. 2007)
- Load multiple versions of type libraries simultaneously

# Co-Variance And Contra-Variance

```
// C# 1, 2, 3, 3.5 and 4
IList<string> strings = new List<string>();
// Cannot implicitly convert type
IList<object> objects = strings;
```

```
// this is illegal because objects[0] is strings[0]
objects[0] = 42;
```

```
IList<string> strings = new List<string>();
// this will work in a future version of C# 4
IEnumerable<object> objects = strings;
```

# Co-Variance (out modifier)

```
public interface IEnumerable<out T>: IEnumerable
{
    IEnumerator<T> GetEnumerator();
}
```

# Contra-Variance (in modifier)

```
public interface IComparer<in T>: IEnumerable
{
    public int Compare(T left, T right);
}
```

# Variance Limitations

- Variance type parameters can only be declared on interfaces and delegates
- Variance only applies when there is a reference conversion
  - `IEnumerable<int>` to `IEnumerable<object>` is a boxing conversion not a reference conversion

# Beyond C# 4

## Compiler As A Service (1 of 3)

```
public class CSharpEvaluator
{
    public CSharpEvaluator();
    public CSharpEvaluator(CSharpCompiler compiler);

    public Collection<assembly> References { get; }
    public Collection<string> Usings { get; }

    public void Clear();
    public EvaluationResult Eval(string program);
    public bool IsComplete(string program);
}
```

# Beyond C# 4

## Compiler As A Service (2 of 3)

```
public class EvaluationResult
{
    public EvaluationResult(
        object value, IList<ICSError> errors);

    public IList<ICSErrors> Errors { get; }
    public object value { get; }

    public override string ToString();
}
```

# Beyond C# 4

## Compiler As A Service (3 of 3)

```
static void Main(string[] args)
{
    CSharpEvaluator evaluator = new CSharpEvaluator();
    evaluator.Usings.Add("System");
    EvaluationResult result = evaluator.Eval("1 + 2");
    Console.WriteLine(result);
}
```

# Uses For "Compiler As A Service"

- Read-Eval-Print Loop (REPL)
- C# scripting (runtime extensibility)
  - Embed C# in Domain Specific Languages
    - e.g. Windows Workflow code snippets
- Meta programming
- Language Object Model
  - To enable C# refactoring

# Summary

- Dynamically Typed Objects
- Optional And Named Parameters
- Improvements To COM Support
- Co- and contra-variance
- Beyond C# 4