Building Metro Style Apps with XAML: What .NET Developers Need to Know

Lieven Iliano
Partner
U2U

Agenda

- WinRT for the .NET Developer
- Async
- Tools

You'll leave understanding:

- New concepts from a XAML developers perspective for building Metro style apps

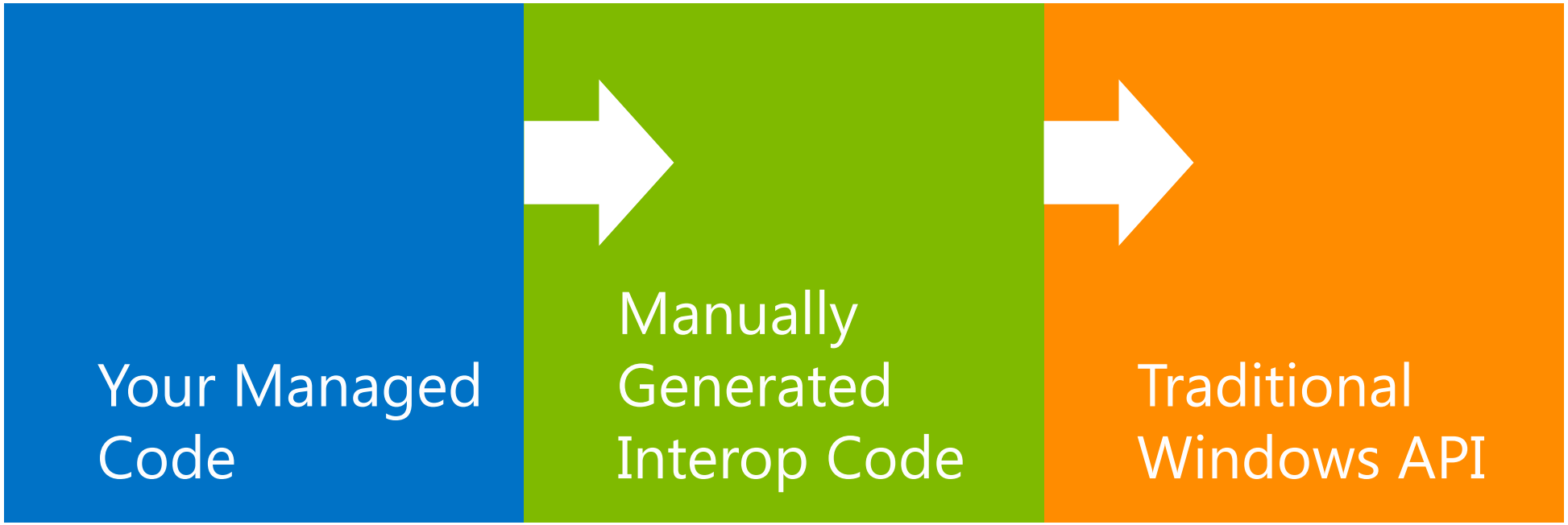
WinRT for the .NET Developer

Windows has always provided compelling capabilities for developers to build upon

Windows has not always made it straightforward for you to use these capabilities from C# or VB

The C# code you have to write today…

```
[DllImport("avicap32.dll", EntryPoint="capCreateCaptureWindow")]
static extern int capCreateCaptureWindow(
   string lpszWindowName, int dwStyle,
   int X, int Y, int nWidth, int nHeight,
   int hwndParent, int nID);

[DllImport("avicap32.dll")]
static extern bool capGetDriverDescription(
   int wDriverIndex,
   [MarshalAs(UnmanagedType.LPTStr)] ref string lpszName,
   int cbName,
   [MarshalAs(UnmanagedType.LPTStr)] ref string lpszVer,
   int cbVer);

// more and more of the same
```

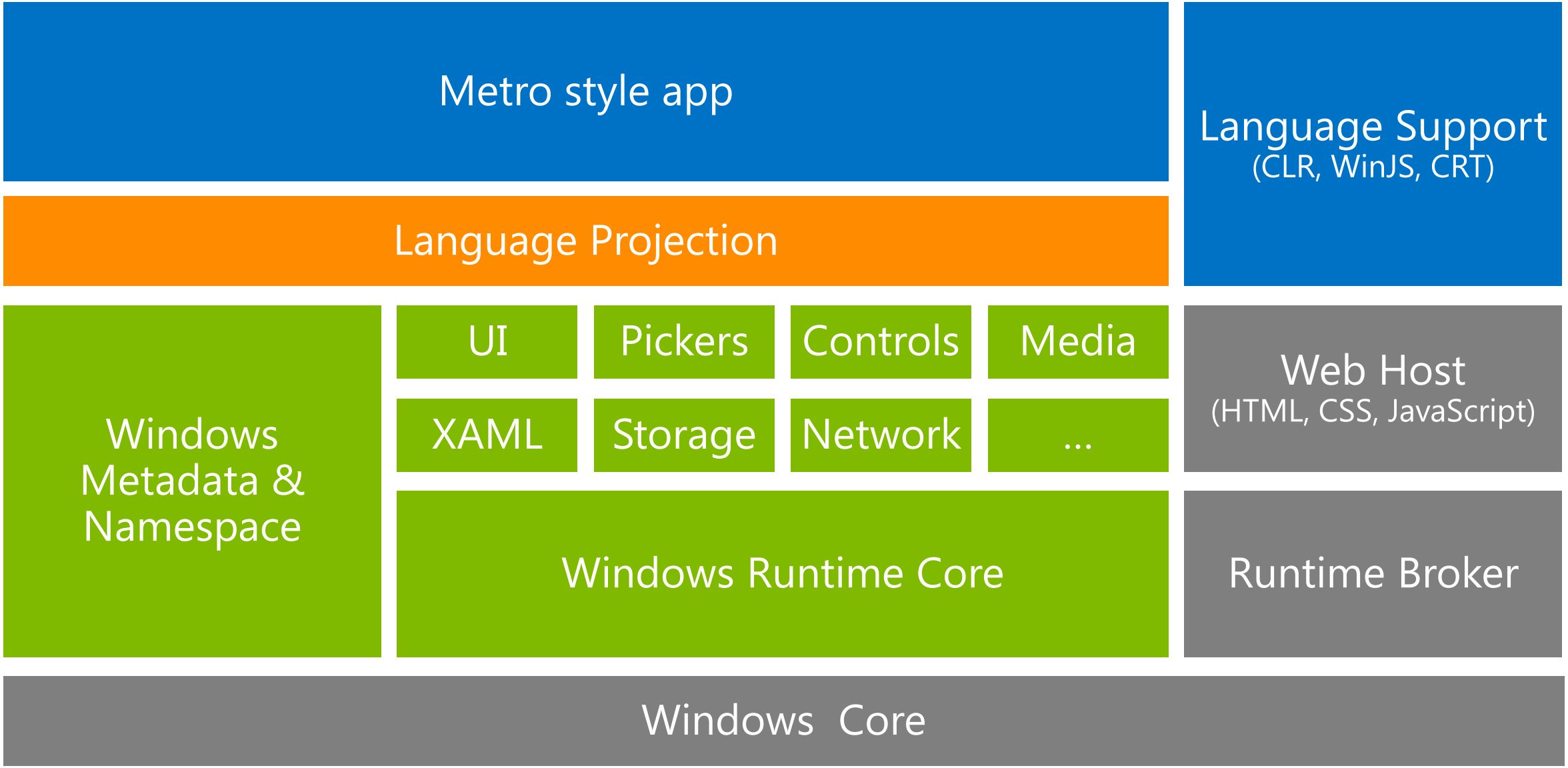

Your Managed Code
Manually Generated Interop Code
Traditional Windows API

The C# code you get to write on Windows 8

```
using Windows.Media.Capture;

var ui = new CameraCaptureUI();
ui.PhotoSettings.CroppedAspectRatio = new Size(4, 3);

var file = await ui.CaptureFileAsync(CameraCaptureUIMode.Photo);

if (file != null)
{
    var bitmap = new BitmapImage() ;
    bitmap.SetSource(await file.OpenAsync(FileAccessMode.Read));
    Photo.Source = bitmap;
}
```


Your Managed Code
Windows Runtime
Windows 8 API

Windows Runtime Architecture



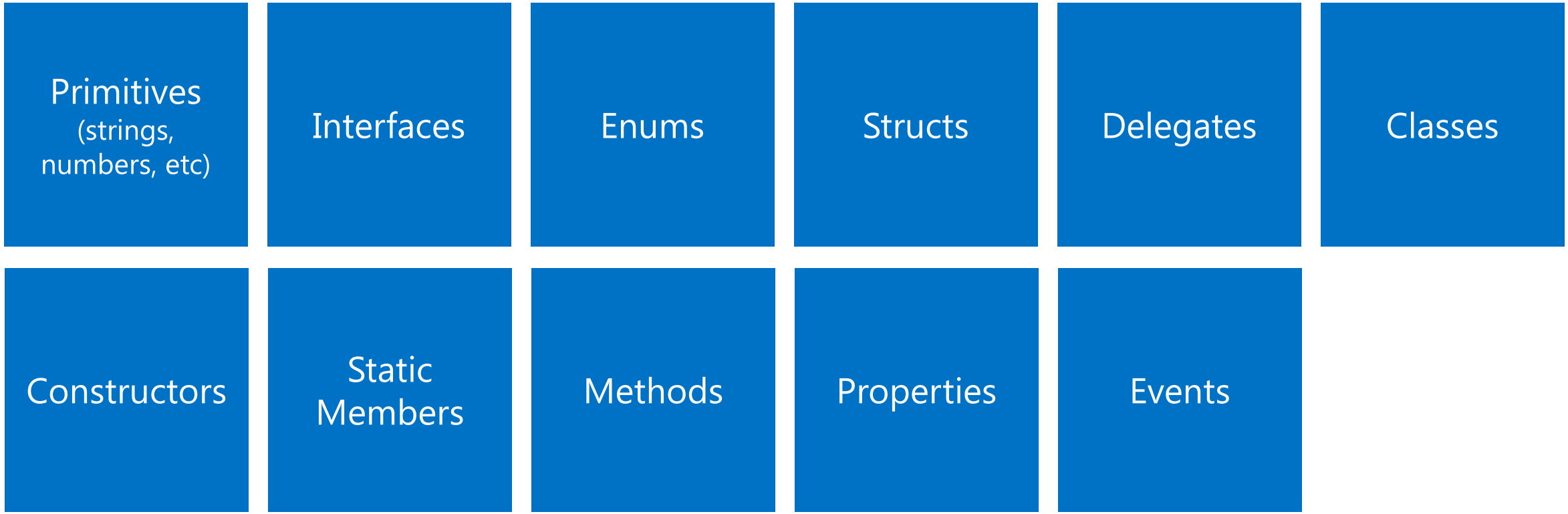
Using the Windows Runtime feels natural and familiar from C# and Visual Basic

You already have the skills to build Metro style apps with C# and VB

demo

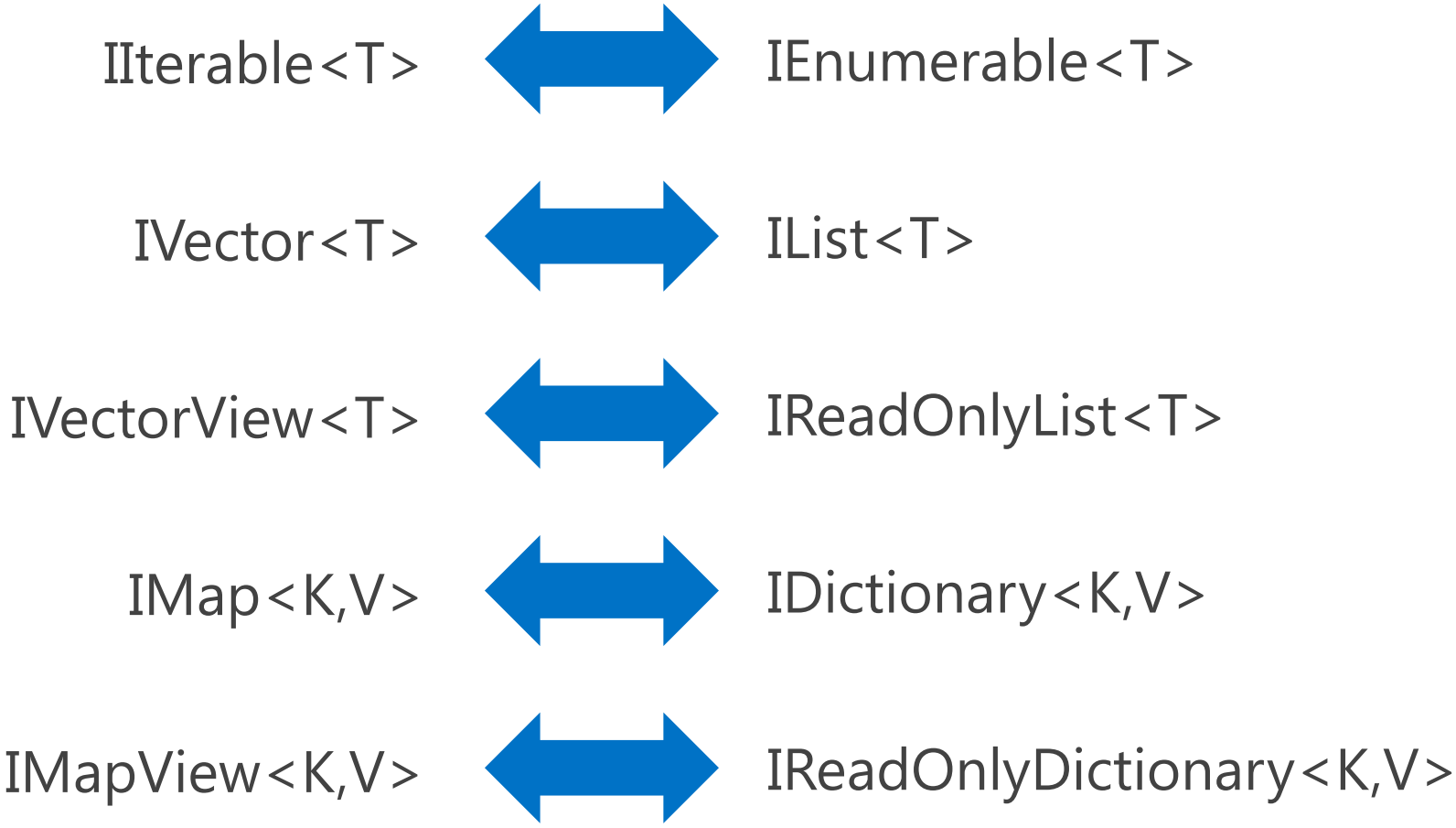
Using the Windows Runtime from C#

Almost everything maps directly between the Windows Runtime and .NET



Most differences between Windows Runtime and .NET are hidden

.NET automatically maps collection interfaces to their Windows Runtime equivalent



Extension methods bridge the gap between Windows Runtime and managed code

Streams Code Sample

```
FileOpenPicker picker = new FileOpenPicker();
picker.FileTypeFilter.Add("*");

StorageFile file = await picker.PickSingleFileAsync();

Windows.Storage.Streams.IInputStream inputStream =
    await file.OpenForReadAsync();

System.IO.Stream stream = inputStream.AsStream();
System.IO.StreamReader reader = new StreamReader(stream);

string contents = reader.ReadToEnd();
```

By adhering to a few simple rules, you can build a managed Windows Runtime component that projects into C++ or JavaScript

Only the public types and members in your managed Windows Runtime component project need to follow the simple rules

API signatures must only use Windows Runtime types

Structs can only have public data fields

Inheritance can only be used for XAML controls,
all other types must be sealed

Only supports system provided generic types

Visual Studio has built-in support for managed Windows Runtime component projects

demo

Building Windows Runtime Components in C#

Async

Asynchronous programming is becoming the norm in modern, connected applications

Synchronous vs. asynchronous

```
var data = DownloadData(...);
ProcessData(data);
```


```
var future = DownloadDataAsync(...);
future.ContinueWith(data => ProcessData(data));
```

Asynchronous programming models

- Windows Runtime: IAsyncOperation<T>
- .NET Framework: Task<T>
- Javascript: Promises
- All are objects representing “ongoing operations”
- All use callbacks to signal completion of operation
- Challenge: Callbacks turn your code inside out
- Insight: Automatic transformation to callbacks is possible

Asynchronous methods automatically transform normal code into a callback state machine

Asynchronous methods

```
public async Task<XElement> GetXmlAsync(string url) {
    var client = new HttpClient();
    var response = await client.GetAsync(url);
    var text = response.Content.ReadAsString();
    return XElement.Parse(text);
}
```

```
public Task<XElement> GetXmlAsync(string url) {
    var tcs = new TaskCompletionSource<XElement>();
    var client = new HttpClient();
    client.GetAsync(url).ContinueWith(task => {
        var response = task.Result;
        var text = response.Content.ReadAsString();
        tcs.SetResult(XElement.Parse(text));
    });
    return tcs.Task;
}
```

Asynchronous methods…

- Are marked with new “async” modifier
- Must return void or Task<T>
- Use “await” operator to cooperatively yield control
- Are resumed when awaited operation completes
- Can await anything that implements the “awaiter pattern”
- Execute in the synchronization context of their caller
- Allow composition using regular programming constructs
- Feel just like good old synchronous code!

C# and Visual Basic let you do asynchronous programming without callbacks

async makes your method asynchronous

await makes the rest of your method a callback

Task lets you coordinate activities

demo

Async in WinRT

Tools

Visual Studio Designer

- Same designer for all supported languages: C#, C++ and VB
- Core authoring UI now shared with Expression Blend
  - Consistent XAML parsing
  - Shared UI where is makes sense
  - Each tool optimized for target user and workflow

demo

Visual Studio Express 11
XAML Designer

Recap

You already have the skills to build Metro style apps with C# and VB

Influenced by C# and VB

Feels natural and familiar from C# and Visual Basic

Build your own managed Windows Runtime components

Async == No more callbacks!

Work visually to be more productive

Edit XAML directly for more control

Templates provide base structure of Metro style apps

Thank you

Feedback and questions	Session feedback
http://forums.dev.windows.com	http://bldw.in/SessionFeedback

Microsoft®