

Programming Windows 8 With WinRT

WinRT

The Start Of A New Era



Rainer Stropek

software architects gmbh

Mail
Web
Twitter

rainer@timecockpit.com
<http://www.timecockpit.com>
[@rstropek](https://twitter.com/rstropek)



time cockpit
Saves the day.

Agenda



Windows reimagined
Metro-Design, Fast And Fluid, Touch, etc.

What's WinRT?

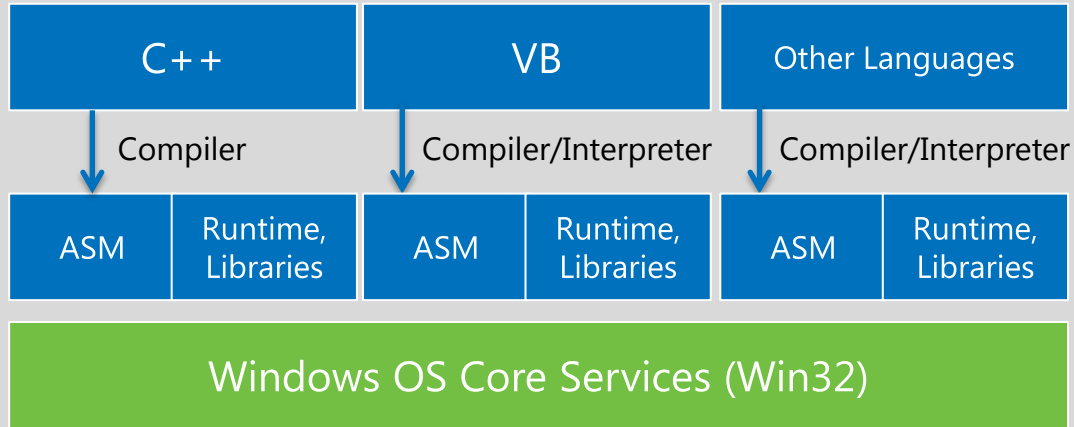
Windows Runtime (WinRT) is the basic technology that enables Windows 8.

What is WinRT from a developer's standpoint?

Is it really revolutionary new?
What has changed? Why was change necessary?

This session will be a technical deep dive into WinRT. It will not be a design or marketing session for Windows 8.

The Early Days...



The World Before .NET...

...consisted of a lot of islands

Before .NET...

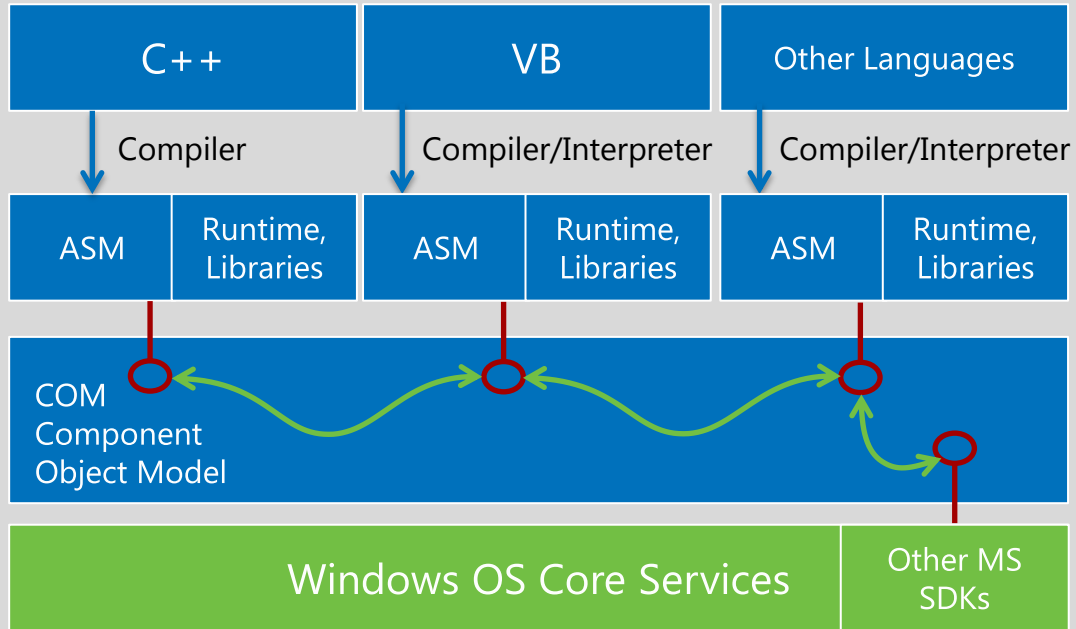
... every language had its own compiler/interpreter, runtime environment, libraries, etc.

This caused a lot of **problems**:

- Expensive to implement.
- How can they work together?
- How can one benefit from innovation happening in another island?

And many, many more.

The Early Days...



COM

Component Object Model

COM...

... introduced a **binary interface for interoperable components**. This opened a whole new world on the Windows platform:

- **Cross-language** component libraries.
- **Rich application integration** became possible – can you remember OLE?
- Even richer **component technologies** appeared – can you remember COM+, DCOM, or VBA?

Problems

What Was Wrong With COM?

- ▶ **Some design flaws**

 - Can you remember the "DLL Hell"?

 - Not really ready for the upcoming internet.

- ▶ **COM was no native part of the programming languages**

 - Inside a component you could use all language features.

 - You had to write separate COM wrapper (e.g. map your types to COM's type system).

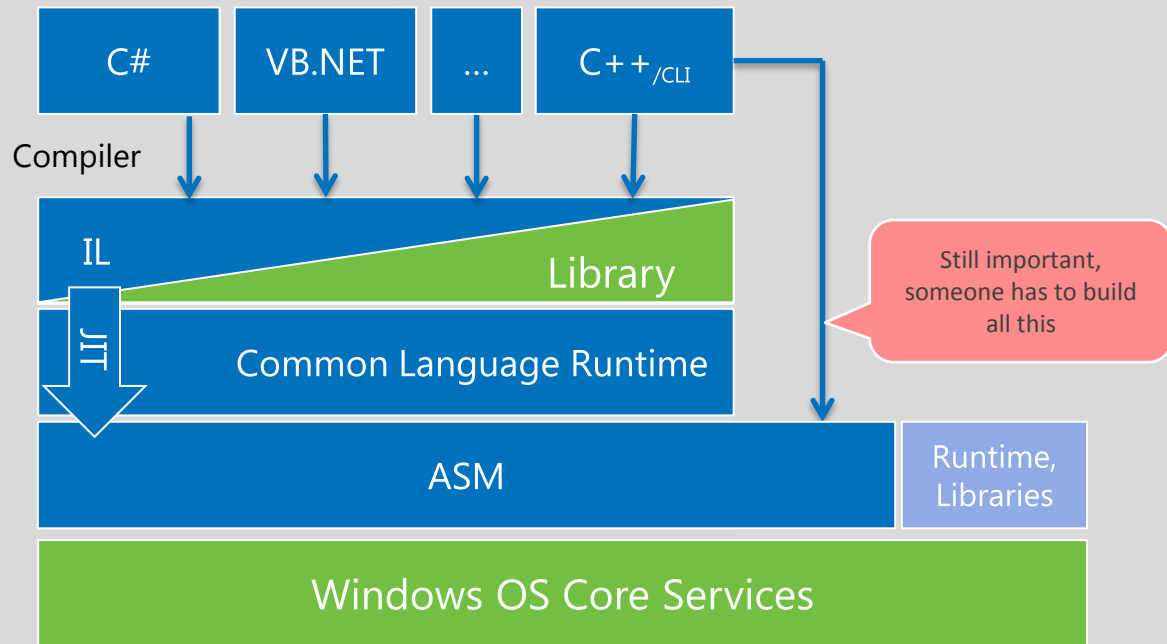
 - Etc.

Problems

And Finally...

- ▶ **There was this new language "Java"**
 - More productive and robust than e.g. C++.
 - Built for a component-oriented world from day one.
 - Platform independent.
 - The next big thing?

.NET Appears...



.NET = CLR + Library

Common Language Runtime + .NET Framework Class Library

The .NET Revolution

Let's introduce a **common** language runtime with a **class library** and a **language** that can compete with Java.

- Building **languages** will be easier.
- Strong cross-language **interoperability**.
- Many languages benefit from the world-class, internet-ready **class library**.
- Make the CLR a standard and become **platform independent**.

.NET – A Big Success!

- ▶ Today dozens of languages are based on CLR
We even have a DLR for dynamic languages like Python, Ruby and JavaScript.
- ▶ In various products (hosts) and on multiple platforms
CLR and C# are standards, somewhat platform independent (e.g. Mono, Silverlight).
- ▶ .NET Framework Class Library
One of the most complete frameworks on the market today.
Big success factor for the .NET platform.

.NET For Everybody!

Well...

- ▶ **Huge C++ Codebase**

PInvoke and COM Interop are ok but definitively no perfect solution.

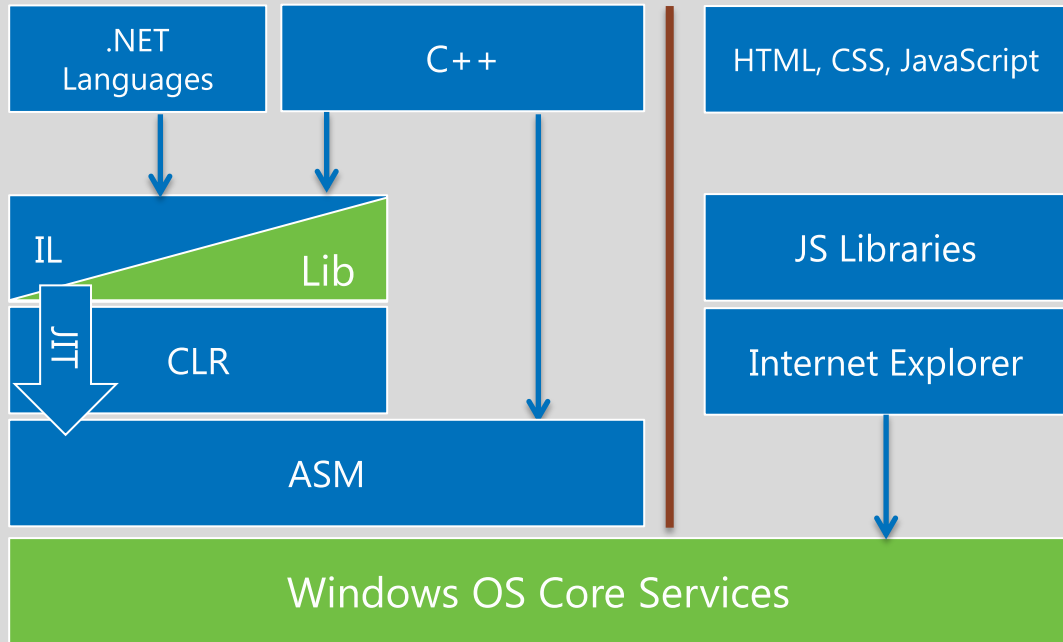
- ▶ **Unmanaged code is important in some cases**

Games, compute-intensive calculations, etc.

- ▶ **...and then there is the browser**

JavaScript rules the web.

10 Year After The Rise Of .NET...



C++ And JavaScript

C++ is still strong and JavaScript is the new big player

Limits Of .NET?

- .NET Library often just a **thin wrapper** around OS services.
- (Unmanaged) C++ is still very important, cannot benefit from .NET Class Library (e.g. XAML).
- JavaScript has become popular; de-facto standard for platform independence.
- .NET Class Library needs some **redesign**.

WinRT is for the new world of touch, tablets, and language interoperability.

Access Native Code From .NET

```
[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);
```

PInvoke

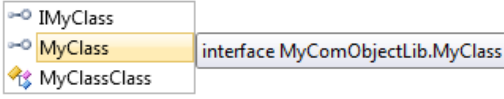
Platform Invoke.

Import functions from
unmanaged libraries in .NET.

See [MSDN](#) for details.

Access Native Code From .NET

```
namespace ConsoleApplication1
{
    class Program
    {
        static void Main(string[] args)
        {
            var x = new XmlDocument();
            var y = new MyComObjectLib.MyClass
        }
    }
}
```



```
//000013: var x = new XmlDocument();
```

```
IL_0001: newobj
        instance void [System.Xml]System.Xml.XmlDocument::.ctor()
```

```
//000014: var y = new MyComObjectLib.MyClass();
```

```
IL_0007: ldstr    "F61ED984-6153-486D-8392-303545E63C2D"
IL_000c: newobj   instance void [mscorlib]System.Guid::.ctor(string)
IL_0011: call     class [mscorlib]System.Type
        [mscorlib]System.Type::GetTypeFromCLSID(valuetype [mscorlib]System.Guid)
IL_0016: call     object [mscorlib]System.Activator::CreateInstance(...)
IL_001b: castclass MyComObjectLib.MyClass
```

COM Interop

Access COM objects from .NET

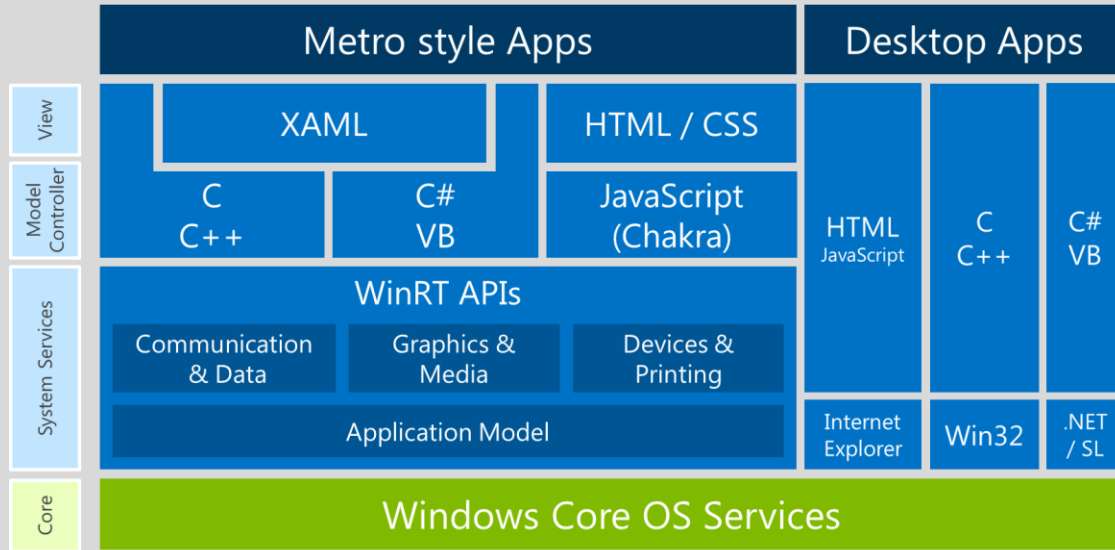
Simplified in latest versions of .NET, still it does not feel "natural" to .NET developers.

See [MSDN](#) for details.

From .NET To WinRT

- ▶ **Combine what's great in COM and .NET**
Binary interface for language interoperability.
Metadata management of .NET.
- ▶ **"New COM" as a natural part of all languages/platforms**
.NET, C++, and JavaScript; maybe others will join the party, too.
- ▶ **Build a new class library on the new runtime**
Cleanup the library compared to .NET (e.g. async rules, dependencies, etc.)
- ▶ **Do not enforce managed code**

The Windows 8 Platform



WinRT is...

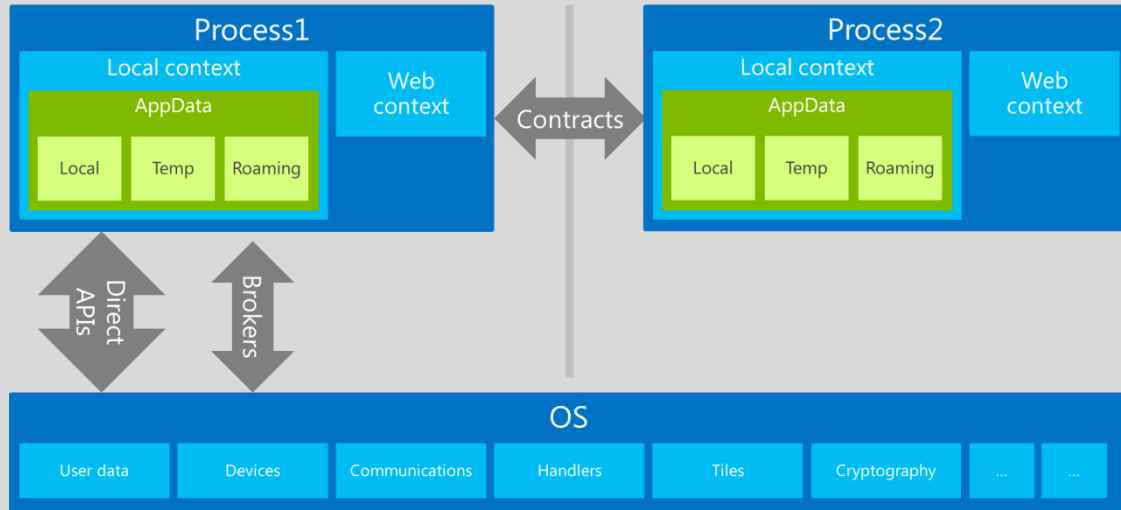
... underlying infrastructure of **metro-style apps** in Windows 8.

- **Binary interface** for interoperability (based on COM).
- **Language projection layer** for .NET languages, C++ and JavaScript.
- **Class library** targeted at development of metro-style apps.

WinRT

Windows Runtime – The Common Runtime Of Windows 8

The Windows 8 Platform



WinRT is...

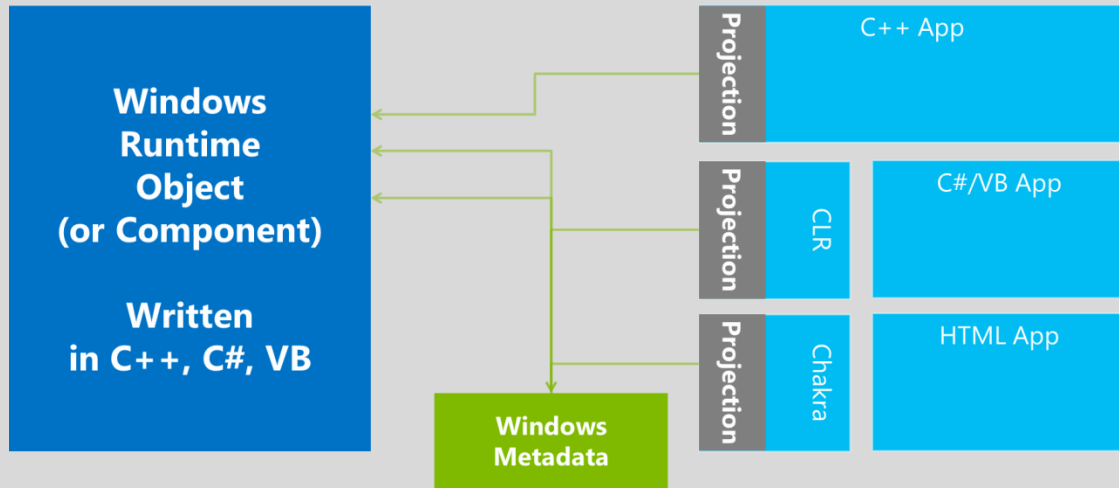
- **secure runtime environment** which is the basis for the upcoming **Windows Store**.

And a lot of other things that we cannot cover here because of the limited time.

WinRT

Windows Runtime – The Common Runtime Of Windows 8

The Windows 8 Platform

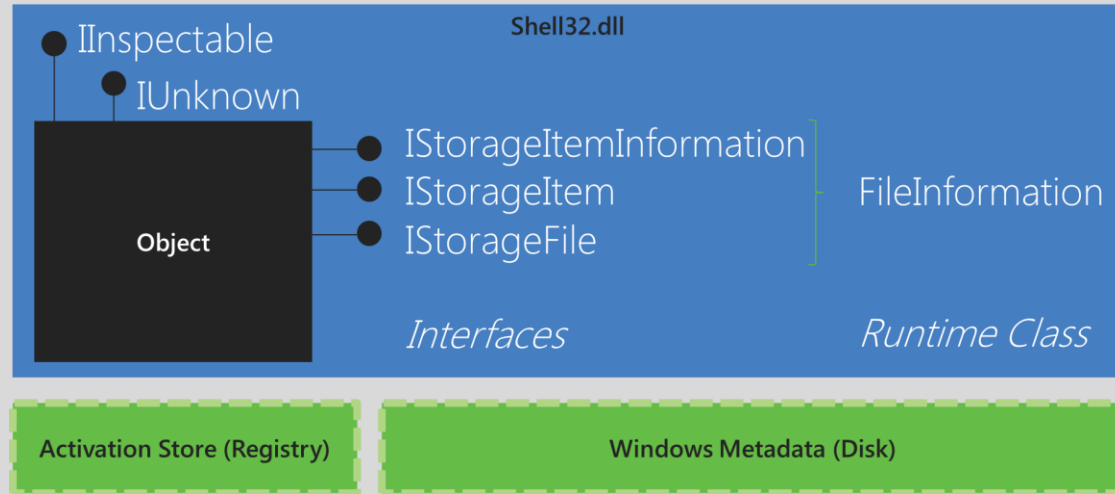


Language Projection Layer
Because WinRT should feel "natural" to you.

Projection Means...

- ... you can stick to **naming conventions** you are used to (e.g. camelCasing in JavaScript, PascalCasing in C#).
- ... you can **write WinRT components** and your compiler will generate the necessary infrastructure (e.g. interfaces)
- ... **ref counting** is automatically done behind the scenes.

The Windows 8 Platform



Projection Means...

... COM's **interface-based logic is translated** into namespaces and classes in the OO world.

Language Projection Layer
Because WinRT should feel "natural" to you.

Language Projection Layer

```
// Note that all parameters are passed as "hat pointers" (^). Hat
// pointers are pointers to WinRT objects. The pointer is
// automatically reference counted (COM's IUnknown.AddRef and
// IUnknown.Release methods).
void OnShowMessageBoxWithCx(
    Object ^sender, Windows::UI::Xaml::RoutedEventArgs ^eventArgs)
{
    // Note that we create the new instance of the WinRT component
    // "MessageDialog" using "ref new" instead of new. This is
    // necessary to retrieve a handle to the instance and let
    // WinRT do the reference counting.
    auto msgBox = ref new Windows::UI::Popups::MessageDialog(
        "Hello World!");
    msgBox->ShowAsync();
}
```

C++/CX

Extends the C++ syntax to make it easier to consume and write WinRT components.

In this sample RoutedEventArgs and MessageDialog are no C++ classes, they are WinRT (=COM) objects.

Quite similar to C#, isn't it?

Language Projection Layer

```
namespace MyApplication
{
    // Note that this class is defined as a ref class.
    // It becomes a WinRT component.
    public ref class BlankPage sealed
    {
        public:
            BlankPage();
        protected:
            virtual void OnNavigatedTo(
                NavigationEventArgs^ e) override;
        private:
            void OnShowMsgBoxWithCx(
                Object ^sender, RoutedEventArgs ^eventArgs)
            { ... }
    }
}
```

C++/CX

Extends the C++ syntax to make it easier to consume and write WinRT components.

In this sample RoutedEventArgs and MessageDialog are no C++ classes, they are WinRT (=COM) objects.

Language Projection Layer

```
Windows::UI::Popups::MessageDialog msgBox(  
    "Hello World!");  
msgBox.ShowAsync();
```

C++, WRL

It is **possible** to consume and write WinRT components in **plain C++**.

You can even do it without any library support (**not recommended**).

If you need to use plain C++ instead of C++/CX use the **Windows Runtime Library (WRL)**.

These two lines of C++/CX code become...

Language Projection Layer

```
void BlankPage::OnShowMsgBoxWrl(
    Object ^sender, Windows::UI::Xaml::RoutedEventArgs ^eventArgs) {
    const wchar_t *msgBoxClassName = L"Windows.UI.Popups.MessageDialog";
    const wchar_t *helloWorldMessage = L"Hello World!";
    HSTRING hString, hWelcomeString;
    auto hr = ::WindowsCreateString(msgBoxClassName,
        static_cast<UINT32>(::wcslen(msgBoxClassName)), &hString);
    if (SUCCEEDED(hr)) {
        Microsoft::WRL::ComPtr<ABI::Windows::UI::Popups::IMessageDialogFactory>
            factory;
        hr = Windows::Foundation::GetActivationFactory(hString, &factory);
        if (SUCCEEDED(hr)) {
            hr = ::WindowsCreateString(helloWorldMessage,
                static_cast<UINT32>(::wcslen(helloWorldMessage)), &hWelcomeString);
            if (SUCCEEDED(hr)) {
                Microsoft::WRL::ComPtr<ABI::Windows::UI::Popups::IMessageDialog> dialog;
                hr = factory->Create(hWelcomeString, &dialog);
                if(SUCCEEDED(hr)) {
                    Microsoft::WRL::ComPtr<
                        __FIAsyncOperation_1_Windows__CUI__CPopups__CIUICommand> command;
                    hr = dialog->ShowAsync(&command);
                }
                ::WindowsDeleteString(hWelcomeString);
            }
        }
    }
    ::WindowsDeleteString(hString);
}
```

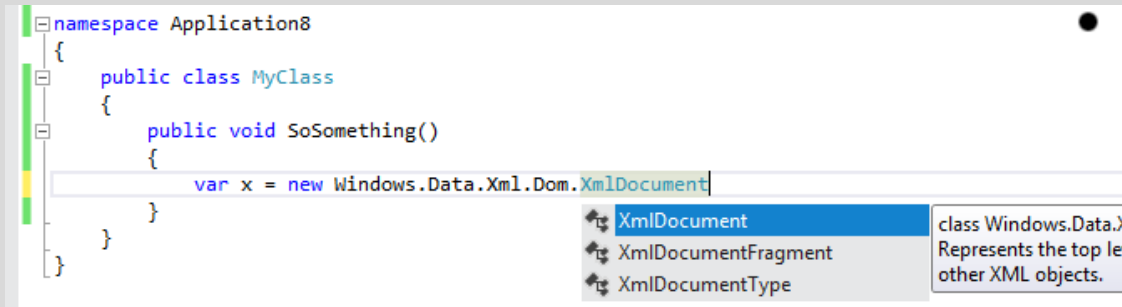
C++, WRL

... many lines of code in plain C++ with WRL.

However, this example shows us that at the very bottom WinRT...

- is based on COM.
- is **exception-free** (exceptions are introduced by the language projection layer).
- has its own **type system**.
- etc.

Language Projection Layer



```
namespace Application8
{
    public class MyClass
    {
        public void SoSomething()
        {
            var x = new Windows.Data.Xml.Dom.XmlDocument();
        }
    }
}
```

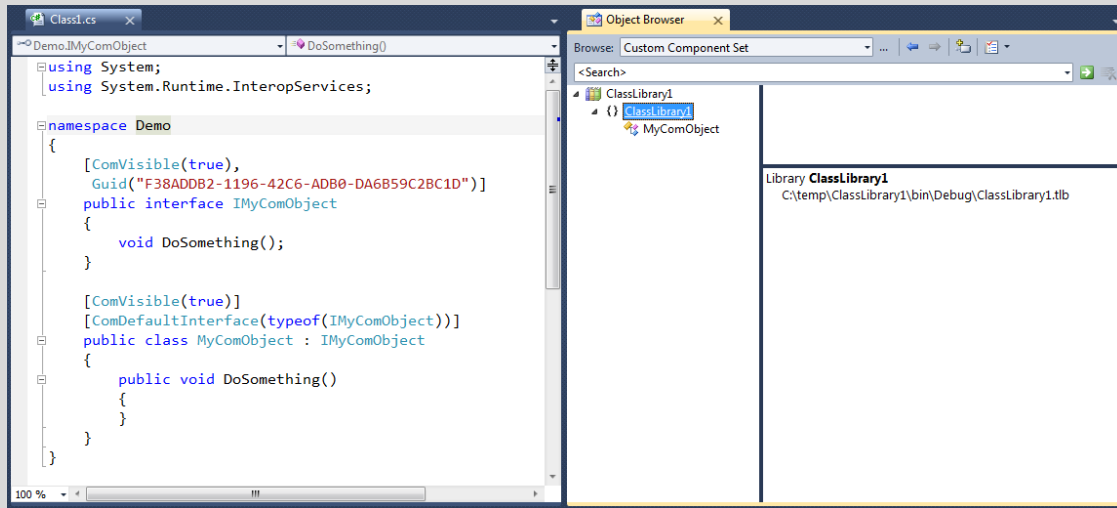
```
.assembly extern System.Runtime {
    .publickeytoken = (B0 3F 5F 7F 11 D5 0A 3A )
    .ver 4:0:0:0
}
.assembly extern windowsruntime Windows {
    .ver 255:255:255:255
}
[...]
//000008: var x = new Windows.Data.Xml.Dom.XmlDocument();
IL_0001: newobj instance void
    [Windows]Windows.Data.Xml.Dom.XmlDocument::.ctor()
```

WinRT And .NET

C# and VB can consume WinRT components as if they were native .NET classes.

Even IL knows about the WinRT. It is deeply integrated into the .NET platform.

Language Projection Layer



WinRT And .NET

While writing a COM component in C#/VS 2010 was complex and a lot of work...

COM Component In C#/VS 2010
A Lot Of Manual Coding...

Language Projection Layer

The screenshot displays the Visual Studio IDE with two main windows. The left window shows the source code for a class named `MyObject` in the `Demo` namespace. The code includes several `using` statements for `System`, `System.Collections.Generic`, `System.Linq`, `System.Text`, and `System.Threading.Tasks`. The class `MyObject` is sealed and contains a constructor `MyObject()` and a public method `DoSomething()`.

The right window shows the Solution Explorer with the project structure. The project is named `ClassLibrary1` and is located in `C:\temp\DummyProjects\ClassLibrary1\bin\Debug\ClassLibrary1.winmd`. The project contains a `MANIFEST` file and a `Demo` namespace. The `Demo` namespace contains the following items:

- `Demo.<CLR>MyObject` (Class)
- `Demo.IMyObjectClass` (Interface)
- `Demo.MyObject` (Class)

The properties window at the bottom shows the following settings:

- Assembly name: `webcamfotoapp`
- Default namespace: `WebCamFotoApp`
- Target framework: (Not set)
- Startup object: (Not set)
- Output type: **Windows Metro style Application** (highlighted with a red box)

WinRT And .NET

... it is easy with the next version of C# and VS.

If you follow some basic rules the compiler will generate the necessary infrastructure for you.

For details see [MSDN](#).

WinRT Component In VS11

Very Simple With Some Limitations You Have To Follow

Language Projection Layer

```
default.js*  Calculator.cs*  
}  MathLibrary.Calculator  
  using System.Collections.Generic;  
  namespace MathLibrary  
  {  
    public sealed class Calculator  
    {  
      public Calculator()  
      {  
        this.MagicNumbers = new List<double>();  
      }  
  
      public IList<double> MagicNumbers { get; private set; }  
      public double DoCalculation() { return 42; }  
      public double Seed { get; set; }  
    }  
  }  
}  
  
WinJS.Application.start();  
  
WinJS.Application.onmainwindowactivated = function (e) {  
  var calc = new MathLibrary.Calculator();  
  calc.magicNumbers.append(15);  
  calc.  
  constructor()  constructor  Result').textContent =  
  doCalculation  
  hasOwnProperty  
  isPrototypeOf  
  magicNumbers  
  propertyIsEnumerable  
  seed  
  toLocaleString  
  toString  
};  
})();
```

Language Interop From C# To JavaScript Via WinRT

Note The Automatic Changes Of Casing...

So What?

If You Are A .NET Developer...

- ▶ **You may ask yourself why you need something new**
Why didn't the other guys simply join your .NET platform?
- ▶ **You will continue to use the CLR.**
It is your bridge to seamless integration with WinRT.
- ▶ **Access OS services directly using WinRT**
No wrappers around Windows API any more.
The parts of the .NET class library you will use will shrink.

So What?

If You Are A C++ Developer...

- ▶ **Welcome to the world of XAML!**
You do not need to be jealous any more because the .NET guys have WPF ;-)
Freely combine e.g. Direct3D with XAML-based UI parts.
- ▶ **You can easily program WinRT with C++/CX**
If you want/need to you can stick to ISO C++ by using WRL.
- ▶ **Language interop with .NET and JavaScript**
Share components with .NET and JavaScript developers.

So What?

If You Are A JavaScript Developer...

- ▶ **Use your existing skillset to write apps for Windows**
You can enter the upcoming Windows Store.
- ▶ **Visit my next session about WinRT and JavaScript**

Programming Windows 8 With WinRT

Q&A

Thank You For Coming.



Rainer Stropek

software architects gmbh

Mail
Web
Twitter

rainer@timecockpit.com
<http://www.timecockpit.com>
@rstropek



time cockpit
Saves the day.