Windows RT Components

CLR and the Windows Runtime

AARHUS
UNIVERSITY
SCHOOL OF ENGINEERING

THE WINDOWS RUNTIME AND .NET

The Sandbox

- Windows RT apps run in some sort of a sandbox.
- The main point of an app sandbox is to protect the user. This is enforced in four different ways:
  - WinRT exposes only safe APIs.
  - The .NET 4.5 subset for Windows Store apps exposes only safe APIs.
  - The code runs in low access, isolated, space, sometimes referred to as the "LowBox".
  - The Windows app store verification tools perform a static analysis and make sure your app isn't accessing anything it shouldn't.

The Driver Model

- In desktop apps, you can access any installed driver you can figure out how to use.
- Windows Store apps use a new driver access model which restricts device access.
- If the device doesn't fit one of the known types with an exposed WinRT API
  - like a webcam or microphone, or network adapter
  - or allowed class driver

  then access is restricted to the Device App identified by the Independent Hardware Vendor (IHV) or Original Equipment Manufacturer (OEM) as part of the driver manifest.

- Access to these devices is done through an API provided by the IHV or OEM which works with the broker to provide access to the device itself, but only to the identified device app.

WinRT vs .Net

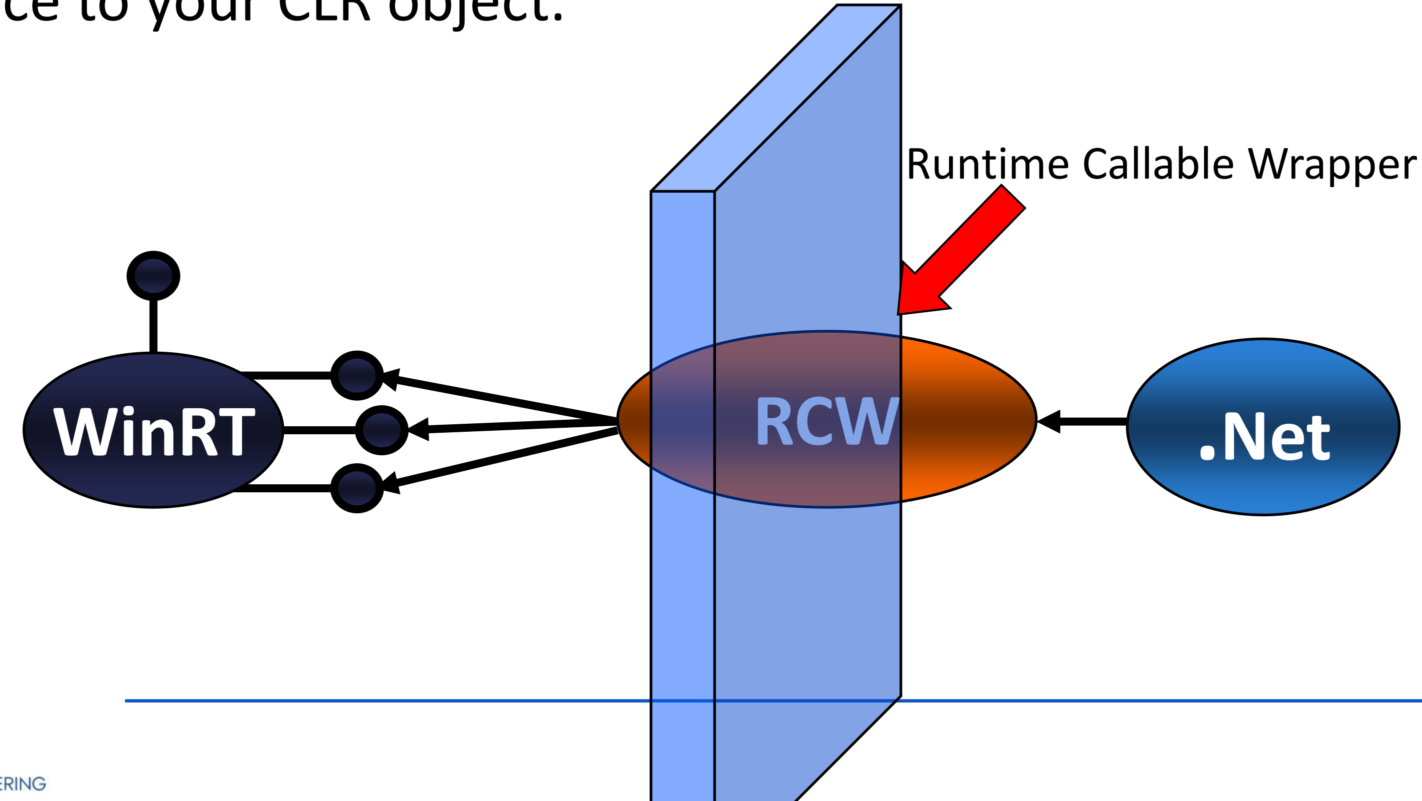
- The number of WinRT components that ship as part of Windows is relatively tiny when compared to the .NET Framework.
- This is by design, because the components are focused on exposing what an operating system does best:
  - abstracting hardware and
  - expose features such as:
    - storage
    - networking
    - graphics
    - media
    - security
    - threading
- Other core language services like string manipulation and LINQ are not offered by the operating system but are provided by the language.

WinRT vs .Net

- WinRT components are internally implemented as Component Object Model (COM) components.
- For WinRT components, Microsoft made a very significant enhancement to COM:
  - instead of using type libraries to describe a COM component's API, they now use metadata.
  - WinRT components describe their API using the same metadata format as .NET (ECMA standard-335).
- This allows languages (like C#) running on top of the CLR to interoperate with WinRT types and components.

WinRT and .Net Interop

- In C#, when you have a reference to a WinRT object, you really have a reference to an RCW that internally refers to the WinRT object.
- Similarly, if you pass a CLR object to a WinRT API, you are really passing a reference to a CCW to the WinRT API and the CCW holds a reference to your CLR object.



Metadata

- WinRT components have their metadata embedded in files with a .winmd file extension.
- The WinRT components that ship with Windows have their metadata in the various Windows.*.winmd files, which can be found in the Windows\System32\WinMetadata directory.
- When building an app, you would reference the one Windows.winmd file that the Windows SDK installs here. ProgramFiles (x86)\Windows Kits\8.0\References\CommonConfiguration\Neutral\Windows.winmd

Projections

- A major design goal of the Windows Runtime type system is to enable developers to be successful in writing apps by using the technologies, tools, practices, and conventions that are already familiar and well-known to them.
- In order to achieve this, some WinRT features are projected to the respective development technologies.
- **CLR projections** are mappings performed implicitly by the CLR, usually related to reinterpreting metadata.
- **Framework projections** are mappings performed explicitly in your code by leveraging new APIs introduced into the Framework Class Library.
- Framework projections are required when the impedance mismatch between the WinRT type system and the CLR's type system is too great for the CLR to do it implicitly.

CLR Projections

- When the CLR sees a WinRT type, it usually allows that type to be used via the CLRs normal COM interop technologies.
- But, in some cases, the CLR hides the WinRT type (by dynamically setting it to private) and then the CLR exposes the type via a different type.

- List of mappings of Windows Runtime types: http://msdn.microsoft.com/en-us/library/windows/apps/hh995050.aspx

WinRT types that map to .NET Framework types with a different name and/or namespace

Windows Runtime type/namespace	.NET Framework type/namespace
DateTime (Windows.Foundation)	DateTimeOffset (System)
EventHandler<T> (Windows.Foundation)	EventHandler<T> (System)
EventRegistrationToken (Windows.Foundation)	EventRegistrationToken (System.Runtime.InteropServices.WindowsRuntime)
HResult (Windows.Foundation)	Exception (System)
IReference<T> (Windows.Foundation)	Nullable<T> (System)
TimeSpan (Windows.Foundation)	TimeSpan (System)
Uri (Windows.Foundation)	Uri (System)
IClosable (Windows.Foundation)	IDisposable (System)
IIterable<T> (Windows.Foundation.Collections)	IEnumerable<T> (System.Collections.Generic)
IVector<T> (Windows.Foundation.Collections)	IList<T> (System.Collections.Generic)
IVectorView<T> (Windows.Foundation.Collections)	IReadOnlyList<T> (System.Collections.Generic)
IMap<K,V> (Windows.Foundation.Collections)	IDictionary<TKey,TValue> (System.Collections.Generic)
IKeyValuePair<K,V> (Windows.Foundation.Collections)	KeyValuePair<TKey,TValue> (System.Collections.Generic)
IBindableIterable (Windows.UI.Xaml.Interop)	IEnumerable (System.Collections)
IBindableVector (Windows.UI.Xaml.Interop)	IList (System.Collections)
INotifyCollectionChanged (Windows.UI.Xaml.Interop)	INotifyCollectionChanged (System.Collections.Specialized)
NotifyCollectionChangedEventHandler (Windows.UI.Xaml.Interop)	NotifyCollectionChangedEventHandler (System.Collections.Specialized)
NotifyCollectionChangedAction (Windows.UI.Xaml.Interop)	NotifyCollectionChangedAction (System.Collections.Specialized)
INotifyPropertyChanged (Windows.UI.Xaml.Data)	INotifyPropertyChanged (System.ComponentModel)
PropertyChangedEventArgs (Windows.UI.Xaml.Data)	PropertyChangedEventArgs (System.ComponentModel)
TypeName (Windows.UI.Xaml.Interop)	Type (System)

Extract!

AARHUS UNIVERSITY
SCHOOL OF ENGINEERING

WinRT types that map to .NET Framework types with the same name and namespace

Namespace	Type
Windows.UI	Color
Windows.Foundation	Point
Windows.Foundation	Rect
Windows.Foundation	Size
Windows.UI.Xaml.Input	ICommand
Windows.UI.Xaml	CornerRadius
Windows.UI.Xaml	Duration
Windows.UI.Xaml	DurationType
Windows.UI.Xaml	GridLength
Windows.UI.Xaml	GridUnitType
Windows.UI.Xaml	Thickness
Windows.UI.Xaml.Controls.Primitives	GeneratorPosition
Windows.UI.Xaml.Media	Matrix
Windows.UI.Xaml.Media.Animation	KeyTime
Windows.UI.Xaml.Media.Animation	RepeatBehavior
Windows.UI.Xaml.Media.Animation	RepeatBehaviorType
Windows.UI.Xaml.Media.Media3D	Matrix3D

WINRT TYPE SYSTEM CORE CONCEPTS

File names and namespaces

- The name of the .winmd file itself must match the name of the namespace containing the WinRT components.
- Because the Windows file system is case insensitive, namespaces that differ by case only are not allowed.
- A WinRT component cannot have the same name as a namespace.

Common base type

- WinRT components do not share a common base class!
- But when the CLR projects a WinRT type, it appears as if the WinRT type is derived from System.Object and therefore all WinRT types inherit public methods like ToString, GetHashCode, Equals, and GetType.
- So, when using a WinRT object via C#, the object will look like it is derived from System.Object, and you can call the "inherited" methods such as ToString.

Core data types

- The WinRT type system supports the core data types such as Boolean, unsigned byte, 16-bit, 32-bit, and 64-bit signed and unsigned integer numbers, single-precision and double-precision floating-point numbers, 16-bit character, strings, and void.
  - Signed byte is not supported by WinRT.
- Like in the CLR, all other data types are composed from these core data types.

Classes

- WinRT is an object-oriented type system, meaning that WinRT components support data abstraction, inheritance, and polymorphism.
- However, some languages (like JavaScript) do not support type inheritance and in order to cater to these languages, almost no WinRT components take advantage of inheritance.
- Only WinRT components consumable from non-JavaScript languages leverage inheritance and polymorphism.
- For the WinRT components that ship with Windows, only the XAML components (for building user interfaces) take advantage of inheritance and polymorphism.
  - Applications written in JavaScript use HTML and CSS to produce their user interface instead.

Structures

- WinRT supports structures (value types), and instances of these are marshaled by value across the COM interoperability boundary.
- Unlike CLR value types, WinRT structures can only have public fields of the core data types or of another WinRT structure.
- WinRT structures cannot have any constructors or helper methods.
- For convenience, the CLR projects some operating system WinRT structures as some native CLR types, which do offer constructors and helper methods.
- These projected types feel more natural to the CLR developer.
- Examples include the Point, Rect, Size, and TimeSpan structures all defined in the Windows.Foundation namespace.

Nullable Structures

- WinRT APIs can expose nullable structures (value types).
- The CLR projects the WinRT's Windows.Foundation.IReference<T> interface as the CLR's System.Nullable<T> type.

DEFINING WINRT COMPONENTS IN C#

Types in Windows Runtime Components

- The fields, parameters, and return values of all the public types and members in your component must be Windows Runtime types.
- Public classes and interfaces can contain:
  - methods
  - properties
  - events
- You can declare delegates for your events, or use the EventHandler<T> delegate.
- All public types must have a root namespace that matches the assembly name.
  - and the assembly name must not begin with "Windows".

Types in Windows Runtime Components

- A public class or interface cannot:
  - Be generic.
  - Implement an interface that is not a Windows Runtime interface.
    - However, you can create your own Windows Runtime interfaces and implement them.
  - Derive from types that are not in the Windows Runtime, such as System.Exception and System.EventArgs.

- Public classes must be sealed.

- Public structures can't have any members other than public fields, and those fields must be value types or strings.

.Net to .Net

- When managed code consumes a WinRT component also written in managed code, the CLR treats the WinRT component as if it were a regular managed component.
  - The CLR will not create CCWs and RCWs,
  - and therefore, it will not invoke the WinRT APIs via these wrappers.
- This improves performance greatly.
- BUT, when testing the component, APIs are not being invoked the way they would be if called from another language (like native C/C++ or JavaScript).