Binding Details

How Binding Occurs

It is possible to use the [Register] attribute, [Export] attribute, and manual Objective-C selector invocation together to manually bind new (previously unbound) Objective-C types.

First, find a type that you wish to bind. For discussion purposes (and simplicity), we'll bind the NSEnumerator type (which has already been bound in MonoTouch.Foundation.NSEnumerator ; the implementation below is just for example purposes).

Second, we need to create the C# type. We'll likely want to place this into a namespace; since Objective-C doesn't support namespaces, we'll need to use the [Register] attribute to change the type name that Xamarin.iOS will register with the Objective-C runtime. The C# type must also inherit from MonoTouch.Foundation.NSObject :

```
namespace MonoTouch.Example.Binding {
    [Register("NSEnumerator")]
    class NSEnumerator : NSObject
    {
        // see steps 3-5
    }
}
```

Third, review the Objective-C documentation and create MonoTouch.ObjCRuntime.Selector instances for each selector you wish to use. Place these within the class body:

```
static Selector selInit       = new Selector("init");
static Selector selAllObjects = new Selector("allObjects");
static Selector selNextObject = new Selector("nextObject");
```

Fourth, your type will need to provide constructors. You *must* chain your constructor invocation to the base class constructor. The [Export] attributes permit Objective-C code to call the constructors with the specified selector name:

```
[Export("init")]
public NSEnumerator()
    : base(NSObjectFlag.Empty)
{
    Handle = Messaging.IntPtr_objc_msgSend(this.Handle, selInit.Handle);
}

// This constructor must be present so that Xamarin.iOS
// can create instances of your type from Objective-C code.
public NSEnumerator(IntPtr handle)
    : base(handle)
{
}
```

Fifth, provide methods for each of the Selectors declared in Step 3. These will use objc_msgSend() to invoke the selector on the native object. Note the use of Runtime.GetNSObject() to convert an IntPtr into an appropriately typed NSObject (sub-)type. If you want the method to be callable from Objective-C code, the member *must* be **virtual**.

```
[Export("nextObject")]
public virtual NSObject NextObject()
{
    return Runtime.GetNSObject(
        Messaging.IntPtr_objc_msgSend(this.Handle, selNextObject.Handle));
```

```
}

// Note that for properties, [Export] goes on the get/set method:
public virtual NSArray AllObjects {
    [Export("allObjects")]
    get {
        return (NSArray) Runtime.GetNSObject(
            Messaging.IntPtr_objc_msgSend(this.Handle, selAllObjects.Handle));
    }
}
```

Putting it all together:

```
using System;
using MonoTouch.Foundation;
using MonoTouch.ObjCRuntime;

namespace MonoTouch.Example.Binding {
    [Register("NSEnumerator")]
    class NSEnumerator : NSObject
    {
        static Selector selInit       = new Selector("init");
        static Selector selAllObjects = new Selector("allObjects");
        static Selector selNextObject = new Selector("nextObject");

        [Export("init")]
        public NSEnumerator()
            : base(NSObjectFlag.Empty)
        {
            Handle = Messaging.IntPtr_objc_msgSend(this.Handle,
                selInit.Handle);
        }

        public NSEnumerator(IntPtr handle)
            : base(handle)
        {
        }

        [Export("nextObject")]
        public virtual NSObject NextObject()
        {
            return Runtime.GetNSObject(
                Messaging.IntPtr_objc_msgSend(this.Handle,
                    selNextObject.Handle));
        }

        // Note that for properties, [Export] goes on the get/set method:
        public virtual NSArray AllObjects {
            [Export("allObjects")]
            get {
                return (NSArray) Runtime.GetNSObject(
                    Messaging.IntPtr_objc_msgSend(this.Handle,
                        selAllObjects.Handle));
            }
        }
    }
}
```

Command Line Bindings

You can use the btouch for Xamarin.iOS (or bmac if you are using Xamarin.Mac) to build your own bindings directly. This is the tool that Xamarin Studio uses to create your bindings.

The general syntax for invoking these tools is:

```
# Use this for Xamarin.iOS:
bash$ /Developer/MonoTouch/usr/bin/btouch -e cocos2d.cs -s:enums.cs -x:extensions.cs

# Use this for MonoMac:
bash$ bmac -e cocos2d.cs -s:enums.cs -x:extensions.cs
```

The above command will generate the file cocos2d.dll in the current directory, and it will contain the fully bound library that you can use in your project.

Source URL: http://docs.xamarin.com/guides/ios/advanced_topics/binding_objective-c_libraries/binding_details