Native .NET Apps for the Mac App Store

Michael Hutchinson

mjhutchinson.com
twitter.com/mjhutchinson
m.j.hutchinson@gmail.com

What is Mono?

.NET everywhere

Compatible and current

Vibrant ecosystem

Free and open-source

- CLI virtual machine, class libraries and ecosystem
- Compatible with .NET 4.0 – no need to recompile
- Use .NET languages, libraries and tools
- Works on many platforms
- Free and open-source

What is MonoMac?

Mac Apps with Mono

Native Mac APIs

Mac App Store

- Provides access to native Mac APIs from Mono
- Build native Mac apps with .NET
- Bridge to Objective-C
- Full access to Cocoa and other Mac APIs from C# and other .NET languages
- Build apps for the Mac App Store

Why Use MonoMac?

Truly Native Experience

.NET languages, libraries, tools

Use existing code and experience

- C#, F#, VB.NET, IronPython, etc.
- Garbage collector, safe managed runtime
- LINQ, Web Services, etc.

Cross-Platform Apps

Native experience
=
native UI toolkit

Shared code, business logic,
libraries, models

Toolkits Everywhere

MonoMac	Mono for Android
WPF/Winforms	GTK#
MonoTouch	Silverlight

- MonoMac = Mac
- MfA = Android
- Winforms = Windows,
- GTK# = Linux
- MonoTouch = iOS
- Silverlight = WP7/Web

MonoDevelop

Mono and .NET

Excellent C# support

Compatible with VS

Open-source

Very extensible

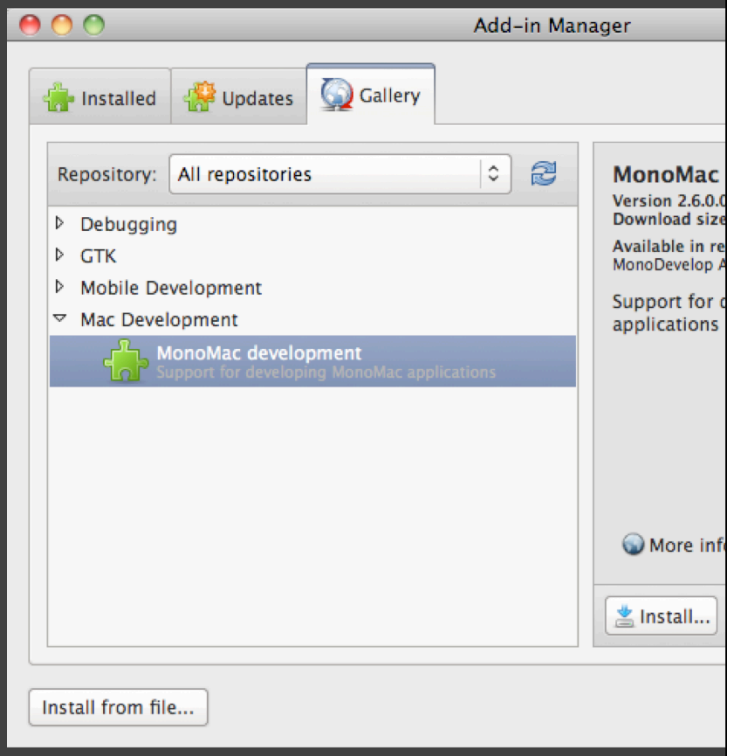
- IDE for Mono and .NET
- Runs on Mac, Linux and Windows
- Excellent C# support
- Compatible with Visual Studio
- Open-source and extensible

Getting MonoMac

1.Mono

2.MonoDevelop

3.MonoMac addin



On a clean, fresh Mac

1. Install Mono and MonoDevelop
2. Open the MonoDevelop Addin Manager
3. Install the MonoMac Addin
4. Create, build and run MonoMac project

Demonstration

Creating a simple MonoMac app

Mac App Bundles

Structured directories

MonoMac projects in MD create and debug apps

Cocoa bundle resources

- Mac apps are bundles, structured directories
  - Info.plist manifest file
  - Resources
- MonoMac projects in MD create and debug App Bundles
- Many Cocoa APIs deal with bundle resources
- Files in a MonoMac project with Content build action are copied into app bundle
- Localizable via lproj bundles

Objective-C

C with Smalltalk -style messaging

Dynamic resolution

Selectors, instances, protocols

Delegate and action/target patterns

- Objective-C is C with Smalltalk-style messaging
  - Separate interface (.h) and implementation (.m)
- Send message to instance or class with objc_MsgSend
  - selector (method name) resolved at runtime
- Classes can have instance fields, like a C struct
- Subclasses can override, handle messages before superclass (base)
- Messages are dynamic
  - Can query whether an object recognizes a selector
  - Can handle unknown selectors
- Protocols are like interfaces
  - But with some methods optional
- Common pattern is for an object to have a “delegate”, an instance conforming to some protocol.
  - Delegate is used like a controller or listener class
  - No relation to .NET delegates

How does it work?

Objective-C bridge

Foundation classes

Binding generator

API wrappers

- MonoMac runtime bridges Objective-C and .NET
  - Selectors, models, foundation classes, etc.
  - Full support for creating and extending/subclassing Obj-C classes from .NET
- MonoMac has wrappers for Cocoa and other Obj-C libraries
  - Can also create new wrappers with the bmac tool
- Bindings for pure C APIs use P/Invoke

Implementing Obj-C Classes

```
[Register("ObjCName")]
public class SomeClass : NSObject
{
    public SomeClass () : base() {}

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

    [Export("initWithCoder:")]
    public SomeClass (NSCoder coder) : base (coder) {}

    [Export("someMethodWithArg1:andArg2:")]
    public void SomeMethod (NSString arg1, int arg2) {}

    public override void EncodeTo (NSCoder coder) {}
}
```

Binding Goals

Easy to use, powerful

.NET conventions and patterns

Consistent and predictable

Everything without BCL equivalent

Stronger typing

- Make the Cocoa and Obj-C APIs easy to use from C#
  - Without preventing advanced uses
- Map Obj-C names to .NET conventions in a consistent and predictable way
  - Cocoa knowledge maps to/from MonoMac
- Bind only APIs that don't have portable .NET equivalents
  - Unless perf or APIs dictate otherwise
- Make the API more strongly typed
- Transparently convert types where possible
  - E.g. NSString in Obj-C becomes System.String in .NET
- Convert constants to enums
  - Enables IDE to provide better code completion
- Map Obj-C "delegates" to .NET events
  - Strongly and weakly typed delegates for advanced use
- Expose .NET delegates to Obj-C as "blocks"

Object Lifetime

Managed NSObject has native backing

Reference-counted

IDisposable is deterministic

Native objects don't retain managed objects

Wrappers have identity

- Every managed class instance derived from NSObject is backed by corresponding native ObjC instance
- Objective-C NSObject is reference-counted
  - `retain` increments, `release` decrements
  - `dealloc` when count reaches zero
- Every managed instance has reference to native instance
  - Releases it when garbage-collected
  - Be careful to hold references to things you need to keep
- Base NSObject wrapper is `IDisposable`
  - Can explicitly release reference with `Dispose()` or C# `using` statement
- Each native instance pointer is surfaced as a single wrapper class instance
  - Runtime.GetNSObject unwraps IntPtr handle

Foundation

MonoMac.Foundation

NSString, NSArray, NSDictionary, NSUrl, NSNumber, etc.

Transparent mapping

Prefer BCL types

- MonoMac.Foundation contains core types from Obj-C
  - NSString, NSArray, NSDictionary, NSUrl, NSNumber, etc.
- Most APIs transparently map these to C# types
- Use .NET BCL types unless good reason to do otherwise
  - But sometimes need direct access for advanced uses
- MonoMac.CoreFoundation contains CFString, etc.
  - Used by some C-based Mac APIs
  - Toll-free bridged to NS* equivalents, Handles are interchangable

MonoMac APIs

AppKit	Security
CoreAnimation	CoreLocation
CoreGraphics	QTKit
CoreText	OpenGL
WebKit	OpenTK
PdfKit	CoreImage
AddressBook	Growl

MonoMac.AppKit
    UI toolkit
MonoMac.CoreGraphics
    Drawing
MonoMac.CoreImage
    Image processing
MonoMac.CoreAnimation
    Animation and compositing
MonoMac.CoreText
    Text rendering
MonoMac.OpenGL, OpenTK
    3D rendering

MonoMac.QTKit
    Quicktime Media
MonoMac.PdfKit
    PDF viewing, annotations, etc
MonoMac.Security
    Mac security framework
MonoMac.WebKit
    Web browser, HTML, JS

AddressBook, AudioToolbox, AudioUnit, CoreData, CoreLocation, CoreMedia, CoreVideo, CoreWlan, Growl, ImageIO , ImageKit, QuartzComposer…

Cocoa AppKit

NSApplication, NSWindow, NSDocument

NSView, NSControl

NSTextField, NSButton

NSTableView, NSTableViewDataSource, NSTableViewDelegate, NSTableViewSource

- NSApplication
  - Single instance, NSApplication.SharedInstance
  - Application-wide menu bar and dock icon
- NSDocument
  - Window and behaviors for document-centric apps
- NSControl
  - Base class for controls
  - Absolute layout, y direction is not what you expect
- NSTableView
  - NSTableViewDataSource (Model)
  - NSTableViewDelegate (Controller)
  - NSTableViewSource (Model & Controller)
- NSMenu
  - Global app-wide menu
  - Context menus

Interface Builder

Integrated into Xcode

xib/nib files

Deserialized using NSCoder

Outlets and Actions

- Apple's GUI designer for Cocoa AppKit
  - Edits xib/nib files
  - Now integrated into Xcode
- Objects deserialized from the nib file at runtime
  - Deserialized using NSCoder
  - Connected using outlets and actions
- Can put locale-specific version of nibs in lproj

Distribution

Mac App Store apps must be self-contained

Create Mac Installer command in MD

Signing, packaging, linker

Just enough Mono

- App Bundles created by MonoDevelop depend on a system installation of Mono
- Mac App Store apps can have no external dependencies
- Use *Create Mac Installer* command in MD
    - Include Mono in the app bundle
    - Use Mono Linker to include “just enough Mono”
    - Create a Mac installer package for the app
    - Sign the app and the installer
    - Any combination of the above

Mac App Store

Mac Developer Program - $99/year

Follow Apple's instructions

Create Mac Installer

Upload to the App Store

???

Profit!

- Join Apple's Mac Developer Program, $99/year
- Follow Apple's instructions
  - Register the App ID
  - Create signing keys for app and installer
- Use *Create Mac Installer* in MonoDevelop
  - Signed app, signed package, no external dependencies
- Upload to the App Store
- ???
- Profit!

Open Community

Core open-sourced from MonoTouch

Bindings, tutorials, samples by the MonoMac community

`mono-osx@lists.ximian.com`

`#monomac on irc.gimp.net`

Resources

Getting started, tutorials, docs

`http://mono-project.com/MonoMac`

Samples in the git repository

`https://github.com/mono/monomac`

Cocoa books and Apple Cocoa docs

- **http://mono-project.com/MonoMac**

- Samples in the git repository
  - https://github.com/mono/monomac
- Cocoa books and Apple Cocoa docs apply to MonoMac too
  - If you can read a little Obj-C

Questions?

C# vs. Objective-C

C#	Objective-C
`instance.InstanceMethod();`	`[instance instanceMethod];`
`instance.Method(arg);`	`[instance methodWithArg:arg];`
`instance.Method(arg1, arg2);`	`[instance methodWithArg1:arg1 andArg2:arg2];`
`this.Method();`	`[self method];`
`base.Method();`	`[super method];`
`SomeClass.StaticMethod();`	`[SomeClass classMethod];`
`foo = new SomeClass();`	`foo = [[SomeClass alloc] init];`
`foo = new SomeClass(arg);`	`foo = [[SomeClass alloc] initWithArg:arg];`
`null`	`nil`
`void Foo()`	`- foo;`
`static void Foo();`	`+ foo;`
`RetType Foo(A arg1, B arg2);`	`- (RetType) fooWithArg:(a)arg1 andArg2:(b)arg2;`

Advanced Bridging

MonoMac.ObjcRuntime is where the magic is:

```
var cls = new Class ("SomeObjcClass");
var sel = new Selector ("someMethodWithCount:andText:");
var str = new MonoMac.Foundation.NSString ("Hi");
IntPtr handle = Messaging.IntPtr_objc_msgSend_int_IntPtr (
    cls.Handle, sel.Handle, 5, str.Handle);
NSObject obj = Runtime.GetNSObject (handle);
```

is equivalent to:

```
NSObject *obj = [SomeObjcClass someMethodWithCount: 5 andText: @"Hi"]
```