MonoMac

Using all of MacOS from C#

The Origins

- UI Models
  - iPhone, iPad and Mac

What is MonoMac?

- .NET Binding to MacOS APIs:
  - Plain C Libraries and Frameworks
  - CoreFoundation based frameworks
    - GObject-like Object Oriented system
    - Many MacOS APIs use this
  - Objective-C based frameworks

MonoMac Benefits on OSX

- One binding, multiple languages
  - C#, F#, IronRuby, IronPython, Phalanger, VB, UnityScript, Boo
- Reuse C# code across platforms
- Strongly typed APIs
  - Easy to explore the API
- Garbage Collection Everywhere
  - Both Foundation and CoreFoundation objects
  - Unlike ARC which is limited to NSObject
- Safe Runtime, access to .NET class libraries, and .NET ecosystem

MonoMac

API Bindings	MonoDevelop Add-In	SDK
• Bindings to the native APIs • Mono Runtime Extensions to bridge .NET and Objective-C • Lion and Snow Leopard	• IDE Project Support • Xcode designer integration • Project Templates	• Build Tools • Build Targets, MSBuild tasks • Mac AppStore Packaging/ Signing • Documentation

Why use MonoMac?

- If you are creating native user experiences
- If you must consume Mac APIs
- If you are focused on the Mac platform
- To publish to the Mac AppStore
  - MonoMac links/bundles the bits you need

Strongly Typed APIs.

- Foundation/CoreFoundation are weakly typed
  - Similar to Glib's GList, GHashTable
  - Or C#'s 1.0 collections: ArrayList, Hashtable

- MonoMac exposes strong types:

```
UIView [] Views { get; set; }
```

vs:

```
(NSArray *) views;
```

Objective-C Sample


```
CIContext *context =
     [CIContext contextWithOptions:
        [NSDictionary dictionaryWithObject:[NSNumber numberWithBool:YES]
                    forKey:kCIContextUseSoftwareRenderer]];
CIImage *ciImage = [CIImage initWithCGImage:cgImage];

CIFilter *hueAdjustFilter = [CIFilter filterWithName:@"CIHueAdjust"];
CIFilter *colorControlsFilter = [CIFilter filterWithName:@"CIColorControls"];

[hueAdjustFilter setValue:[NSNumber numberWithDouble:3.0 * M_PI] forKey:@"inputAngle"];

[colorControlsFilter setDefaults];
[colorControlsFilter setValue:[NSNumber numberWithDouble:1.3] forKey:@"inputSaturation"];
[colorControlsFilter setValue:[NSNumber numberWithDouble:0.3] forKey:@"inputBrightness"];

[hueAdjustFilter setValue:ciImage forKey:@"inputImage"];
[colorControlsFilter setValue:[hueAdjustFilter valueForKey:@"outputImage"] forKey:@"inputImage"];
ciImage = [colorControlsFilter valueForKey:@"outputImage"];

[context [createCGImage: ciImage fromExtent:[ciImage extent]]];
```

C# Version


```
var context = CIContext.FromOptions (new CIContextOptions ()
    UseSoftwareRenderer = true
});
var ciImage = new CIImage (cgImage);
var hueAdjustFilter = new CIHueAdjust {
    InputAngle = 3.0f * Math.PI,
    Image = ciImage,
};

var colorControlsFilter = new CIColorControls {
    InputSaturation = 1.3f,
    InputBrightness = 0.3f,
    Image = hueAdjustFilter.OutputImage
};

ciImage = colorControlsFilter.OutputImage;
context.CreateImage (ciImage, ciImage.Extent);
```

C APIs

- CoreFoundation APIs

```
CFStringRef keys[] = {
    kCTFontAttributeName,
    kCTForegroundColorAttributeName
};

CFTypeRef bval[] = {
    cfListLineCTFontRef,
    CGColorGetConstantColor(kCGColorBlack)
};

attr = CFDictionaryCreate (kCFAllocatorDefault,
    (const void **) &keys, (const void **) &bval,
    sizeof(keys) / sizeof(keys[0]), &kCFTypeDictionaryKeyCallBacks,
    &kCFTypeDictionaryValueCallBacks);

astr = CFAttributedStringCreate(kCFAllocatorDefault, CFSTR("Hello World"), attr);
```

- C# Projection:

```
var attrs = new CFStringAttributes {
    Font = listLineCTFont,
    ForegroundColor = UIColor.Black.CGColor
};

var astr = new NSAttributedString ("Hello World", attrs);
```

- AudioToolbox:

```
UInt32 maxPacketSize;
UInt32 Propertysize = sizeof(maxPacketSize);
AudioFileGetProperty (
    audioFileID,
    kAudioFilePropertyPacketSizeUpperBound,
    &Propertysize,
    &maxPacketSize
    );
```

- C# Projection:

```
var maxPacketSize = audioFile.PacketSizeUpperBound;
```

Code Completion, online help

- Explore the API with Code Completion
- Online help contents rendered with API use
- Enums vs integer constants:

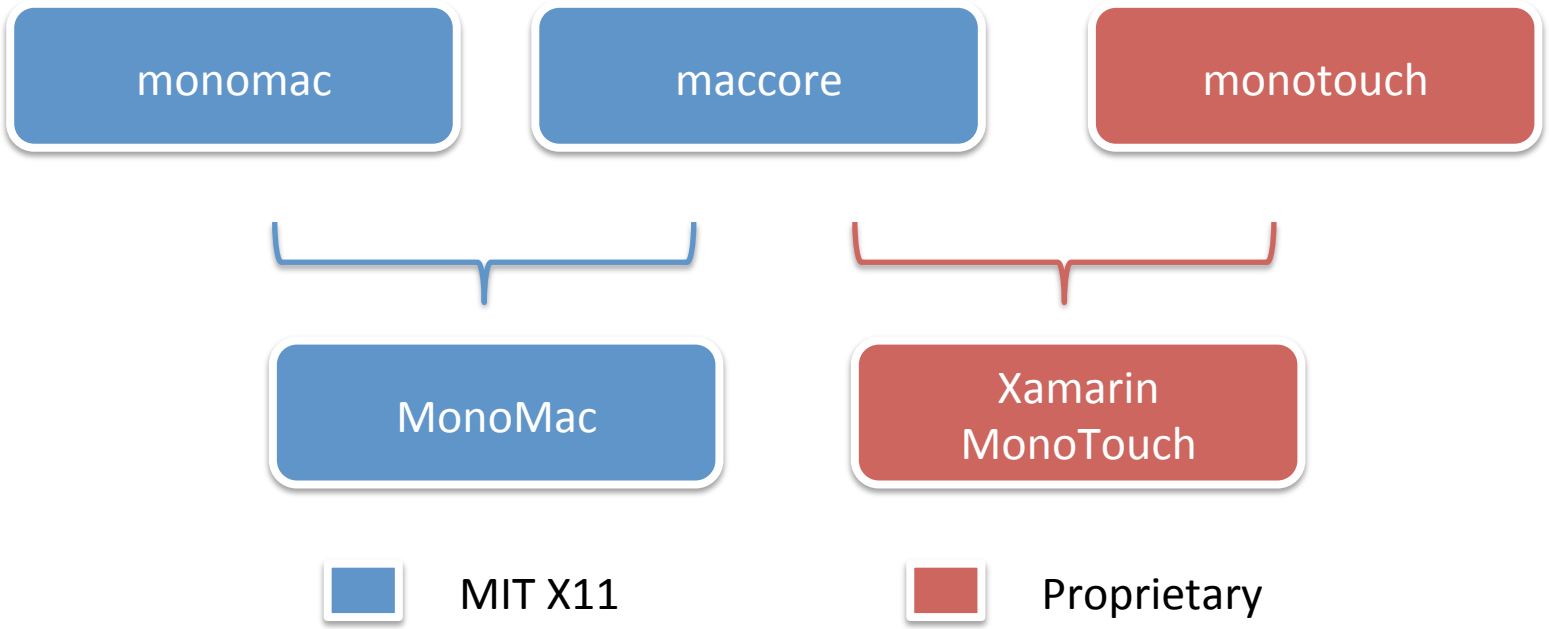
  UIView.Style

  vs

  UIVIEW_STYLE

MonoMac Licensing

- MonoMac is MIT X11 Licensed
- Shares its DNA with MonoTouch:



DEMO

What is in the Package

monomac

- AppKit + CoreAnimation
  - Includes Lion APIs
- CoreWLan
- ImageKit
- OpenTK 1.0
  - OpenGL and OpenAL bindings
- PdfKit
- QTKit + AVFoundation
- QuartzComposer
- WebKit

maccore

- AddressBook
- AudioToolbox
- AudioUnit
- CoreData
- CoreFoundation
- CoreGraphics
- CoreImage
- CoreLocation
- CoreMedia
- CoreText
- CoreVideo
- Security

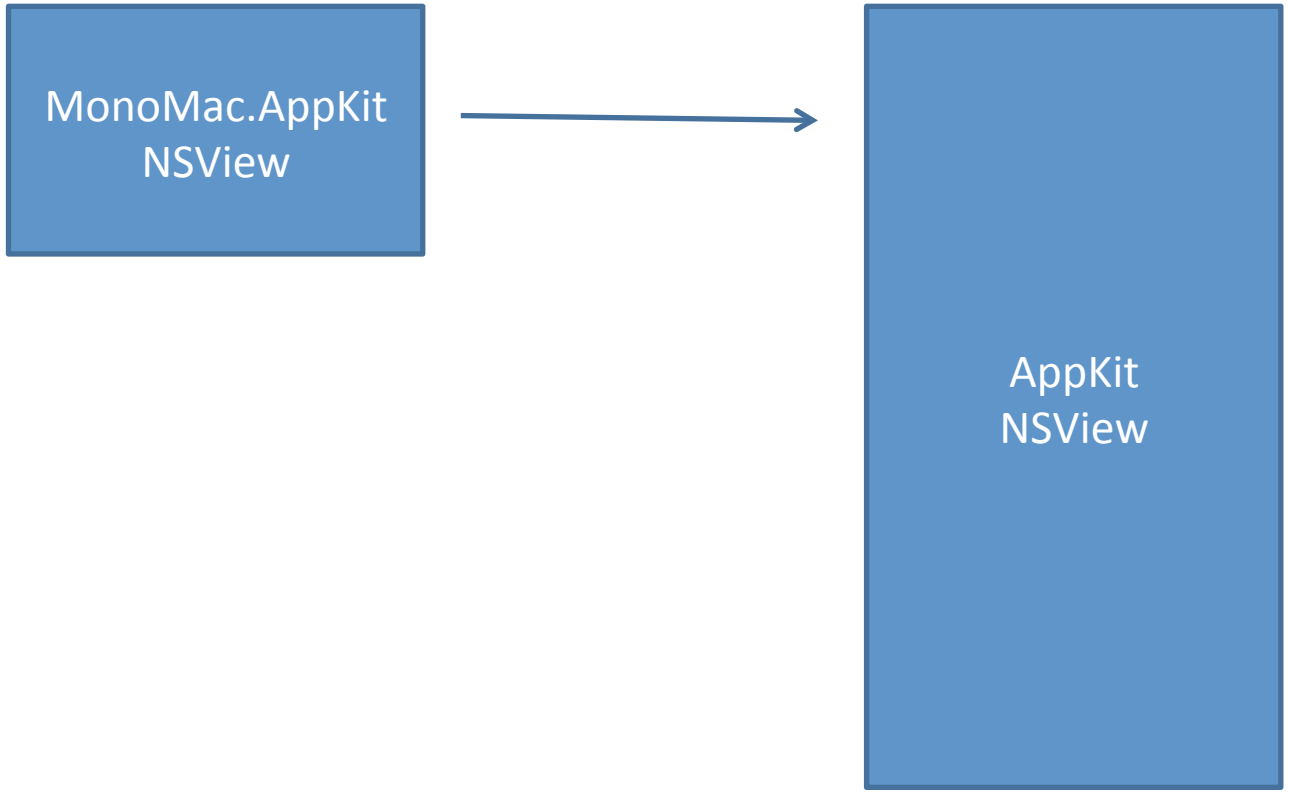
Design Philosophy

- Expose a direct binding to native APIs
  - Most of the time 1:1 mapping
- Expose strong types, hide weak types
- Wrap “Dictionary”-based APIs into strong types

CoreFoundation Style Bindings

- Manually bound
- Some conventions:
  - IDisposable, INativeObject
  - public Constructor (IntPtr handle, bool owns)

- Requires mapping C design to OO/.NET design
  - Follow .NET Framework guidelines
  - Manually design resulting API


MonoMac.AppKit
NSView
AppKit
NSView
C#
Objective-C

Objective-C APIs

- Easy to bind – Already OO
  - Every ObjectiveC class is mirrored to C#
- Subclassing
  - You can override Objective-C methods
  - Using standard C# constructs
- Maps C# delegates to Objective-C blocks
- Maps Objective-C Delegate pattern to C# events

Binding Objective-C

- Objective-C Selector:

```
sendMessage:(NSMessage*msg )
    toWindows:(NSWindow *)targetWindow
     withEvent:(NSEvent *) theEvent
```

- Must map to a C# style name:

```
SendMessage (
    NSMessage msg,
    NSWindow targetWindow,
    NSEvent theEvent);
```

Binding

- Curated API
  - We avoided automatic generation from APIs
  - Header parser does the heavy lifting
  - Then edit/maintain class-by-class
- We tweak the naming to match conventions
  - .NET Framework Design Guidelines
- Write glue by hand for doing the strong type setup.

Missing Features

- Some Lion APIs are missing
- More templates

Code Sharing and Native Experience