IOS451

Build an Objective-C bindings library with Objective Sharpie

Download class materials from university.xamarin.com

Microsoft

Xamarin University

Xamarin University

Recall: Create a binding manually

Creating a manual binding to a native library is a process which builds in complexity as the native library gets larger

Locate/create umbrella header

Create a class file for each enum, const, struct and protocol

Register the types

Export the members

Create an interface/abstract class for the protocols

Objectives

1. Download and evaluate Objective Sharpie
2. Create an API definition using Objective Sharpie
3. Consume the API definition to create a binding
4. Review/cleanup generated Binding and consume in a Xamarin.iOS project

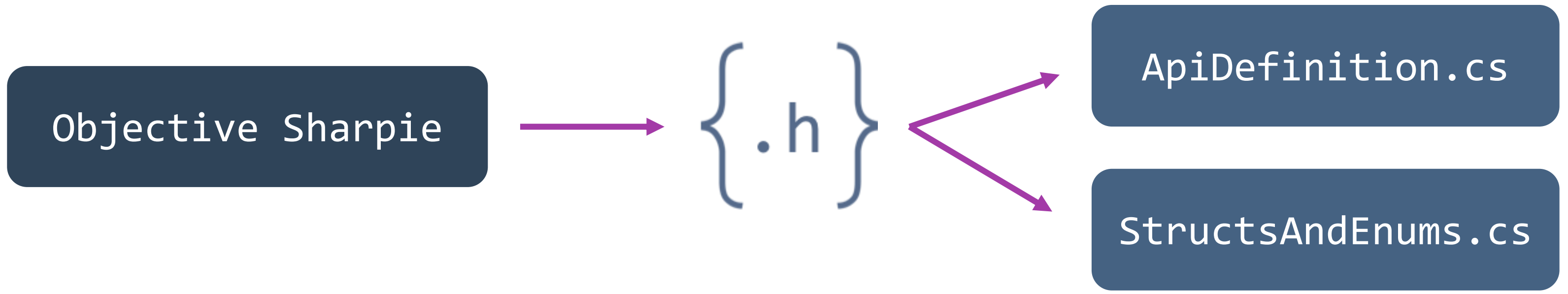
Download and evaluate Objective Sharpie

Tasks

1. Compare manual binding to binding with Objective Sharpie
2. Describe a binding definition
3. Download Objective Sharpie
4. Evaluate the Objective Sharpie tools
5. Determine the SDK versions

What is Objective Sharpie?

Objective Sharpie is a command line tool used to parse Objective-C header files (`*.h`) to map the API into an editable binding definition



What is a binding definition?

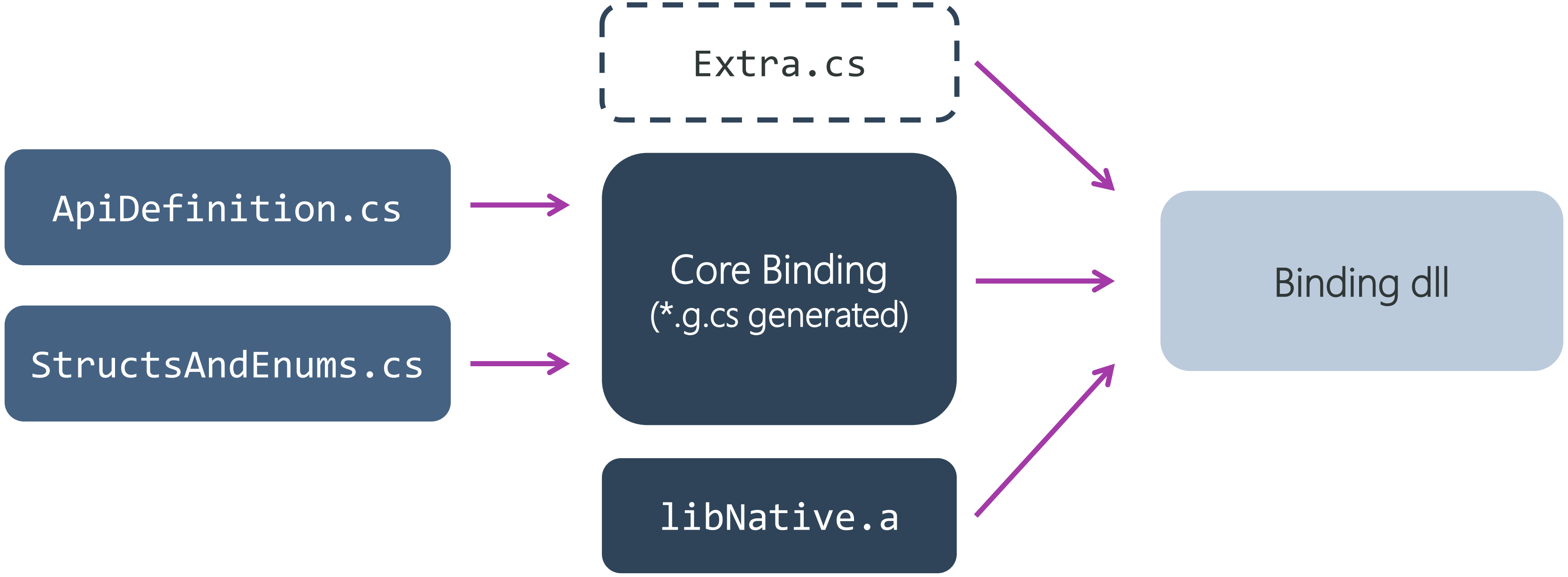
The binding definition is the contract containing the namespaces and interface definitions that are used to generate the API

```
namespace MBProgressHUD
{
   // typedef void (^MBProgressHUDCompletionBlock)();
   delegate void MBProgressHUDCompletionHandler();
   delegate void NSDispatchHandlerT();
   // @interface MBProgressHUD : UIView
   [BaseType(typeof(UIView), Name = "MBProgressHUD",
   Delegates = new string[] { "WeakDelegate" },
   Events = new Type[] { typeof(MBProgressHUDDelegate) })]
   interface MTMBProgressHUD
   {
       // + (MB_INSTANCETYPE)showHUDAddedTo:
         (UIView *)view animated:(BOOL)animated;
       [Static]
       [Export("showHUDAddedTo:animated:")]
       MTMBProgressHUD ShowHUD(UIView view, bool animated);
       ...
}
```

```
using System;
namespace MBProgressHUD
{
   public enum MBProgressHUDMode
   {
      /** Progress is shown using an UIActivity … */
      Indeterminate,
      /** Progress is shown using a round … */
      Determinate,
      /** Progress is shown using a horizontal … */
      DeterminateHorizontalBar,
      ...
   }
   ...
}
```

How definition files are used

At compile time the definition files are combined with the native library, and an optional Extras file to create the binding dll



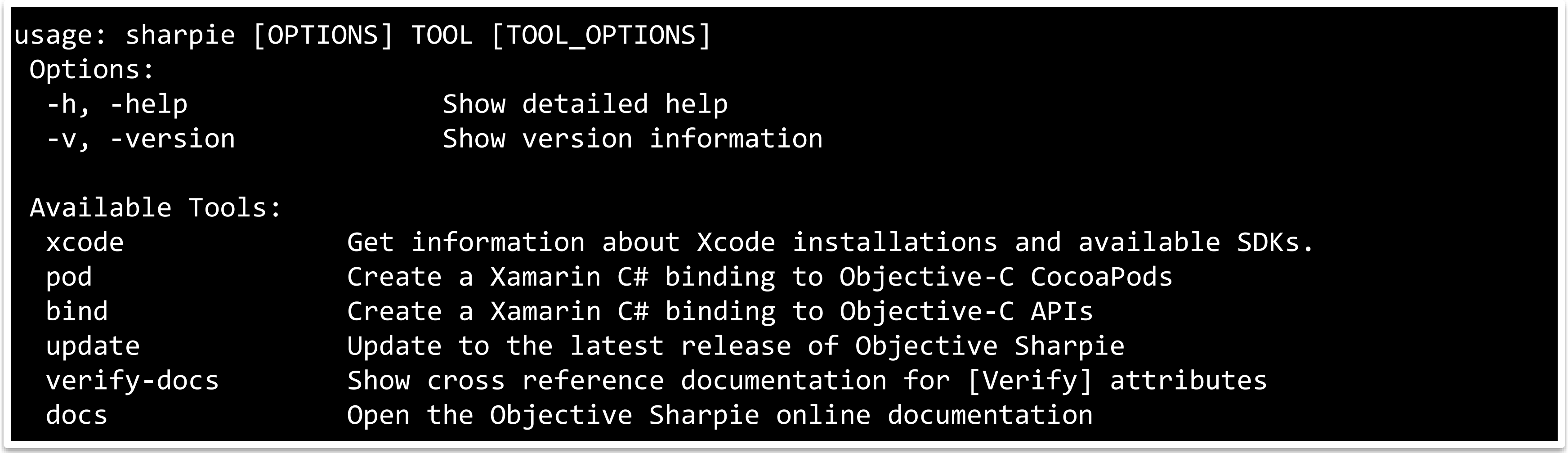
Download Objective Sharpie

Objective Sharpie is a separate downloadable component that run on macOS and requires several additional tools to be installed

- macOS 10.10+
- Xcode
- Xcode command line tools
- Apple SDKs you will bind against

Check the installation

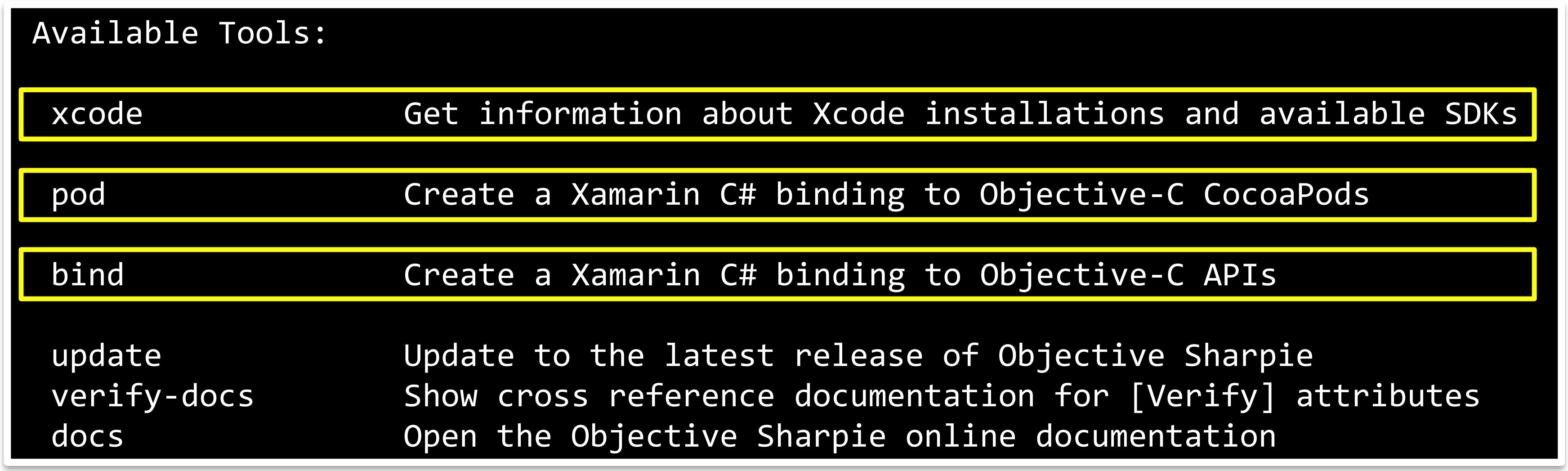
Run **`$ sharpie -help`** to verify the Objective Sharpie installation



```
usage: sharpie [OPTIONS] TOOL [TOOL_OPTIONS]
 Options:
  -h, -help                    Show detailed help
  -v, -version                 Show version information

 Available Tools:
  xcode                  Get information about Xcode installations and available SDKs.
  pod                    Create a Xamarin C# binding to Objective-C CocoaPods
  bind                   Create a Xamarin C# binding to Objective-C APIs
  update                 Update to the latest release of Objective Sharpie
  verify-docs            Show cross reference documentation for [Verify] attributes
  docs                   Open the Objective Sharpie online documentation
```

Objective Sharpie tools

There are three Objective Sharpie commands used to create bindings

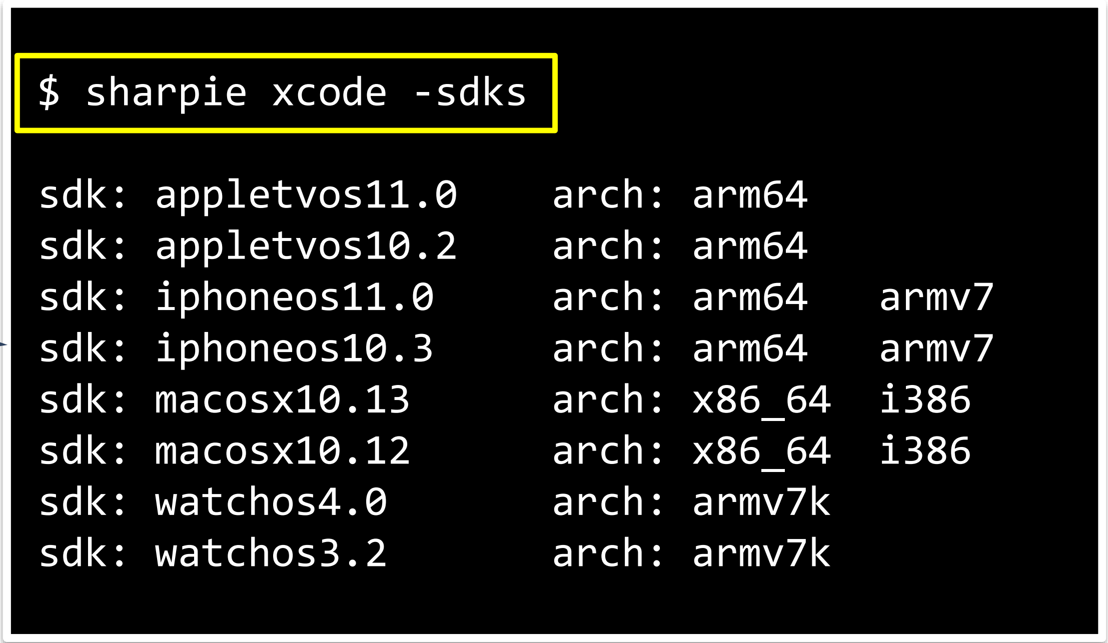


```
Available Tools:

  xcode              Get information about Xcode installations and available SDKs

  pod                Create a Xamarin C# binding to Objective-C CocoaPods

  bind               Create a Xamarin C# binding to Objective-C APIs

  update             Update to the latest release of Objective Sharpie
  verify-docs        Show cross reference documentation for [Verify] attributes
  docs               Open the Objective Sharpie online documentation
```

Determine the SDK versions

The SDK version must always be passed to the binding command

Some libraries may have a dependency against specific version of an SDK



```
$ sharpie xcode -sdks

sdk: appletvos11.0    arch: arm64
sdk: appletvos10.2    arch: arm64
sdk: iphoneos11.0     arch: arm64   armv7
sdk: iphoneos10.3     arch: arm64   armv7
sdk: macosx10.13      arch: x86_64  i386
sdk: macosx10.12      arch: x86_64  i386
sdk: watchos4.0       arch: armv7k
sdk: watchos3.2       arch: armv7k
```

Demonstration

Download Objective Sharpie and check the installation

Create an API definition using Objective Sharpie

Xamarin University

Tasks

1. Generate API definitions from an Xcode project
2. Generate API definitions from Cocoa Pods
3. Generate API definitions Manually

Three project styles

Objective Sharpie supports three project types you can create bindings from

Xcode projects

CocoaPods

Manual

Three steps to bind an Xcode project

There are three steps to produce the binding definition files for an Xcode project

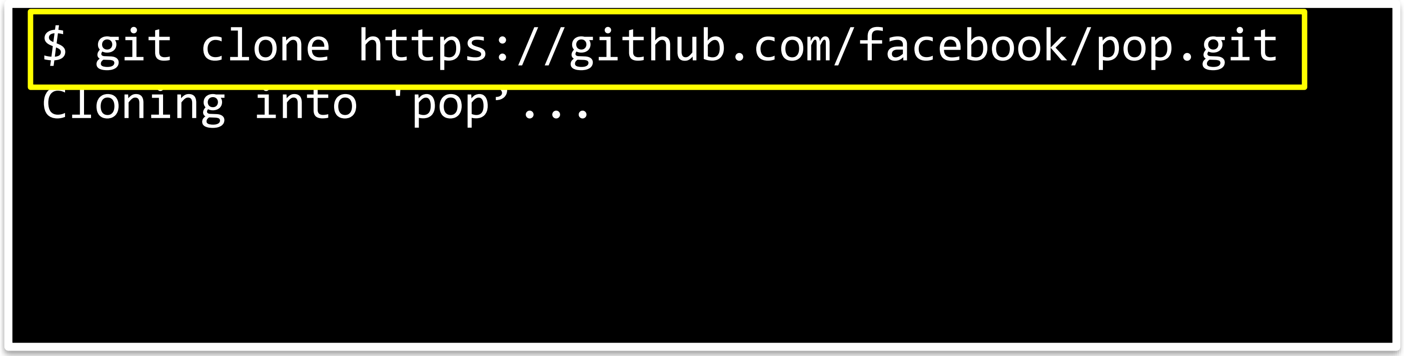
Clone the project

Point Objective Sharpie to the project file

Specify the SDK

Reference the library

Run **`$ git clone [url]`** to clone the the library



```
$ git clone https://github.com/facebook/pop.git
Cloning into 'pop'...
```

1 Clone the library

Point to the project

You must indicate where you want your project cloned to

```
$ git clone https://github.com/facebook/pop.git
 Cloning into 'pop'...
   ...

$ cd pop
```

Xamarin University

Specify the SDK

Objective Sharpie requires the SDK version to bind against

```
$ git clone https://github.com/facebook/pop.git
 Cloning into 'pop’...

$ cd pop
$ sharpie bind pop.xcodeproj -sdk iphoneos11.1
```

Demonstration

Initiate the binding process on an existing Xcode project

What are CocoaPods?

CocoaPods is a dependency manager for Objective-C Cocoa projects

CocoaPods are built with Ruby and are installable with the default OSX Ruby installation

Installation instruction detailed usage can be found at - https://cocoapods.org

Binding CocoaPods with Objective Sharpie

Objective Sharpie supports binding CocoaPods and allows you to download, configure, and build your binding definition files in two steps:

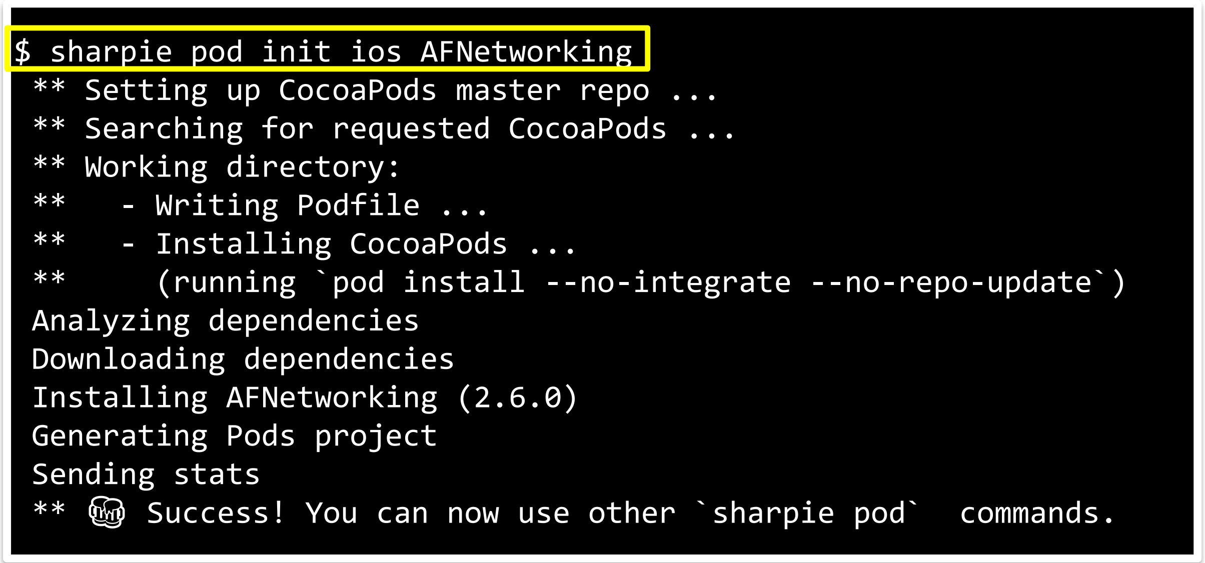
1. Initialize a CocoaPods binding project
2. Create the binding

Xamarin University

Configure and compile the CocoaPod

Run the initialization command for a CocoaPods project

Name of CocoaPod

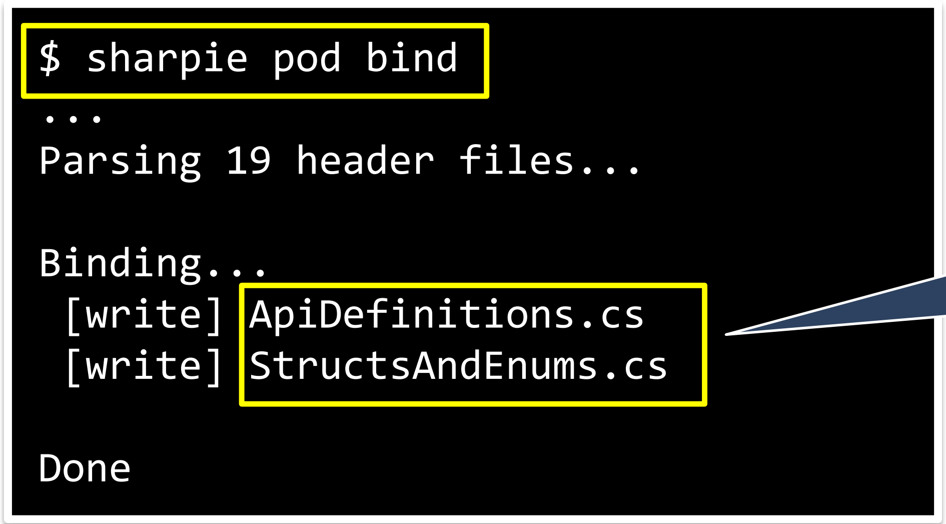


```
$ sharpie pod init ios AFNetworking
 ** Setting up CocoaPods master repo ...
 ** Searching for requested CocoaPods ...
 ** Working directory:
 **    - Writing Podfile ...
 **    - Installing CocoaPods ...
 **      (running `pod install --no-integrate --no-repo-update`)
 Analyzing dependencies
 Downloading dependencies
 Installing AFNetworking (2.6.0)
 Generating Pods project
 Sending stats
 ** 🍻 Success! You can now use other `sharpie pod`  commands.
```

1 Initialize a CocoaPods binding project

Run the CocoaPod binding command

Run the bind command to create your binding definition files



```
$ sharpie pod bind
...
Parsing 19 header files...

Binding...
  [write] ApiDefinitions.cs
  [write] StructsAndEnums.cs

Done
```

Generated API definition files

Demonstration

Initiate the binding process on an existing CocoaPod project

Xamarin University

Xamarin University

Manual binding with Objective Sharpie

You can take control over the binding process with Objective Sharpie

Source the Xcode project

Build the project using xcodebuild

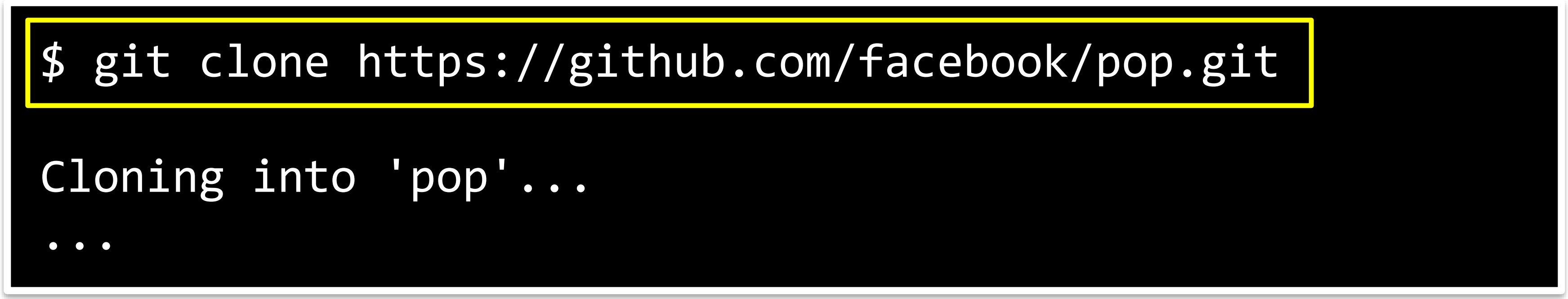
Command line tool similar to msbuild

Modify header files as required

Pass specific project parameters

Source the Xcode project

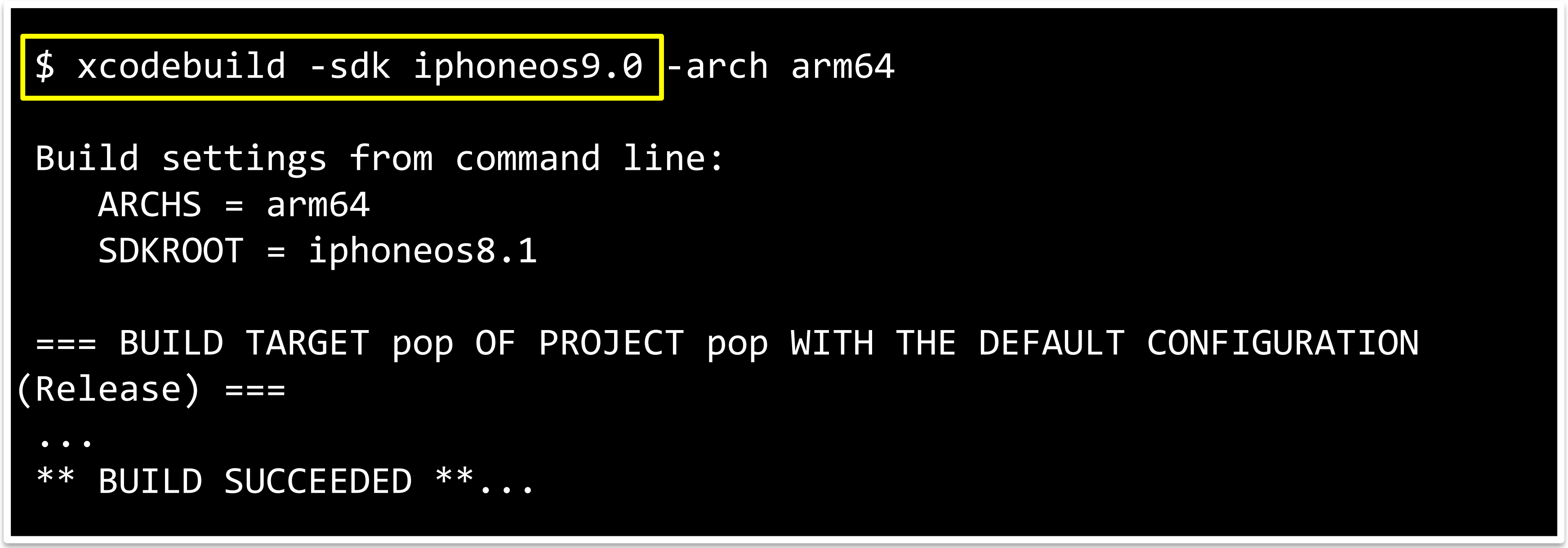
You can source a compiled library or an Xcode project for your binding



```
$ git clone https://github.com/facebook/pop.git

Cloning into 'pop'...
...
```

Build the Xcode project

The Xcode project must be built using Xcodebuild

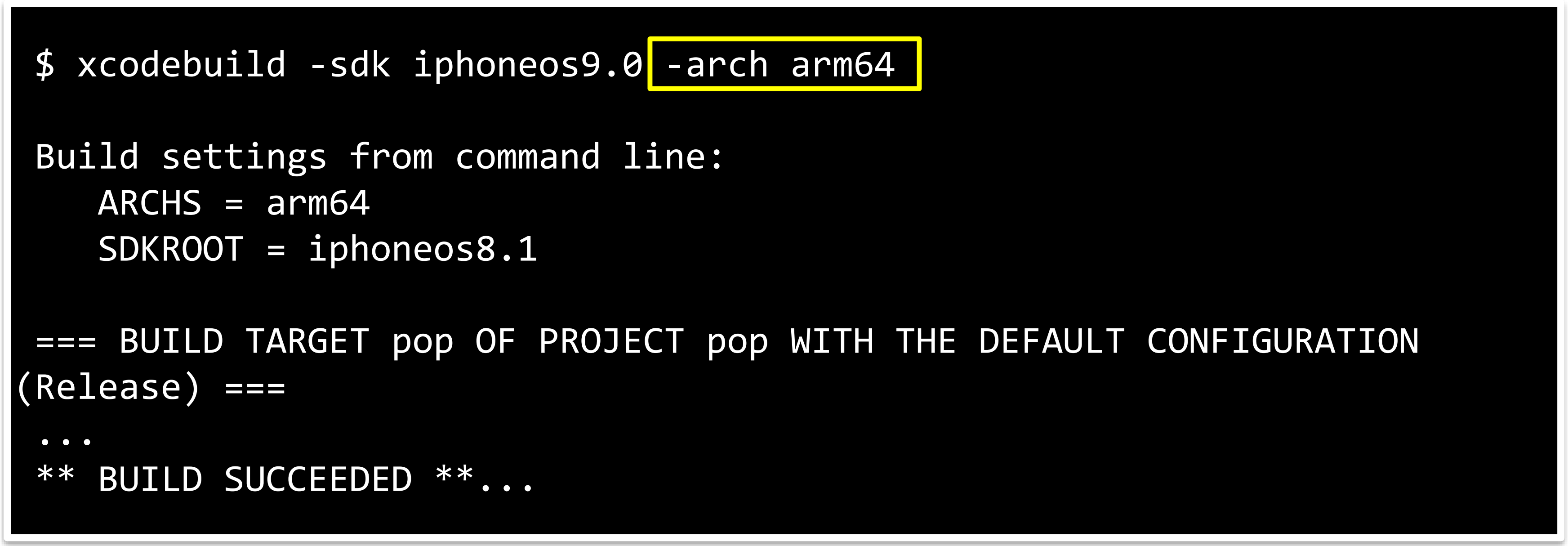


```
$ xcodebuild -sdk iphoneos9.0 -arch arm64

 Build settings from command line:
    ARCHS = arm64
    SDKROOT = iphoneos8.1

 === BUILD TARGET pop OF PROJECT pop WITH THE DEFAULT CONFIGURATION
(Release) ===
 ...
 ** BUILD SUCCEEDED **...
```

2 Build the project using Xcodebuild

Specify the architecture

Optionally, you can specify the target architecture before you build



```
$ xcodebuild -sdk iphoneos9.0 -arch arm64

Build settings from command line:
   ARCHS = arm64
   SDKROOT = iphoneos8.1

=== BUILD TARGET pop OF PROJECT pop WITH THE DEFAULT CONFIGURATION
(Release) ===
...
** BUILD SUCCEEDED **...
```

Locate the header files

The header files (`*.h`) must be located and supplied to Objective Sharpie as a command line parameter

Umbrella header file

```
$ ls build/Headers/POP/
 POP.h                     POPAnimationTracer.h     POPDefines.h
 POPAnimatableProperty.h   POPAnimator.h            POPGeometry.h
 POPAnimation.h            POPAnimatorPrivate.h     POPLayerExtras.h
 POPAnimationEvent.h       POPBasicAnimation.h      POPPropertyAnimation.h
 POPAnimationExtras.h      POPCustomAnimation.h     POPSpringAnimation.h
 POPAnimationPrivate.h     POPDecayAnimation.h
```

Xamarin University

Provide project-specific parameters

The Objective Sharpie **bind** command controls project parameters in the binding process

```
$ sharpie bind  \
    -output BindingOutputPath \
    -namespace FacebookPOP \
    -sdk iphoneos8.1 \
    -scope build/Headers \
     build/Headers/POP/POP.h \
    -c -Ibuild/Headers \
    -arch arm64
```

4 Bind the project with specific project parameters

The output command

The **output** command controls the location of the binding definition files

```
$ sharpie bind  \
   -output BindingOutputPath \
   -namespace FacebookPOP \
   -sdk iphoneos8.1 \
   -scope build/Headers \
    build/Headers/POP/POP.h \
   -c -Ibuild/Headers \
   -arch arm64
```

The location of the generated ApiDefintion.cs and StructsAndEnums.cs

The namespace argument

The **-namespace** argument defines the default namespace associated with the C# wrapper class API

```
$ sharpie bind  \
    -output BindingOutputPath \
    -namespace FacebookPOP \
    -sdk iphoneos8.1 \
    -scope build/Headers \
     build/Headers/POP/POP.h \
    -c -Ibuild/Headers \
    -arch arm64
```

Namespace is included in all code consuming the API via a C# **using** statement

Specify the SDK's

The SDK version must be included in your binding

```
$ sharpie bind  \
    -output BindingOutputPath \
    –namespace FacebookPOP \
    -sdk iphoneos8.1 \
    -scope build/Headers \
     build/Headers/POP/POP.h \
    -c -Ibuild/Headers \
    -arch arm64
```

The SDK version is a required parameter

4 Bind the project with specific project parameters

The scope argument

The **-scope** argument defines the path used to search for header files

```
$ sharpie bind  \
    -output BindingOutputPath \
    –namespace FacebookPOP \
    -sdk iphoneos8.1 \
    -scope build/Headers \
     build/Headers/POP/POP.h \
    -c -Ibuild/Headers \
    -arch arm64
```

Objective Sharpie ignores any API that is not defined in a file somewhere within the **-scope** path

Specify the header file

The header file supplies all of the API definitions to Objective Sharpie

```
$ sharpie bind  \
    -output BindingOutputPath \
    –namespace FacebookPOP \
    -sdk iphoneos8.1 \
    -scope build/Headers \
    build/Headers/POP/POP.h \
    -c -Ibuild/Headers \
    -arch arm64
```

You have to create an umbrella file or point to each header file individually

Reference the header files to clang

The **-c** argument defines all arguments passed onto the clang compiler

```
$ sharpie bind  \
    -output BindingOutputPath \
    -namespace FacebookPOP \
    -sdk iphoneos8.1 \
    -scope build/Headers \
     build/Headers/POP/POP.h \
    -c -Ibuild/Headers \
    -arch arm64
```

-Ibuild/Headers specifies that clang should only parse headers in the **build>Headers** path

Specify the architecture

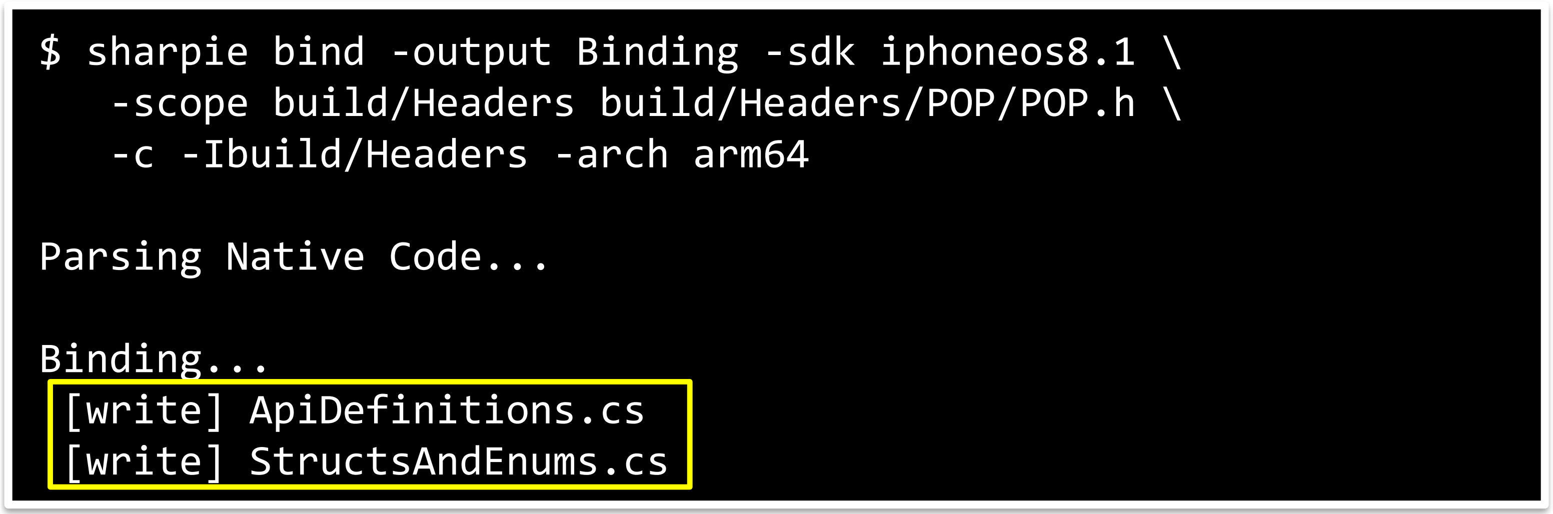
The binding architecture is required by some libraries to ensure we have the proper instructions

```
$ sharpie bind  \
    -output BindingOutputPath \
    -namespace FacebookPOP \
    -sdk iphoneos8.1 \
    -scope build/Headers \
     build/Headers/POP/POP.h \
    -c -Ibuild/Headers \
    -arch arm64
```

May be required for your specific library

Create the API Definition

Objective Sharpie generates an **ApiDefintions.cs** file and a **StructsAndEnums.cs** file if structs and/or enums are generated



```
$ sharpie bind -output Binding -sdk iphoneos8.1 \
   -scope build/Headers build/Headers/POP/POP.h \
   -c -Ibuild/Headers -arch arm64

Parsing Native Code...

Binding...
 [write] ApiDefinitions.cs
 [write] StructsAndEnums.cs
```

4 Bind the project with specific project parameters

Demonstration

When Sharpie fails

Individual Exercise

Initiate the binding process on an existing Xcode Framework project

Xamarin University

Consume the API definition to create a binding

Tasks

1. Describe the Xamarin.iOS Binding Project Structure
2. Set the Native Reference properties
3. Compile the project
4. Evaluate the errors

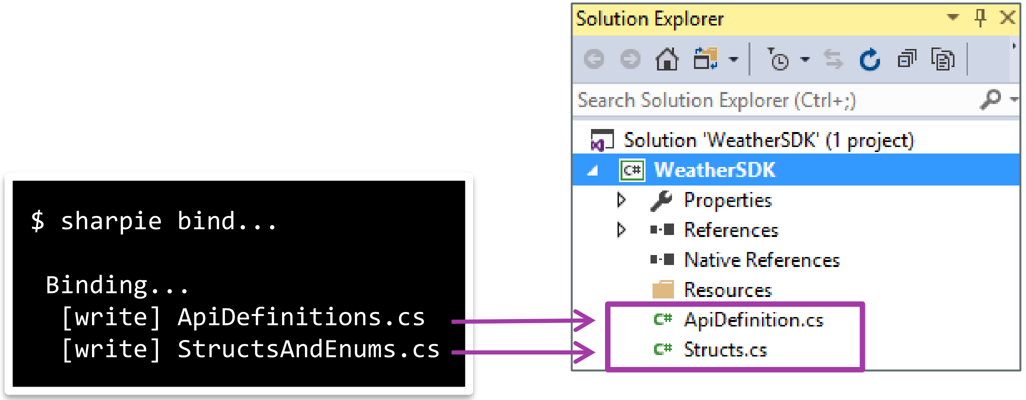
Consume the API definition files

There are four steps to consume the API definition files

- Bring in the API files
- Reference the library
- Compile the project
- Verify the files

Replace the default files

The **ApiDefinition.cs** and **StructsAndEnums.cs** from Objective Sharpie replace the default project files



```
$ sharpie bind...

 Binding...
  [write] ApiDefinitions.cs
  [write] StructsAndEnums.cs
```

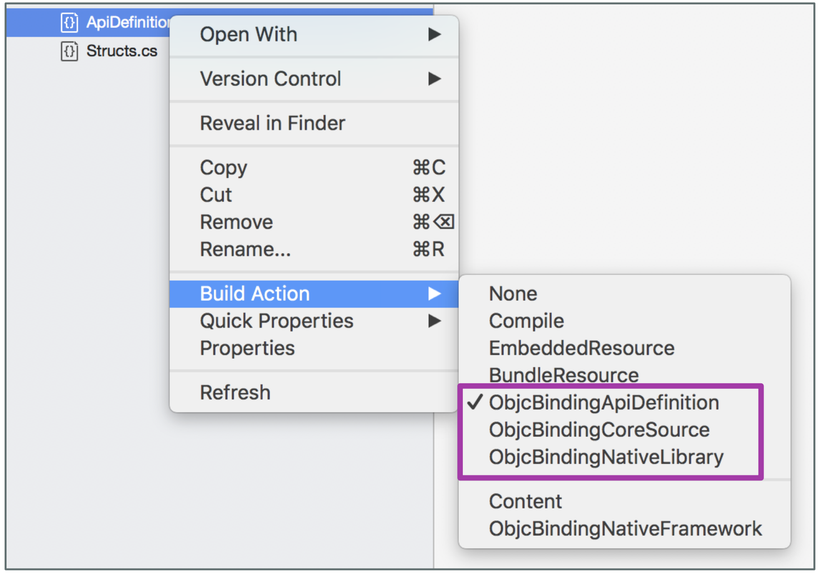
Demonstration

Replace the API definition files and compile the project

Xamarin University

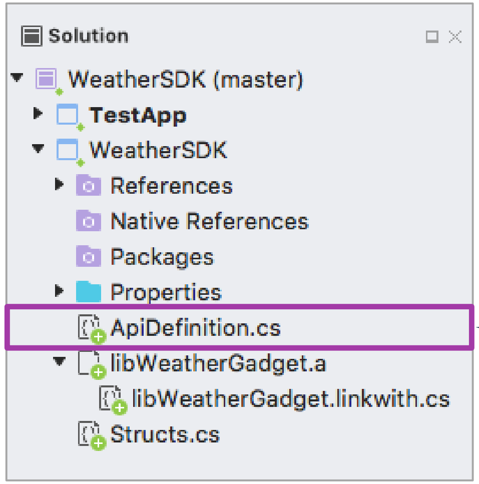
Project Anatomy [Build Actions]

There are three unique Build Actions you will use to bind your library



ObjcBindingApiDefinition

The **ObjcBindingApiDefinition** Build Action generates the wrapper classes

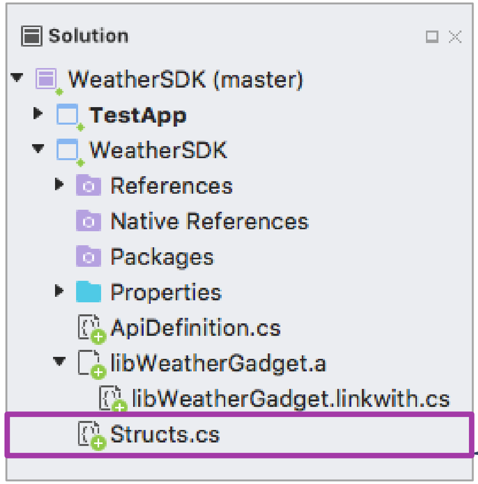


Solution
- WeatherSDK (master)
  - **TestApp**
  - WeatherSDK
    - References
    - Native References
    - Packages
    - Properties
    - ApiDefinition.cs
    - libWeatherGadget.a
      - libWeatherGadget.linkwith.cs
    - Structs.cs

This C# code is used to create the C# wrapper classes

ObjcBindingCoreSource

The **ObjcBindingCoreSource** Build Action indicates the file is used to support the C# generation process

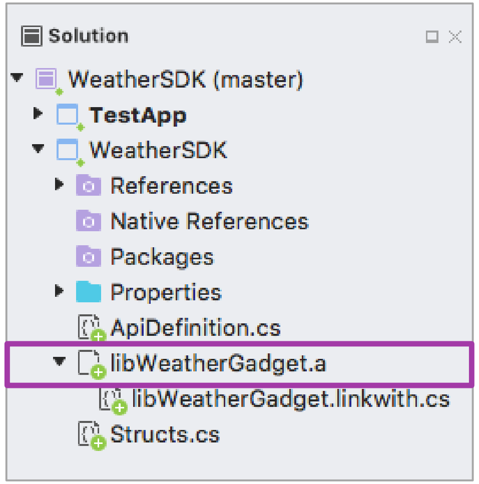


Solution
- WeatherSDK (master)
  - TestApp
  - WeatherSDK
    - References
    - Native References
    - Packages
    - Properties
    - ApiDefinition.cs
    - libWeatherGadget.a
      - libWeatherGadget.linkwith.cs
    - Structs.cs

Files compiled as C# code are set to `ObjcBindingCoreSource`

ObjcBindingNativeLibrary

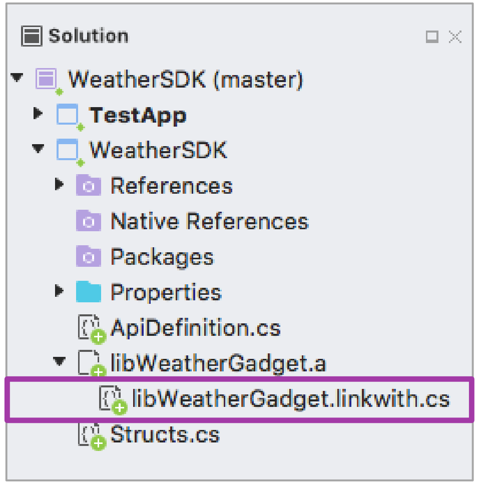
The **ObjcBindingNativeLibrary** Build Action indicates that this file is the native binary which will be linked into the final library



Bound static library included is set to `ObjcBindingNativeLibrary`

Pass additional parameters to the linker

Parameters must be passed to the linker to ensure that the native binary is properly included into the final binding library

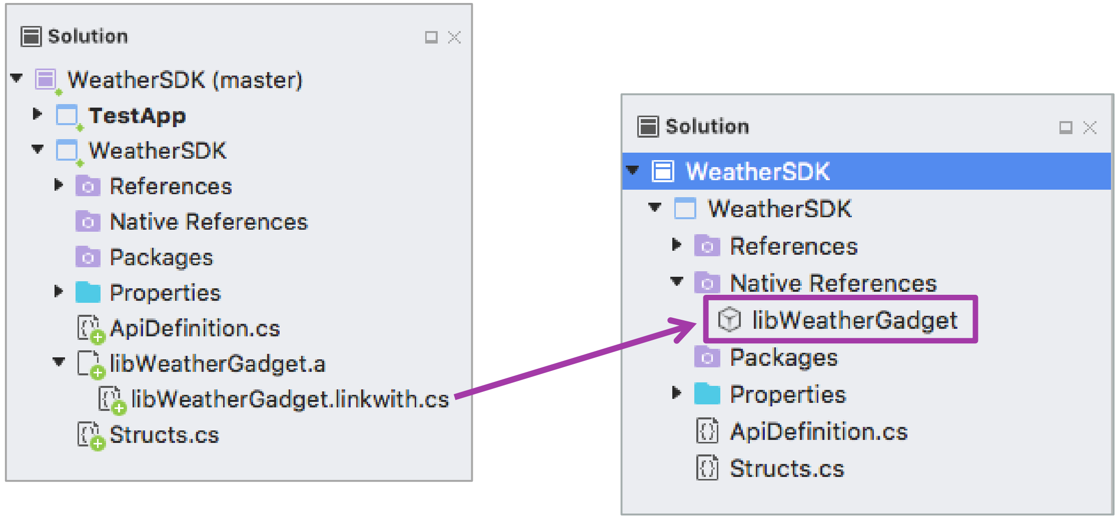


Created automatically with default linker parameters

Most of these properties are gcc and Xcode related options

Native References

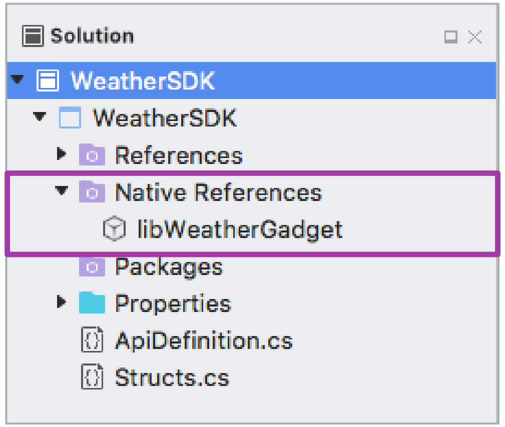
To simplify linking, we can move the libraries to the Native References container

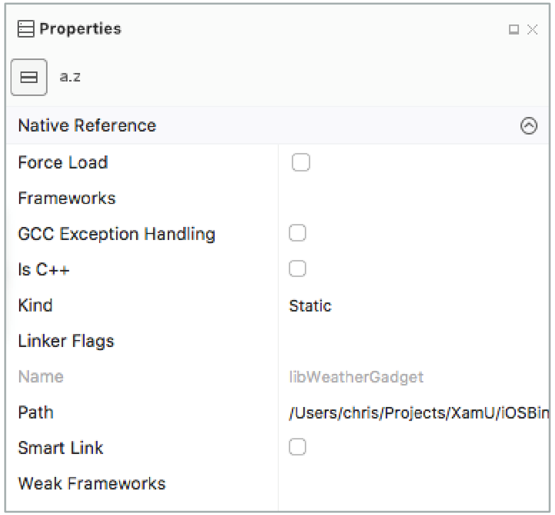


Xamarin University

Linker properties

The ***.linkwith.cs** file is replaced by a file properties dialogue



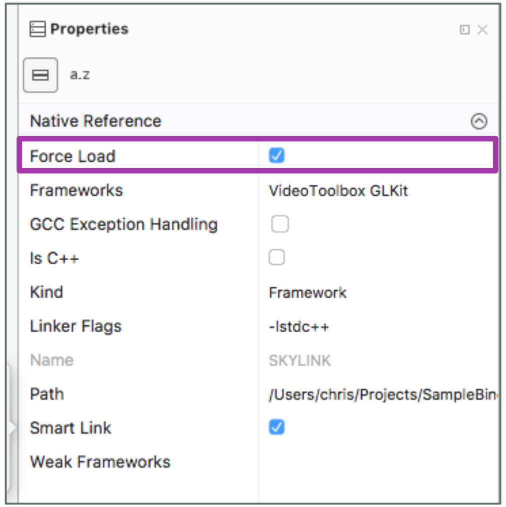


Native Reference

Force Load	☐
Frameworks	
GCC Exception Handling	☐
Is C++	☐
Kind	Static
Linker Flags	
Name	libWeatherGadget
Path	/Users/chris/Projects/XamU/iOSBin
Smart Link	☐
Weak Frameworks	

Properties – Force Load

Force Load ensures that all object files are linked from the native library



Properties

a.z

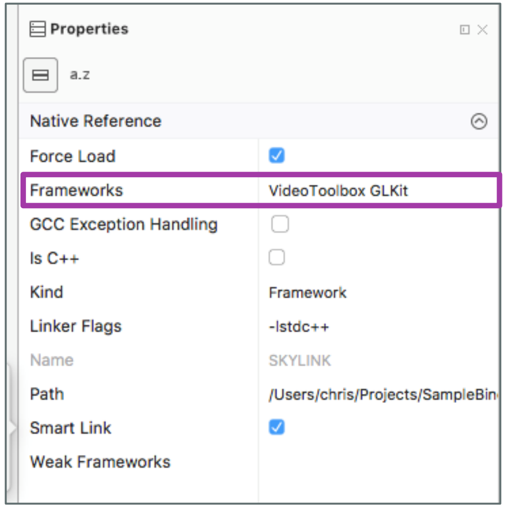
Native Reference

Force Load	✓
Frameworks	VideoToolbox GLKit
GCC Exception Handling	☐
Is C++	☐
Kind	Framework
Linker Flags	-lstdc++
Name	SKYLINK
Path	/Users/chris/Projects/SampleBin
Smart Link	✓
Weak Frameworks	

Xamarin University

Properties – Frameworks

Frameworks specifies the additional frameworks/libraries which are required by your native library



Properties

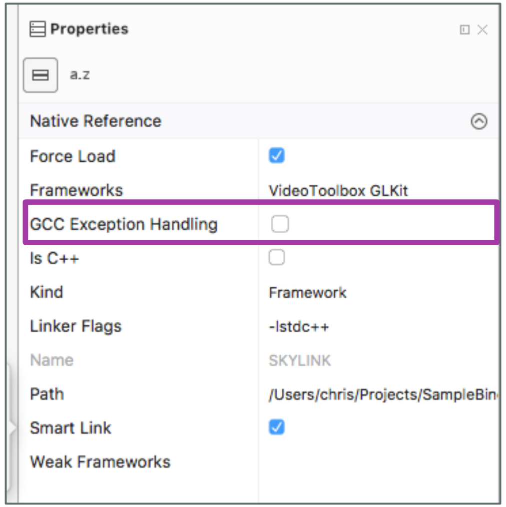
a.z

Native Reference

Force Load	✓
Frameworks	VideoToolbox GLKit
GCC Exception Handling	☐
Is C++	☐
Kind	Framework
Linker Flags	-lstdc++
Name	SKYLINK
Path	/Users/chris/Projects/SampleBin
Smart Link	✓
Weak Frameworks	

Properties – GCC Exception Handling

GCC Exception Handling ensures we are able to support stack unwinding


Properties

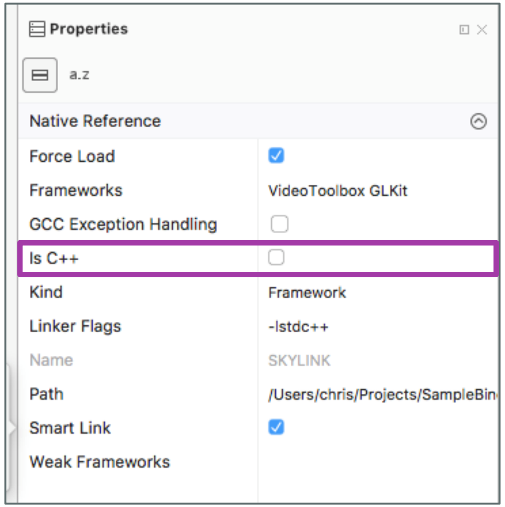
a.z

Native Reference

Force Load	☑
Frameworks	VideoToolbox GLKit
GCC Exception Handling	☐
Is C++	☐
Kind	Framework
Linker Flags	-lstdc++
Name	SKYLINK
Path	/Users/chris/Projects/SampleBin
Smart Link	☑
Weak Frameworks	

Properties – Is C++

Is C++ ensures the correct compiler is used with the native binary



Properties

a.z

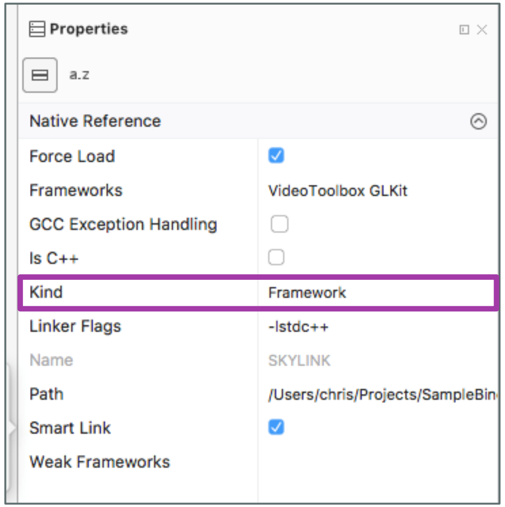
Native Reference

Force Load	✓
Frameworks	VideoToolbox GLKit
GCC Exception Handling	☐
Is C++	☐
Kind	Framework
Linker Flags	-lstdc++
Name	SKYLINK
Path	/Users/chris/Projects/SampleBin
Smart Link	✓
Weak Frameworks	

If your native library contains C++ code you must pass the cxx flag

Properties – Kind

Kind indicates if it is a **Static** or **Framework** library used in the binding



Properties

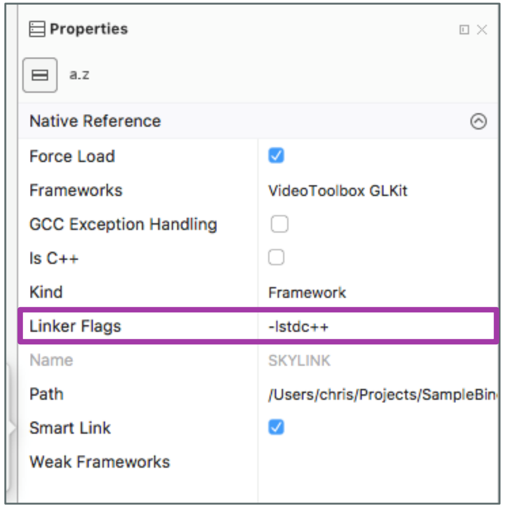
a.z

Native Reference

Force Load	✓
Frameworks	VideoToolbox GLKit
GCC Exception Handling	☐
Is C++	☐
Kind	Framework
Linker Flags	-lstdc++
Name	SKYLINK
Path	/Users/chris/Projects/SampleBin
Smart Link	✓
Weak Frameworks	

Pass Linker Flags directly to the compiler

Linker Flags specify which native libraries are required for linking


Properties

a.z

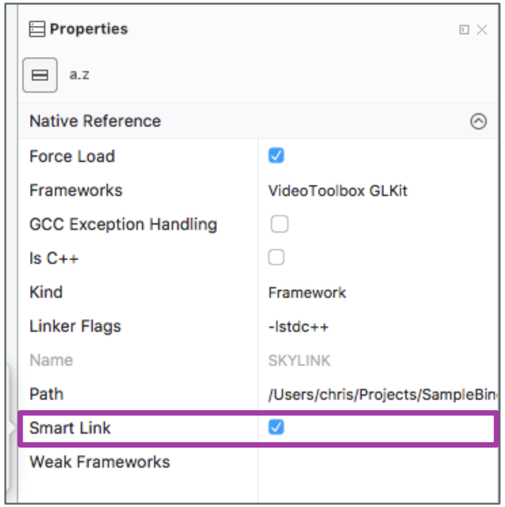
Native Reference

Force Load	✓
Frameworks	VideoToolbox GLKit
GCC Exception Handling	☐
Is C++	☐
Kind	Framework
Linker Flags	-lstdc++
Name	SKYLINK
Path	/Users/chris/Projects/SampleBin
Smart Link	✓
Weak Frameworks	

Consult the library or framework documentation for specific requirements

Properties – Smart Link

Smart Link allows you to ignore the ForceLoad value



Properties

a.z

Native Reference

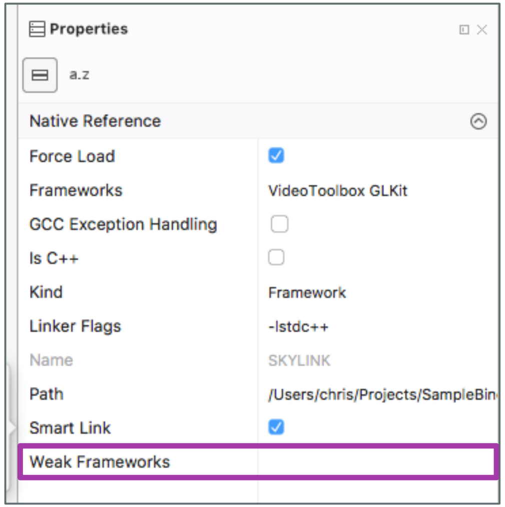
Force Load	✓
Frameworks	VideoToolbox GLKit
GCC Exception Handling	☐
Is C++	☐
Kind	Framework
Linker Flags	-lstdc++
Name	SKYLINK
Path	/Users/chris/Projects/SampleBin
Smart Link	✓
Weak Frameworks	

ForceLoad value will be ignored

The **ForceLoad** flag is usually not required when the static registrar is used at compilation

Properties – Weak Frameworks

Weak Frameworks allows for feature compatibility in libraries



Properties

a.z

Native Reference

Force Load	✓
Frameworks	VideoToolbox GLKit
GCC Exception Handling	☐
Is C++	☐
Kind	Framework
Linker Flags	-lstdc++
Name	SKYLINK
Path	/Users/chris/Projects/SampleBin
Smart Link	✓
Weak Frameworks	

Frameworks are listed as space delimited string values

Demonstration

Test our binding

Xamarin University

Review/cleanup generated Binding and consume in a Xamarin.iOS project

Xamarin University

Tasks

1. Describe how the Binding definition files are used
2. Use Attributes to cleanup the Binding definition files

ApiDefinition.cs [Interpreted]

Objective Sharpie creates the ApiDefinition.cs file by interpreting the umbrella header file and all/some of the **`#include`** or **`#import`** headers

```
#import <Wearable/WXAMAccelerometer.h>
#import <Wearable/WXAMAccelerometerData.h>
#import <Wearable/WXAMAmbientLight.h>
#import <Wearable/WXAMBarometer.h>
#import <Wearable/WXAMConstants.h>
#import <Wearable/WXAMData.h>
#import <Wearable/WXAMGPIO.h>
...
```

Umbrella header file

```
// typedef void (^WXAMVoidHandler)();
delegate void WXAMVoidHandler();

// typedef void (^WXAMErrorHandler)(NSError * _Nullable);
delegate void WXAMErrorHandler([NullAllowed] NSError arg0);

// typedef void (^WXAMDataHandler) ...
delegate void WXAMDataHandler([NullAllowed] NSData arg0, [N
ullAllowed] NSError arg1);
...
```

API definition file

[Verify] Attribute

Objective Sharpie emits a **`[Verify]`** attribute when there is not enough metadata in the original C/Objective-C declaration

The **`[Verify]`** attributes intentionally cause C# compilation errors so you are forced to verify the binding

```
[Native]
[Verify (InferredFromMemberPrefix)]
Public enum kCFSocket : nuint
{
        AutomaticallyReenableReadCallBack = 1,
        AutomaticallyReenableAcceptCallBack = 2,
        AutomaticallyReenableDataCallBack = 3,
        AutomaticallyReenableWriteCallBack = 8,
        LeaveErrors = 64,
        CloseOnInvalidate = 128
}
```

[Verify] annotations

Objective Sharpie annotates the **Verify** attributes to provide you with a hint of what needs to be confirmed

```
[Native]
[Verify (InferredFromMemberPrefix)]
Public enum kCFSocket : nuint
{
      AutomaticallyReenableReadCallBack = 1,
      AutomaticallyReenableAcceptCallBack = 2,
      AutomaticallyReenableDataCallBack = 3,
      AutomaticallyReenableWriteCallBack = 8,
      LeaveErrors = 64,
      CloseOnInvalidate = 128
}
```

[InferredFromMemberPrefix]

Objective Sharpie will infer the name of an anonymous type from the common prefix of its members

```
[Native]
[Verify (InferredFromMemberPrefix)]
Public enum kCFSocket : nuint
{
      AutomaticallyReenableReadCallBack = 1,
      AutomaticallyReenableAcceptCallBack = 2,
      AutomaticallyReenableDataCallBack = 3,
      AutomaticallyReenableWriteCallBack = 8,
      LeaveErrors = 64,
      CloseOnInvalidate = 128
}
```

```
[Flags]
Public enum CFSocketFlags
{
      AutomaticallyReenableReadCallBack = 1,
      AutomaticallyReenableAcceptCallBack = 2,
      AutomaticallyReenableDataCallBack = 3,
      AutomaticallyReenableWriteCallBack = 8,
      LeaveErrors = 64,
      CloseOnInvalidate = 128
}
```

[MethodToProperty]

Methods may need to be bound as properties to surface a nicer API

```
// -(BOOL)getRts;
[Export ("getRts")]
[Verify (MethodToProperty)]
bool Rts { get; }
```

```
// -(BOOL)getRts;
[Export ("getRts")]
bool Rts { get; }
```

[ConstantsInterfaceAssociation]

Objective Sharpie requires you create class associations for defined constants

```
[Static]
[Verify
(ConstantsInterfaceAssociation)]
partial interface Constants {
     [Field ("kSecMatchLimitOne")]
     IntPtr MatchLimitOne { get; }

     [Field ("kSecMatchLimitAll")]
     IntPtr MatchLimitAll { get; }
}
```

```
[Static]
interface SecMatchLimit {
     [Field ("kSecMatchLimitOne")]
     IntPtr MatchLimitOne { get; }

     [Field ("kSecMatchLimitAll")]
     IntPtr MatchLimitAll { get; }
}
```

Xamarin University

[StronglyTypedNSArray]

Objective Sharpie uses generic types where it cannot infer the actual type

```
// @interface SMRespondent : NSObject <SMJSONSerializableProtocol>
[BaseType (typeof(NSObject))]
interface SMRespondent : ISMJSONSerializableProtocol
{
    // @property (nonatomic, strong) NSArray * questionResponses;
    [Export ("questionResponses", ArgumentSemantic.Strong)]
    [Verify (StronglyTypedNSArray)]
    NSObject[] QuestionResponses { get; set; }
}
```

[StronglyTypedNSArray]

Objective Sharpie uses generic types where it cannot infer the actual type

```
// @interface SMRespondent : NSObject <SMJSONSerializableProtocol>
[BaseType (typeof(NSObject))]
interface SMRespondent : ISMJSONSerializableProtocol
{
    // @property (nonatomic, strong) NSArray * questionResponses;
    [Export ("questionResponses", ArgumentSemantic.Strong)]
    SMQuestionResponse[] QuestionResponses { get; set; }
}
```

Xamarin University

[PlatformInvoke]

P/Invoke statements must be verified or removed because P/Invoke bindings are not as correct or complete as Objective-C bindings

```
// extern void CLSLog (NSString * format, ...);
[DllImport("__Internal", EntryPoint = "CLSLog")]
[Verify (PlatformInvoke)]
interface static extern void __CLSLog(IntPtr format, string arg0);
```

```
// extern void CLSLog (NSString * format, ...);
[DllImport("__Internal", EntryPoint = "CLSLog")]
interface static extern void __CLSLog(IntPtr format, string arg0);
```

For P/Invoke guidance, see http://www.mono-project.com/docs/advanced/pinvoke/

[DisableDefaultCtor] – Constructors

When the C# wrappers are created, the Xamarin binding tool generates default constructors

```
[Register("CCSequence", true)]
public partial class CCSequence : NSobject
{
   [Export("init")]
   public CCSequence() : base(NSObjectFlag.Empty)
   {
       InitializeHandle(ApiDefinition.Messaging.IntPtr_objc_msgSend(this.Handle,
                                         Selector.GetHandle("init")), "init");
   }
   ...
}
```

Xamarin University

[DisableDefaultCtor] – Constructors

The default constructors can be marked for removal using the `[DisableDefaultCtor]` attribute

```
[BaseType(typeof(CCActionInterval))]
[DisableDefaultCtor] // Objective-C exception thrown.
Name: NSInternalInconsistencyException Reason: IntervalActionInit: Init
 not supported. Use InitWithDuration

interface CCSequence
{
   //...
}
```

Xamarin University

[DesignatedInitializer]

You can indicate a default constructor using the `[DesignatedInitializer]` attribute

```
[BaseType(typeof(UIViewController))]
interface XAMBViewController
{
   // (instancetype _Nullable)initWithCoder:(NSCoder * _Nonnull)…
   [Export("initWithCoder:")]
   [DesignatedInitializer]
   IntPtr Constructor(NSCoder aDecoder);
   ...
}
```

Individual Exercise

Clean-up and complete an iOS bindings project

Thank You!

Please complete the class survey in your profile:
university.xamarin.com/profile

Microsoft