IOS450

Building an Objective-C Bindings Library

Download class materials from
university.xamarin.com

Microsoft

Xamarin University

Information in this document is subject to change without notice. The example companies, organizations, products, people, and events depicted herein are fictitious. No association with any real company, organization, product, person or event is intended or should be inferred. Complying with all applicable copyright laws is the responsibility of the user.

Microsoft or Xamarin may have patents, patent applications, trademarked, copyrights, or other intellectual property rights covering subject matter in this document. Except as expressly provided in any license agreement from Microsoft or Xamarin, the furnishing of this document does not give you any license to these patents, trademarks, or other intellectual property.

Xamarin, MonoTouch, MonoDroid, Xamarin.iOS, Xamarin.Android, Xamarin Studio, and Visual Studio are either registered trademarks or trademarks of Microsoft in the U.S.A. and/or other countries.

Other product and company names herein may be the trademarks of their respective owners.

Objectives

1. Bind an iOS library manually
2. Describe Frameworks & Libraries

Bind an iOS library manually

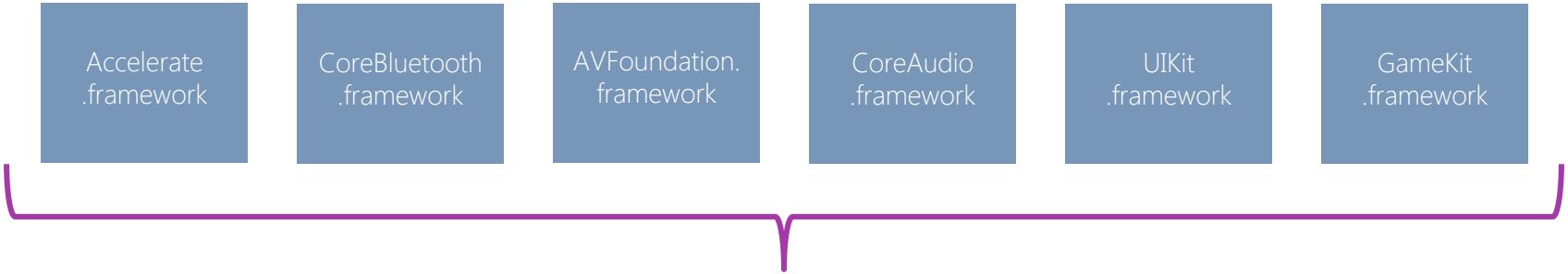
Xamarin University

Tasks

1. Consume static & dynamic libraries
2. Define Obj-C messages and selectors
3. Expose your wrapper class with the Xamarin.iOS Type Registrar

Frameworks and libraries

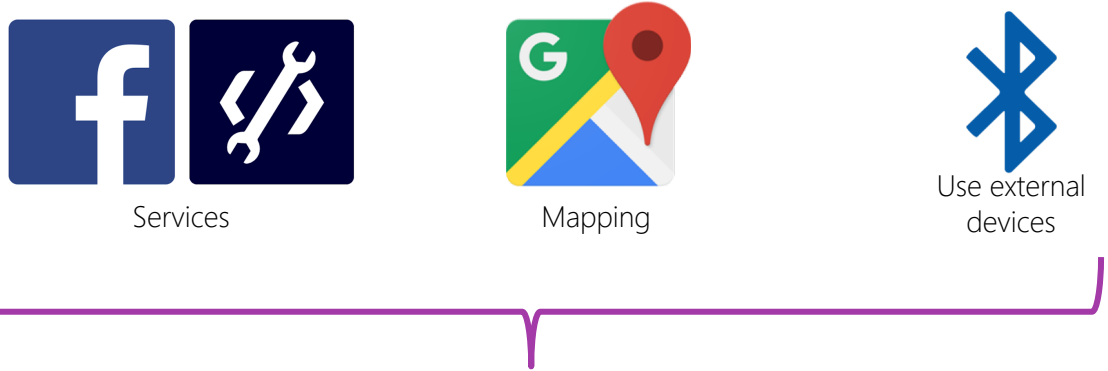
Apple publishes Frameworks & Libraries which allow developers to use features built into iOS, tvOS and macOS in their applications



Xamarin exposes these Frameworks to developers in the Xamarin.iOS, Xamarin.tvOS & Xamarin.Mac class libraries

3rd Party libraries

3rd Party device and service vendors publish Frameworks / Libraries to provide access services and/or devices



The core Xamarin libraries do not expose these frameworks and libraries

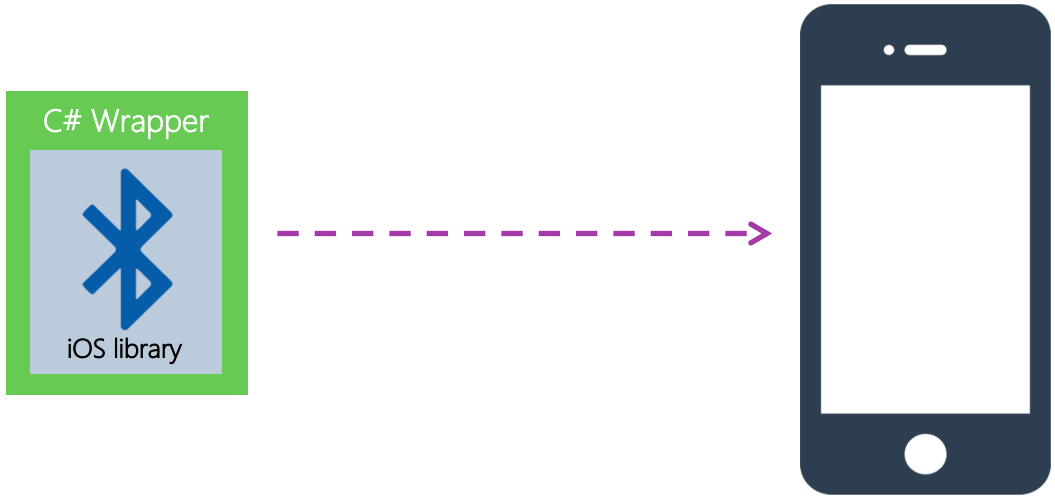
Demo

Investigate an existing Library to bind

Xamarin University

What is a Binding Library?

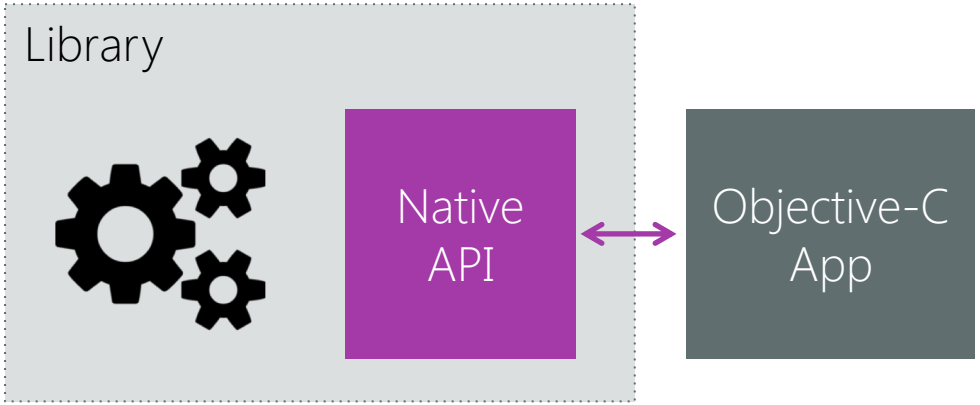
A *Binding Library* is a Xamarin project type that can wrap one or more existing Objective-C frameworks or libraries into a managed assembly



We will focus on iOS bindings, though the concepts are similar for macOS and tvOS

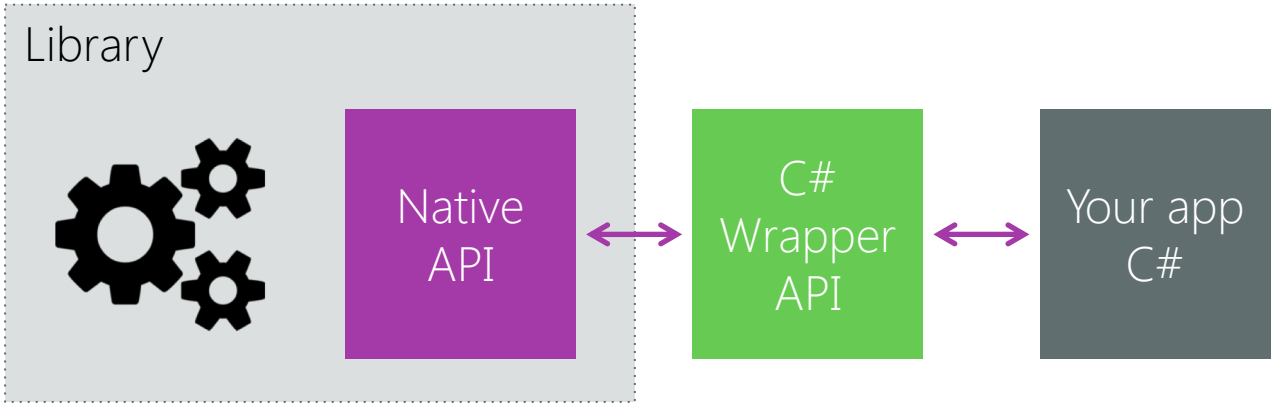
Accessing native objects with Objective-C

With native development we directly access libraries



Accessing native objects with Xamarin.iOS

With Xamarin.iOS development we need an intermediary to access native libraries



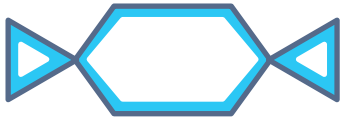
Demo

Investigate existing P/Invoke code

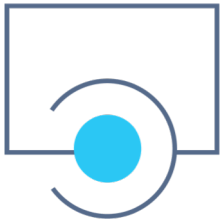
Xamarin University

Steps to building the C# wrapper

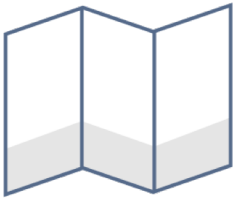
There are three main steps to building a C# wrapper



Create a C# wrapper class



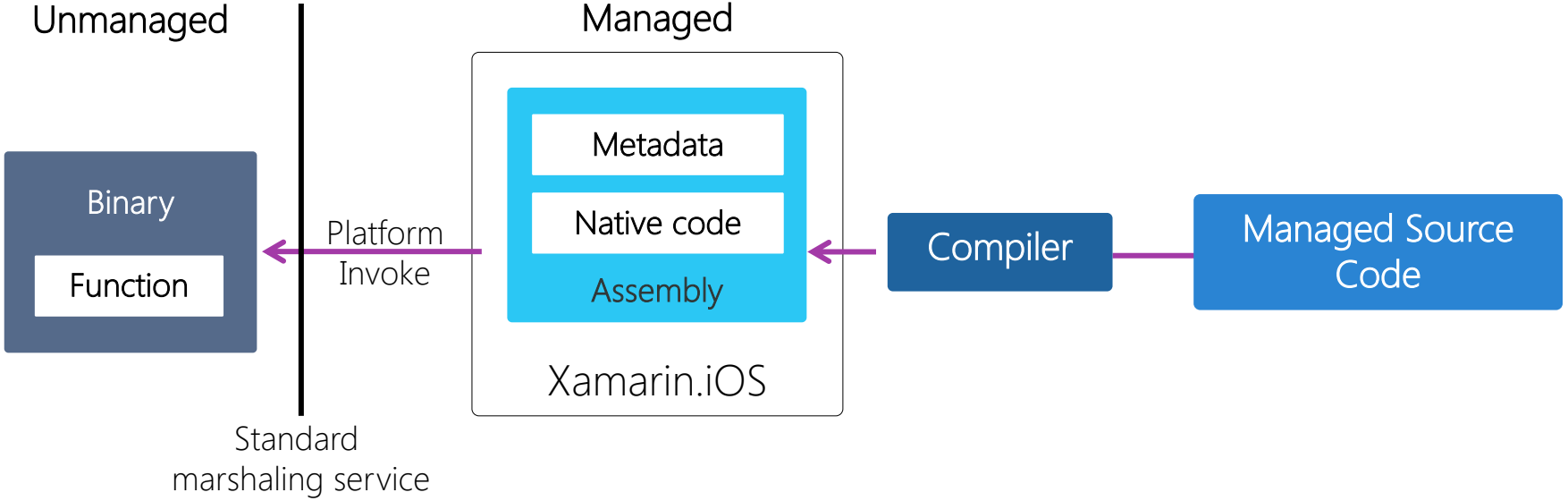
Expose the C# API



Map the C# API to the native API

What is P/Invoke?

Platform Invoke (P/Invoke) is a service provided by the Xamarin.iOS layer for calling unmanaged flat APIs from managed code



Implement a P/Invoke

In order to access a Objective-C/C++ function you will need to implement the P/Invoke which requires several steps

1. Determine which C function you want to invoke
2. Define the function's signature
3. Determine which library the function lives in
4. Write the appropriate P/Invoke declaration

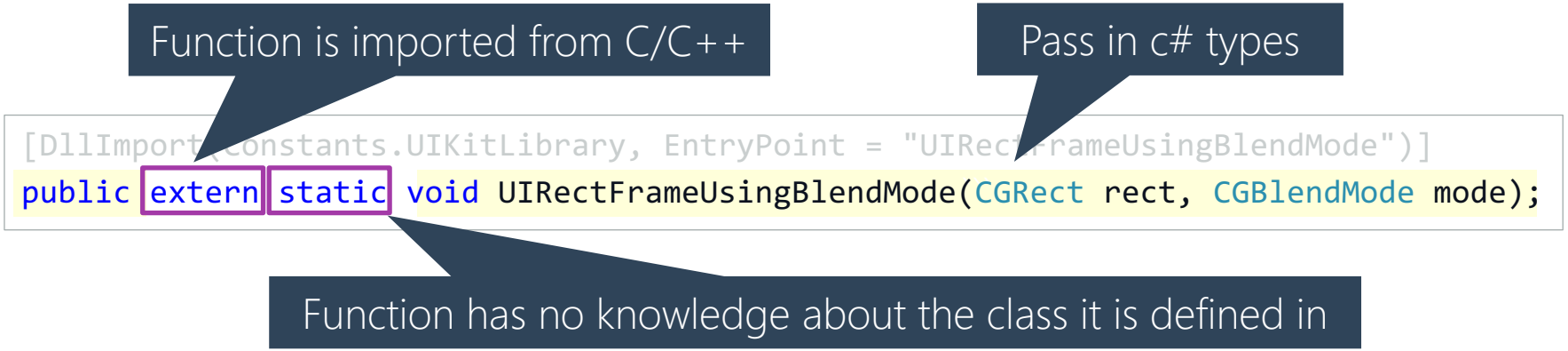
Determine the function to invoke

You'll need to find the member of the native library that you want to access from your managed code

```
void UIRectFrameUsingBlendMode(CGRect rect, CGBlendMode mode);
```

Define the C# API signature

Define the API as it will be used from within the managed code



Specify the library

Specify the library the function is invoked from as well as the entry point of the function in the library, i.e. the function name

```
[DllImport(Constants.UIKitLibrary, EntryPoint = "UIRectFrameUsingBlendMode")]
public extern static void UIRectFrameUsingBlendMode(CGRect rect, CGBlendMode mode);
```

Add code to the C# library

Place the code in your managed code library with the P/invoke statement to access your chosen native member

```
public static ShapesDecorator
{
    ...

    [DllImport(Constants.UIKitLibrary, EntryPoint = "UIRectFrameUsingBlendMode")]
    public extern static void UIRectFrameUsingBlendMode(CGRect rect, CGBlendMode mode);
}
```

Access dynamic C library methods

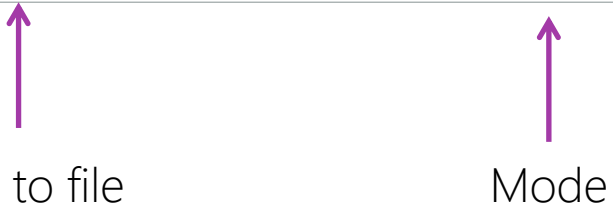
Dynamic libraries must be loaded by the Xamarin.iOS runtime before they can be used

1. Load the dynamic library first
2. Write the appropriate P/Invoke declaration

Use dlopen to load the library

The **dlopen** static method, exposed by the Xamarin.iOS runtime, is used to load and link a dynamic library or bundle

```
MonoTouch.ObjCRuntime.Dlfcn.dlopen ("/full/path/to/libSome.dylib", 0);
```



1 Load the dynamic library first

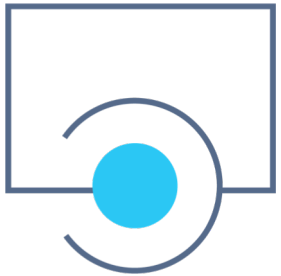
Use DllImport to add the library

Complete the **`DllImport`** statement and add to your C# library,

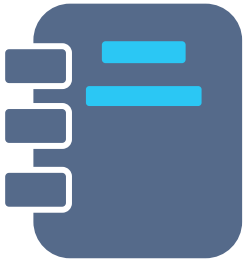
```
public static ShapesDecorator
{
    ...

    [DllImport("libShapes.dylib", EntryPoint="libVersion")]
    public static extern double LibraryVersion();
}
```

API vs library

An API describes the expected behaviors of a system while a library provides the implementation of those behaviors



API



Library

Headers files

Header files are created to separate the API from their implementation

Header file are suffixed *.h

Weather.h

```
#import <Foundation/Foundation.h>

@interface Weather : NSObject
{
    ...
}

@property(nonatomic, readwrite) double
currentTemperature;
@end
```

Implementation files are suffixed *.m

Weather.m

```
#import <Foundation/Foundation.h>
#import "Weather.h"

double _currentTemperature = 0;

@implementation Weather
{
    ...
}
@end
```

Umbrella headers

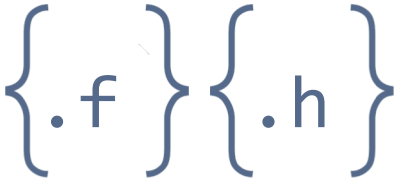
Umbrella headers are top-level header files that include references to all of the header files in the framework

```
#import <UIKit/UIKit.h>
```

```
#import <UIKit/UIViewController.h>
#import <UIKit/UILabel.h>
#import <UIKit/UIButton.h>
#import <UIKit/UIDatePicker.h>
```

Frameworks and umbrella headers

When binding a framework, umbrella headers will typically adhere to certain guidelines



Umbrella header file must have the same name as the framework

Only required to bind against a single header file

There may be instances where you need to change the header file by hand

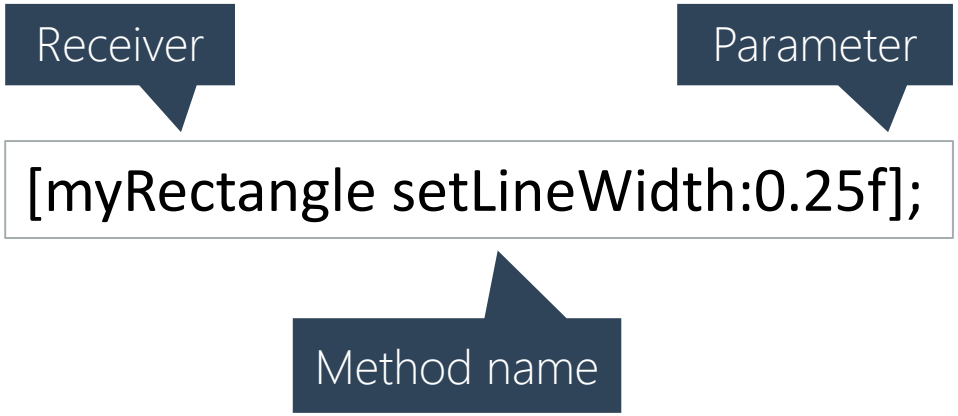
Demo

Investigate objc_msgSend P/Invoke

Xamarin University

What is an Obj-C message?

An Obj-C message is an expression comprised of a method name and its associated parameters used to execute a method on an object



Sending messages

In Objective-C applications, objects send messages to other objects to get work done (methods, fields, properties)

```
tempWeatherService.FetchWeather(location);
```

C#

```
[tempWeatherService fetchWeather:location];
```

Objective-C

The receiver of the message, i.e. the instantiated object

The message, i.e. the method to call

What is a selector?

Method names in messages are referred to as *selectors* in Objective-C

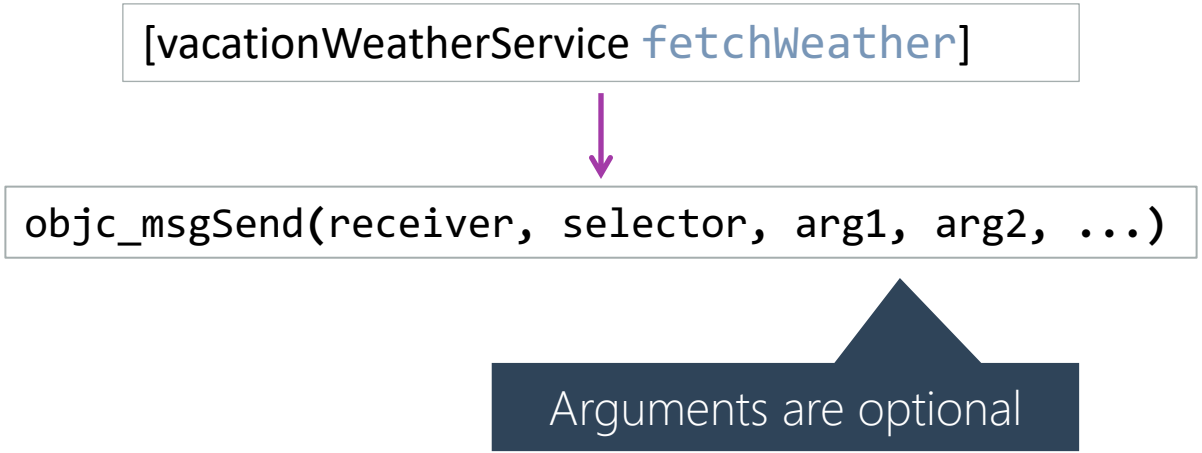
```
@selector(methodname)
```

A *Selector* is created from a string, and is used to send a message whose name may not be known until runtime

By itself a selector does not do anything, it only identifies a method

What is obj_msgSend?

In Objective-C, **`objc_msgSend`** is a function which sends a message to an instance of a class



What is libobjc?

The Objective-C runtime library (**libobjc**) is a runtime library that supports the dynamic properties of Objective-C and contains **objc_msgSend**

```
const string LIBOBJC_DYLIB = "/usr/lib/libobjc.dylib";
```

Functions are defined in libojbc.dylib

Import objc_msgSend

In Xamarin.iOS applications, you will import **libobjc** with **objc_msgSend** as the entry-point to access the objective-C runtime library

```
const string LIBOBJC_DYLIB = "/usr/lib/libobjc.dylib";

[DllImport(LIBOBJC_DYLIB, EntryPoint = "objc_msgSend")]
public static extern IntPtr IntPtr_objc_msgSend(IntPtr receiver, IntPtr
selector);
```


The macOS implementation of the Objective-C runtime library is unique to the Mac and available in Xamarin.Mac

Send a message with Xamarin.iOS

```
-(void) processWeather;
```

Class level definition

```
[tempWeatherService processWeather]
```

Message implementation

C# method invocation

```
void_objc_msgSend (this.Handle, Selector.GetHandle ("processWeather"));
```

`IntPtr` to instance of class receiving message

Selector of the method handling the message

Question

Exposing C# wrapper classes

In addition to consuming our libraries, we also allow these wrapper classes to be exposed to the native libraries

Provide ability to behave as an Objective-C object

Manage the messages and selectors

Register the class with the iOS runtime

What is NSObject?

`NSObject` is the root class of most class hierarchies in Objective-C

Inherits the ability to behave as an Objective-C object

Has all methods fundamental to an Objective-C object

Demo

Investigate NSObject used as a base class

Xamarin University

Xamarin.iOS NSObject

Xamarin.iOS provides a managed **`NSObject`** class that allows you to access unmanaged **`NSObject`** instances

The C# **`NSObject`** class is a managed representation of an underlying Objective-C instance

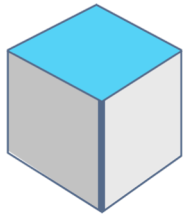
`NSObject.Handle` property provides a pointer to the unmanaged Objective-C object

Manage messages and selectors

`NSObject` provides the methods to manage messages and selectors

`RespondsToSelector`

`PerformSelector`



`GetMethodForSelector`

Access iOS objects in C#

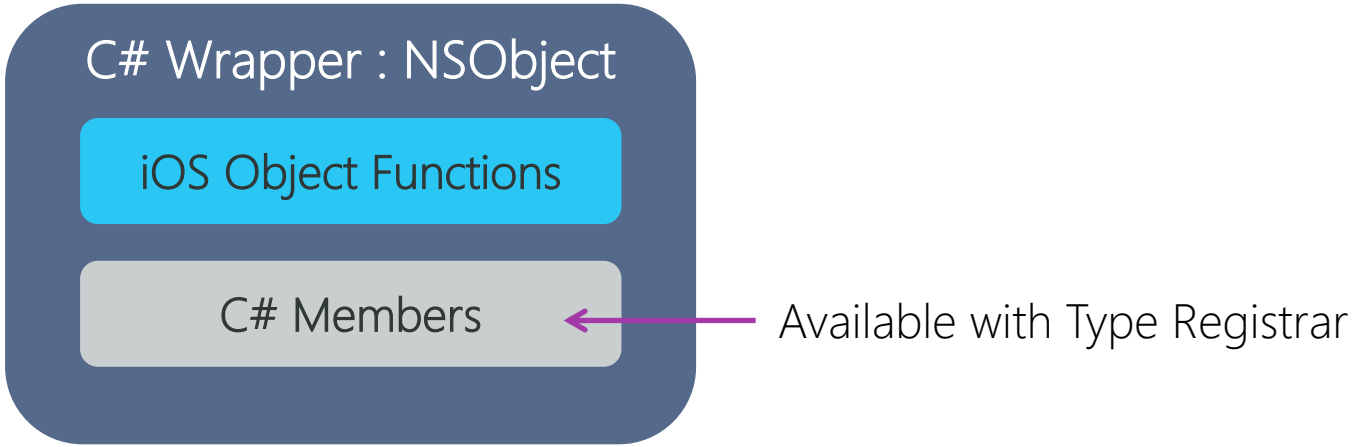
Derive your C# class from **`Foundation.NSObject`** to create a class that can be passed between your managed and unmanaged code


```
class MyBusinessLogic : NSObject
{
    ...
}
```


Classes derived from **`NSObject`** are prefixed with ***`__Namespace_*`***

Foundation.NSObject limitations

Xamarin.iOS uses a Type Registrar system to make all members of an **`NSObject`** derived class available to your unmanaged code



By default, only overridden methods are exposed to Objective-C

Demo

Investigate exposed class members

Xamarin University

Type registrar

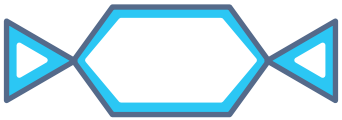
Xamarin.iOS implements a type registration system that registers attributed C# elements during application startup

Attribute	From C#	To Objective-C
[Register]	Class	Class
[Export]	Members	Members
[Protocol]	Interface	Protocols
[Category]	Extension Method	Category

What is the [Register] attribute?

The **[Register]** attribute is used to rename a class, identify a class as a wrapper and expose a class to the iOS Designer

Rename a class



Identify a wrapper class

Exposes class to the iOS Designer

Set the class name

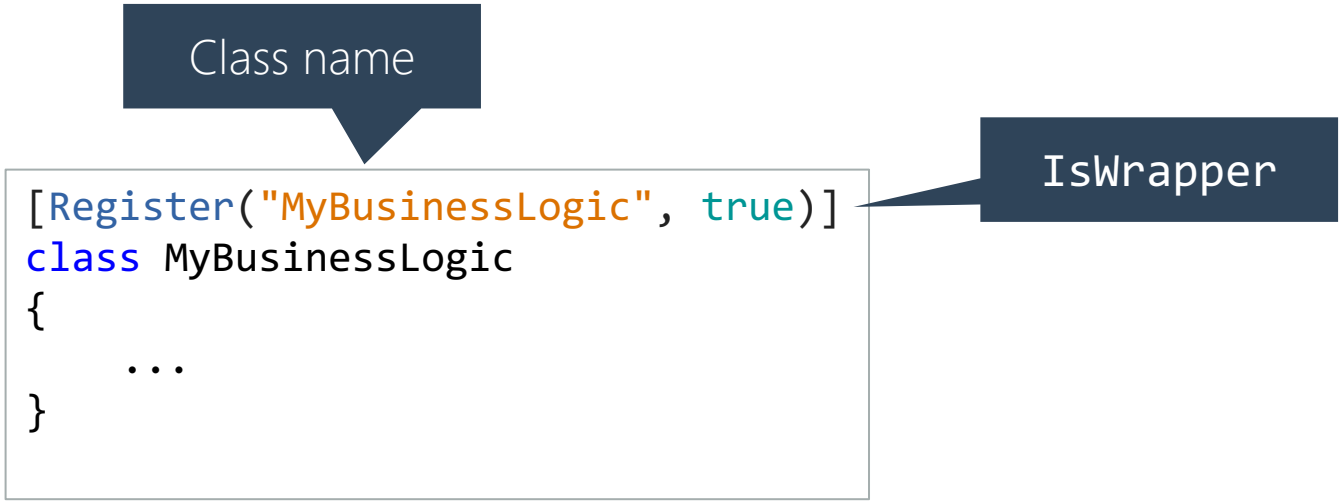
You can use the **[Register]** attribute to provide a different name for a C# class to the Objective-C runtime

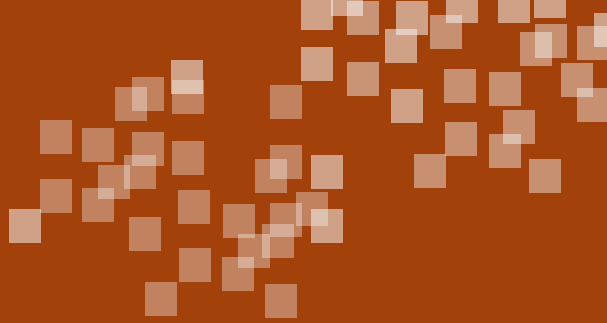
```
[Register("BizLogic")]
class MyBusinessLogic : NSObject
{

}
```

Class will be exposed to Objective-C runtime as "BizLogic"

Identify a wrapper class

The **[Register]** attribute has a Boolean property, **IsWrapper**, which allows you to identify your class as a wrapper





Demo

Implement [Register]

Register your class with the Designer

The **[Register]** attribute surfaces all of the members of a class, making them available to the iOS Designer

Accessible by unmanaged code

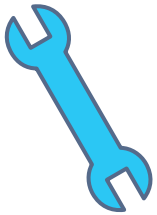
```
class BusinessLogic : NSObject
{
    ...
}
```

Accessible by unmanaged code and the iOS Designer

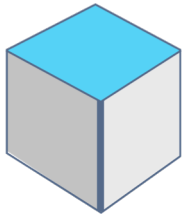
```
[Register ("MyBusinessLogic", true)]
class BusinessLogic : NSObject
{
    ...
}
```

[Export] attribute

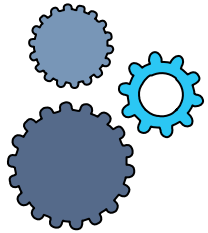
The **`[Export]`** attribute exposes C# class constructors, methods and properties to the Objective-C runtime



Constructors



Methods



Properties

[Export] constructors

The **[Export]** attribute allows you to expose a constructor of an **NSObject** derived class to the Objective-C runtime

```
public class Test : NSObject
{
    [Export("init")]
    public Weather() : base(NSObjectFlag.Empty)
    {
        ...
    }
}
```

Objective-C initializer

Demo

Implement [Export] of constructor

Xamarin University

[Export] methods

Use the **[Export]** attribute to expose methods of an **NSObject** derived class to the Objective-C runtime

```
public class Test : NSObject
{
    [Export ("setText:withFont)]
    public void SetText (string text, string font)
    {
        ...
    }
}
```

Objective-C compliant method name

[Export] properties

Use the **[Export]** attribute to expose the properties of an **NSObject** derived class to the Objective-C runtime

```
public class Test : NSObject
{
    [Export("active")]
    bool Active { get; set; }
}
```

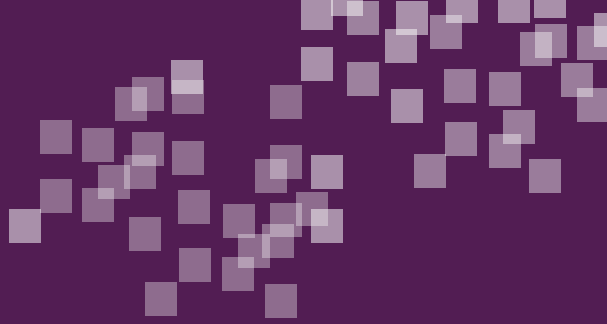
C# property

Get and set the Objective-C property

Demo

Implement [Export] of property

Xamarin University



Individual Exercise

Bind an Objective-C class

Xamarin University

Describe Frameworks & Libraries

Xamarin University

Tasks

1. Define a Bindings Library
2. Define a Framework
3. Investigate dynamic vs static libraries
4. Define a universal library
5. Use lipo to list library architectures
6. Use nm to display a symbol table

What is a Framework ?

A *Framework* is a collection of developer resources bundled into a single package

Binary code libraries

Header files

Designer files

Localized strings

Documentation

What is a Library?

A *Library* is the combination of the binary code and header files contained in a Framework

Binary code libraries

Header files

Designer files

Localized strings

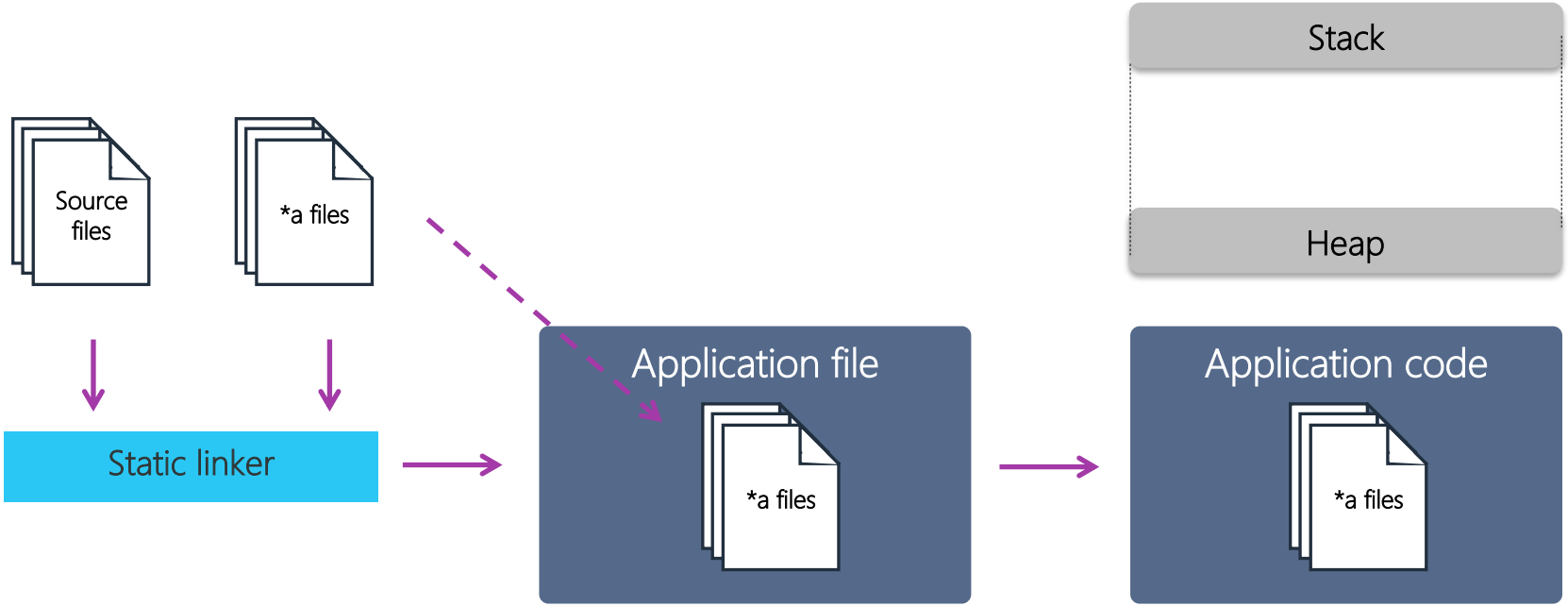
Documentation

Two ways to package your library

Frameworks and libraries may be included directly in your application or loaded at runtime

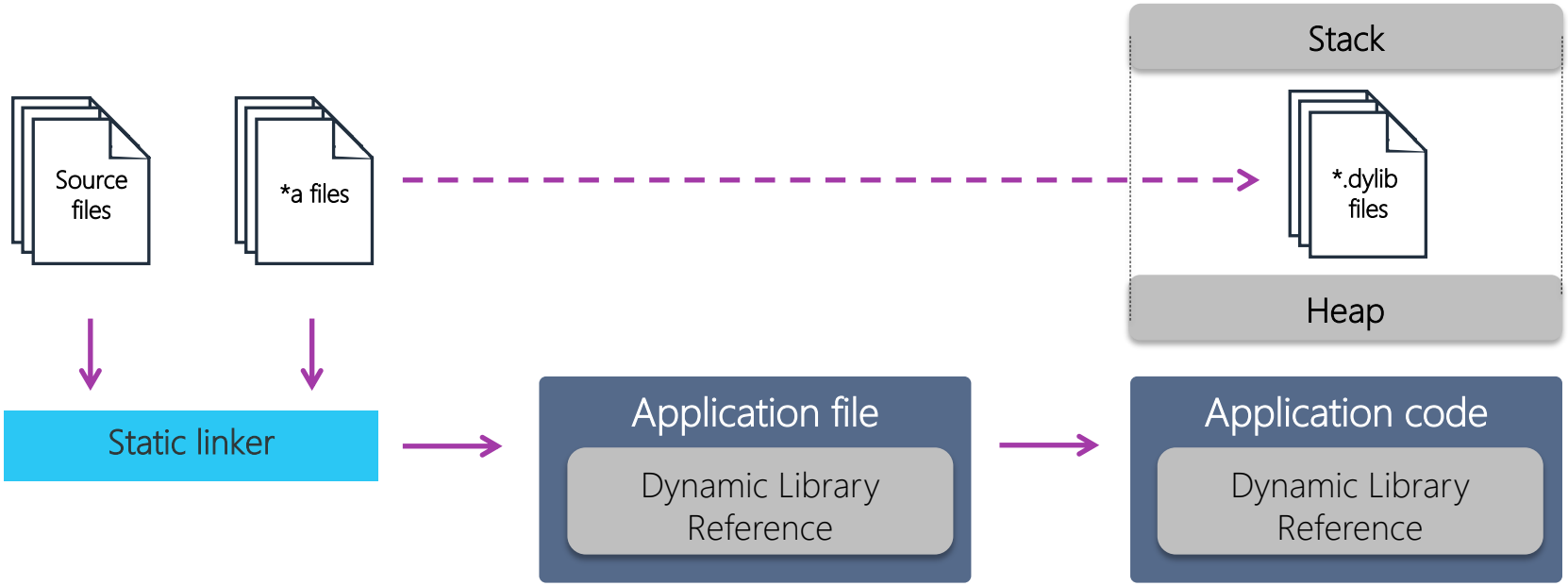
Static Libraries

A Static library is linked in the the target's binary



Dynamic libraries

Dynamic libraries created by Apple are loaded at runtime and may be updated independently of the target binary

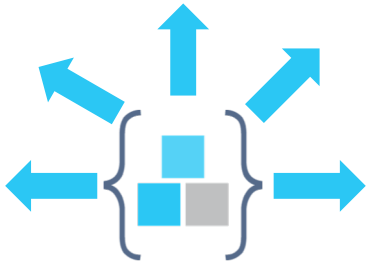


Why we use libraries

Code libraries make it convenient to package and reuse components

Encapsulate code

Code inheritance and modification



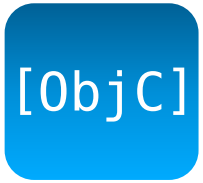
Distribute units of functionality

Files within a library are written in the same language

Library types

Libraries comes in different formats depending on the developers' choice of technology stack

Managed code

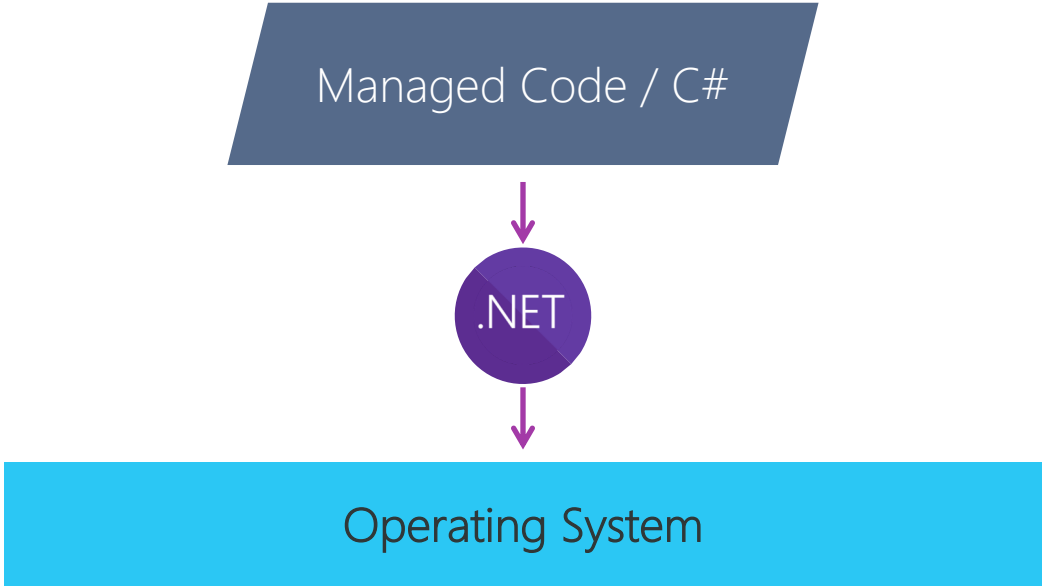


Unmanaged code

Native code

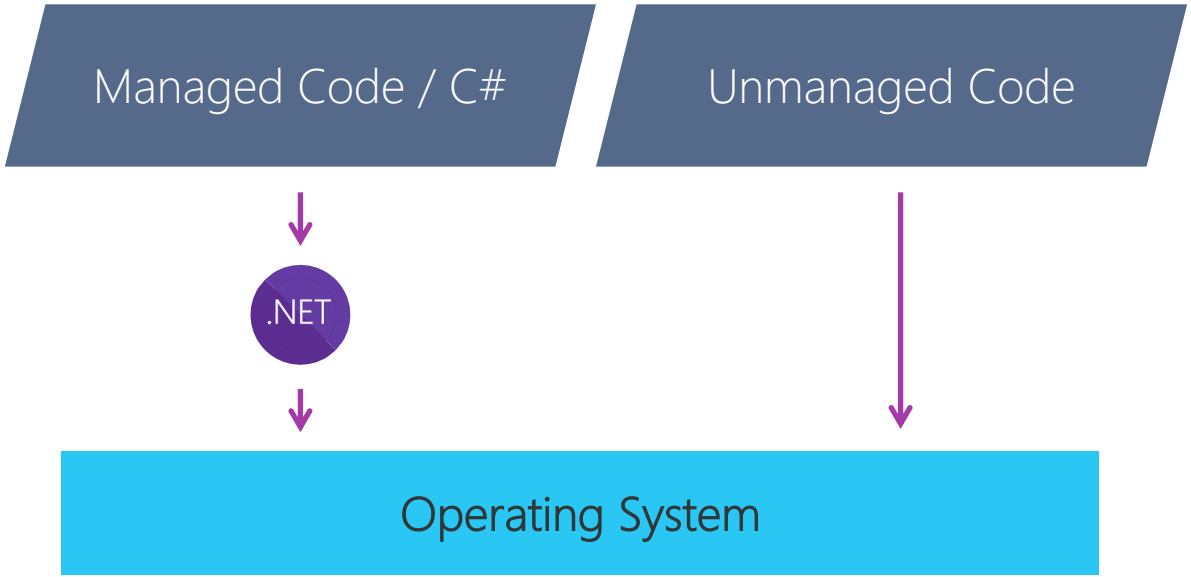
Managed code

Managed code is written to target the services of the managed runtime execution environment



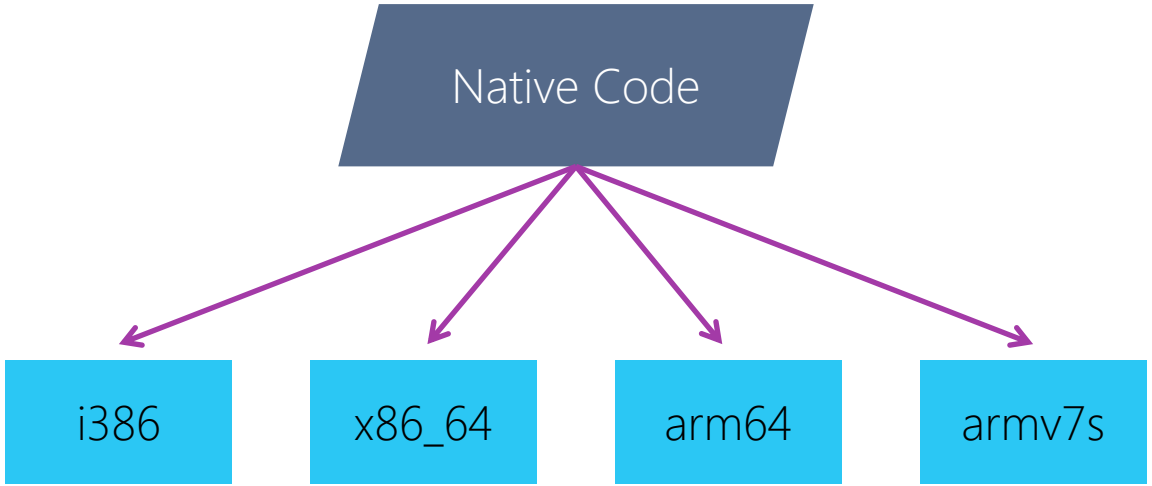
Unmanaged code

Unmanaged code compiles to the operating system, memory management and garbage collection are handled manually



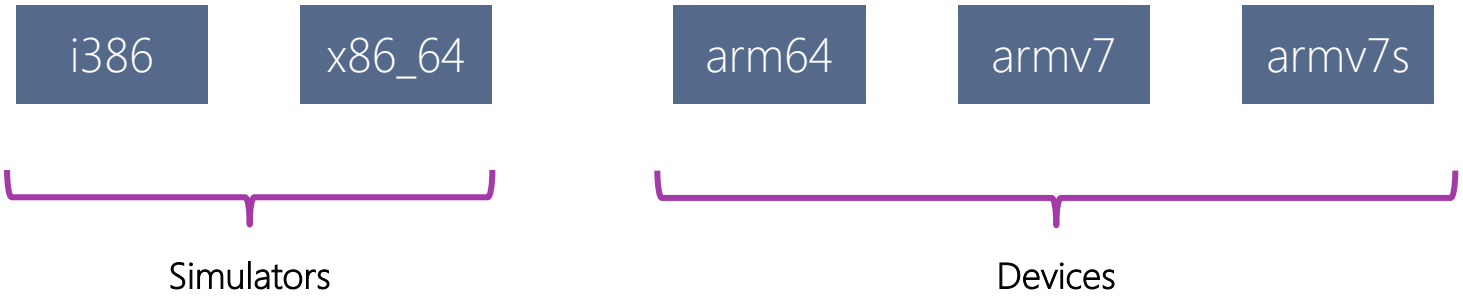
Native code

Native code is directly executed by the OS, and is compiled to target a specific architecture/platform



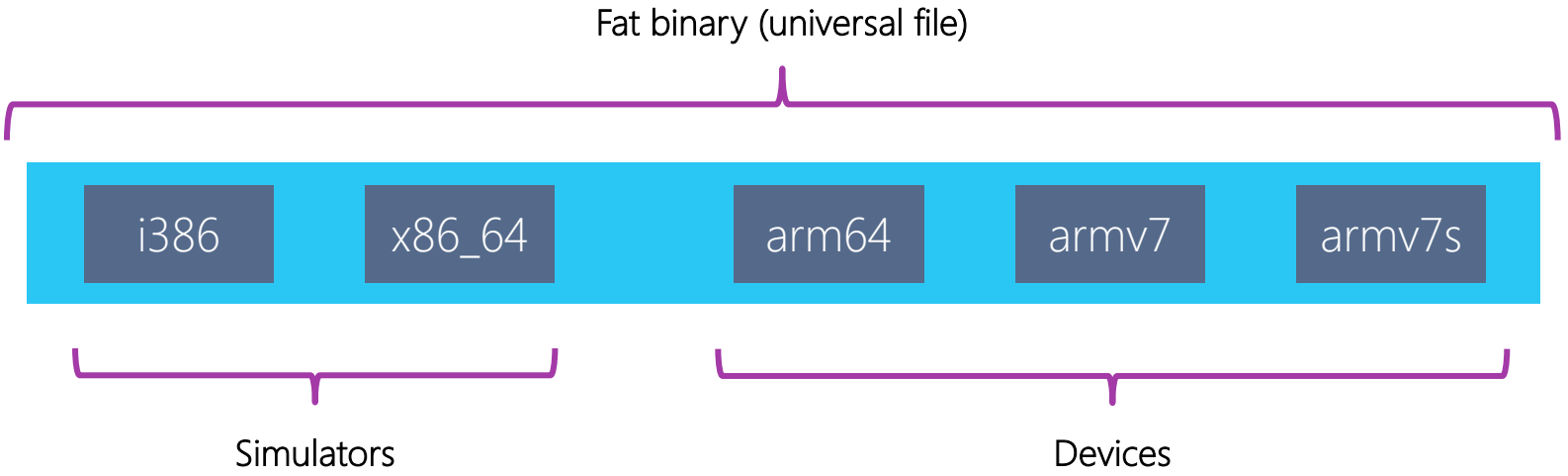
Binary Architectures

Binaries must be (re)compiled for the target architecture the library will be used with



Fat Binary

A *fat* or *universal binary* combines multiple architectures to simplify development



What is lipo?

Lipo is a command line utility you can access from Terminal use to manipulate universal binaries allowing uses to inspect and change the contents

Lists architecture types in a universal file

Creates a universal file

Extract, replace or remove architecture types from universal file

Demo

Investigate the implemented Library architecture

Xamarin University

Check the architecture(s) with lipo

Use lipo to check if the library you want to bind contains the architectures require (e.g. simulator and/or device)

```
$ lipo -info libWeatherGadget.a
input file libWeatherGadget.a is not a fat file
Non-fat file: libWeatherGadget.a is architecture: x86_64
```

```
$ lipo -info libWeatherGadget.a
Architectures in the fat file: libWeatherGadget.a  are: armv7 x86_64
arm64
```

Combine architectures with lipo

Lipo can be used to combine binaries from multiple architectures into a single fat binary

Take the input files (or file) and creates one universal output file from them

```
$ lipo -create -output fatfilepath/libWeatherGadget.a
armfileprath/libWeatherGadget.a simfilepath/libWeatherGadget.a
```

Specifies its argument as the output file

Fat binaries can be created via the Xcode project compilation process

Remove simulator binaries with lipo

Lipo can be used to remove binaries for specific architectures - it is recommended to remove simulator architectures from your libraries

Removes an architecture type from the universal file

```
$ lipo -remove i386 x86_64 -output libWeatherGadget.a
```


Important: This step is specific to libraries you are binding and not the application.

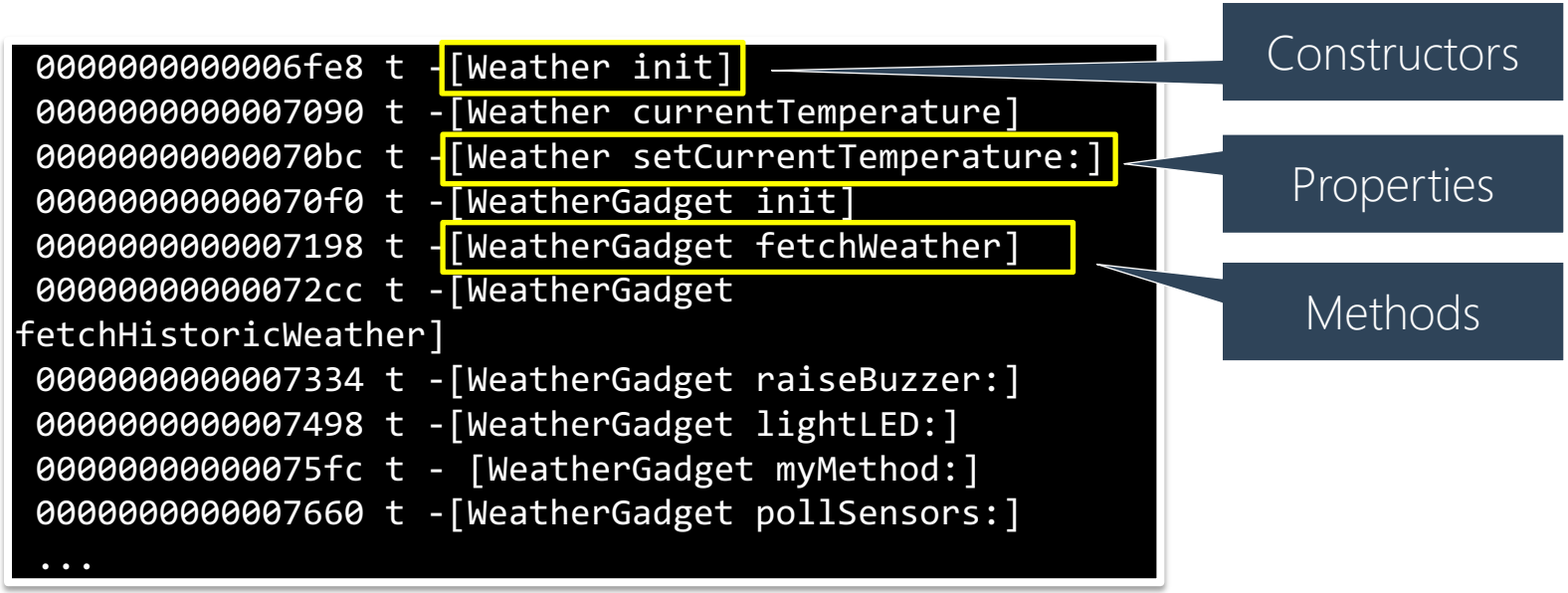
Demo

Inspect a static library with lipo

Xamarin University

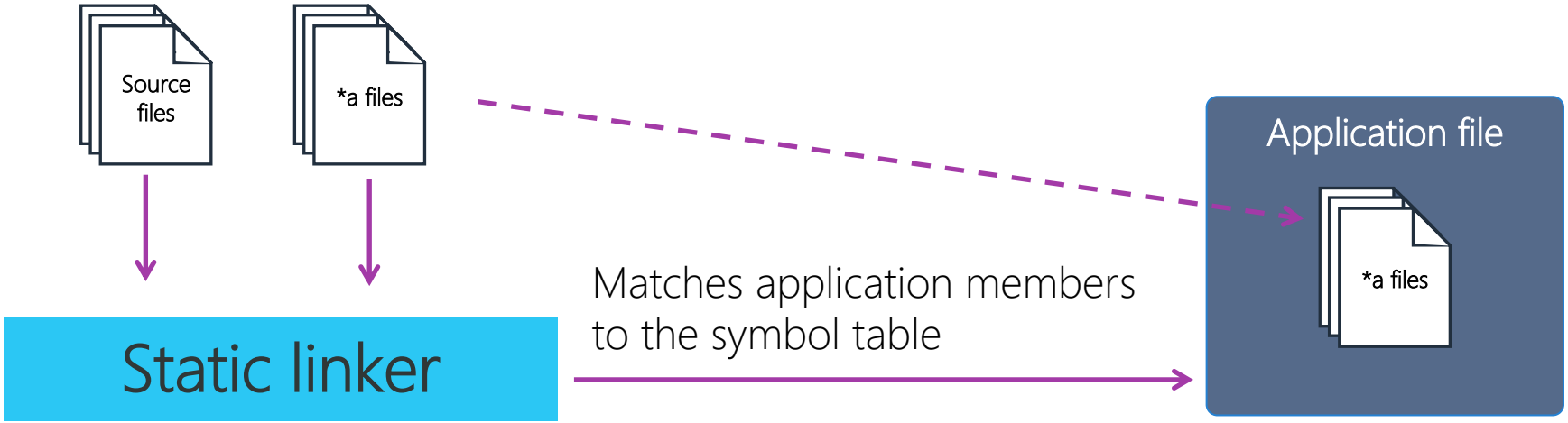
What is a symbol table?

A symbol table is a data structure used by the complier to associate attributes with identifiers used in an application



The linking process

At the time of linking the compiler will match the symbol table to the constructors, methods, properties, etc. used in your application



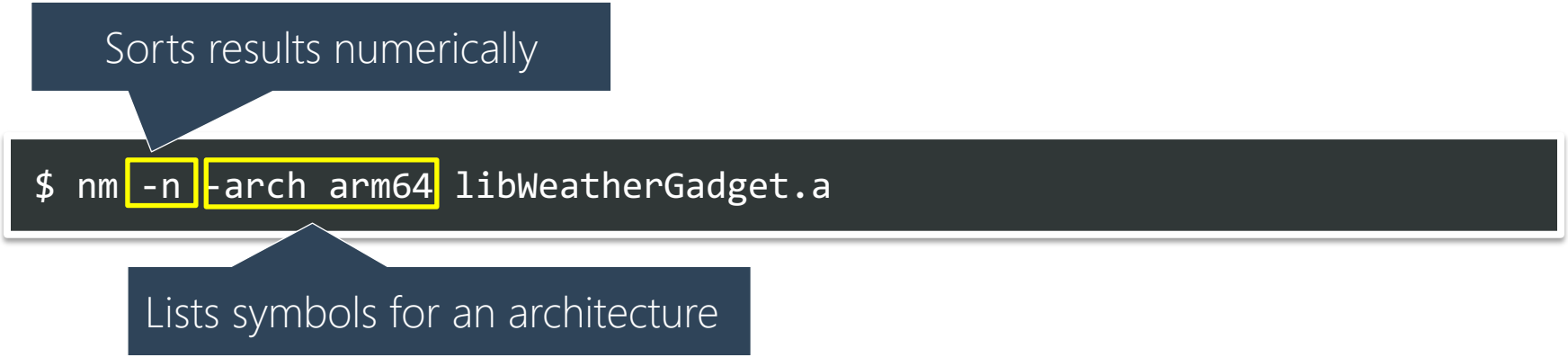
Missing classes

When a class or member is found to be missing during linking, an error will be thrown

MTOUCH : error MT5211: Native linking failed, undefined Objective-C class: Weather. The symbol '_OBJC_CLASS_$_Weather' could not be found in any of the libraries or frameworks linked with your application.

What is nm?

nm is a command line tool which displays the symbol table of an object file



Display the symbol table

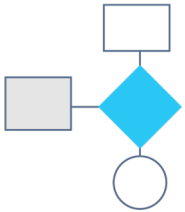
Displaying the symbol table with nm will allow you to quickly figure out what types and members are missing

```
0000000000006fe8 t -[Weather init]
0000000000007090 t -[Weather currentTemperature]
00000000000070bc t -[Weather setCurrentTemperature:]
00000000000070f0 t -[WeatherGadget init]
0000000000007198 t -[WeatherGadget fetchWeather]
00000000000072cc t -[WeatherGadget fetchHistoricWeather]
0000000000007334 t -[WeatherGadget raiseBuzzer:]
0000000000007498 t -[WeatherGadget lightLED:]
00000000000075fc t -[WeatherGadget myMethod:]
0000000000007660 t -[WeatherGadget pollSensors:]
...
```


Can combine the nm command with **grep** for e.g. **nm –n libXXX.so | grep " –t \["**

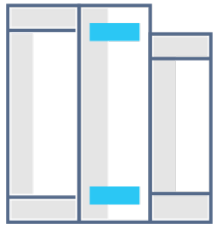
Missing symbols

If you find you're missing a symbol there are a few steps you can take to try and correct the issue



Check the architectures

Edit the library and recompile



Go back to the provider of the library

Demo

Inspect a static library with nm

Xamarin University

Summary

1. Bind an iOS library manually
2. Describe Frameworks & Libraries

Thank You!

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