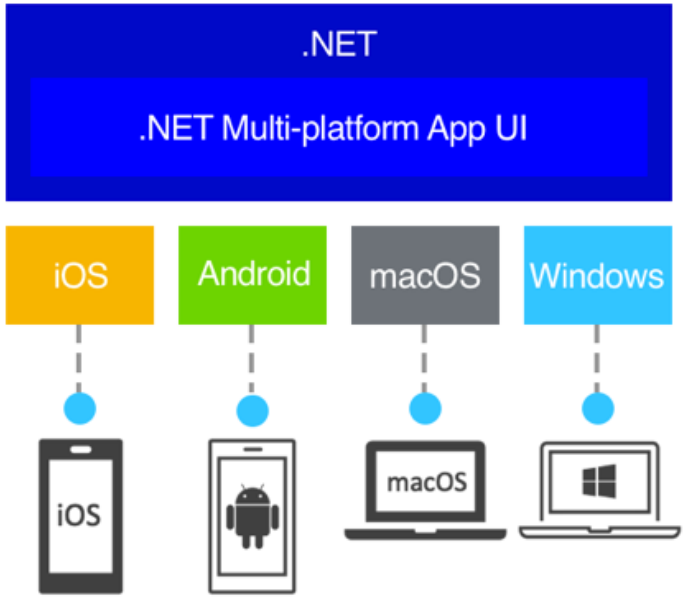
.NET runtime on mobile: Swift interop

Milos Kotlar

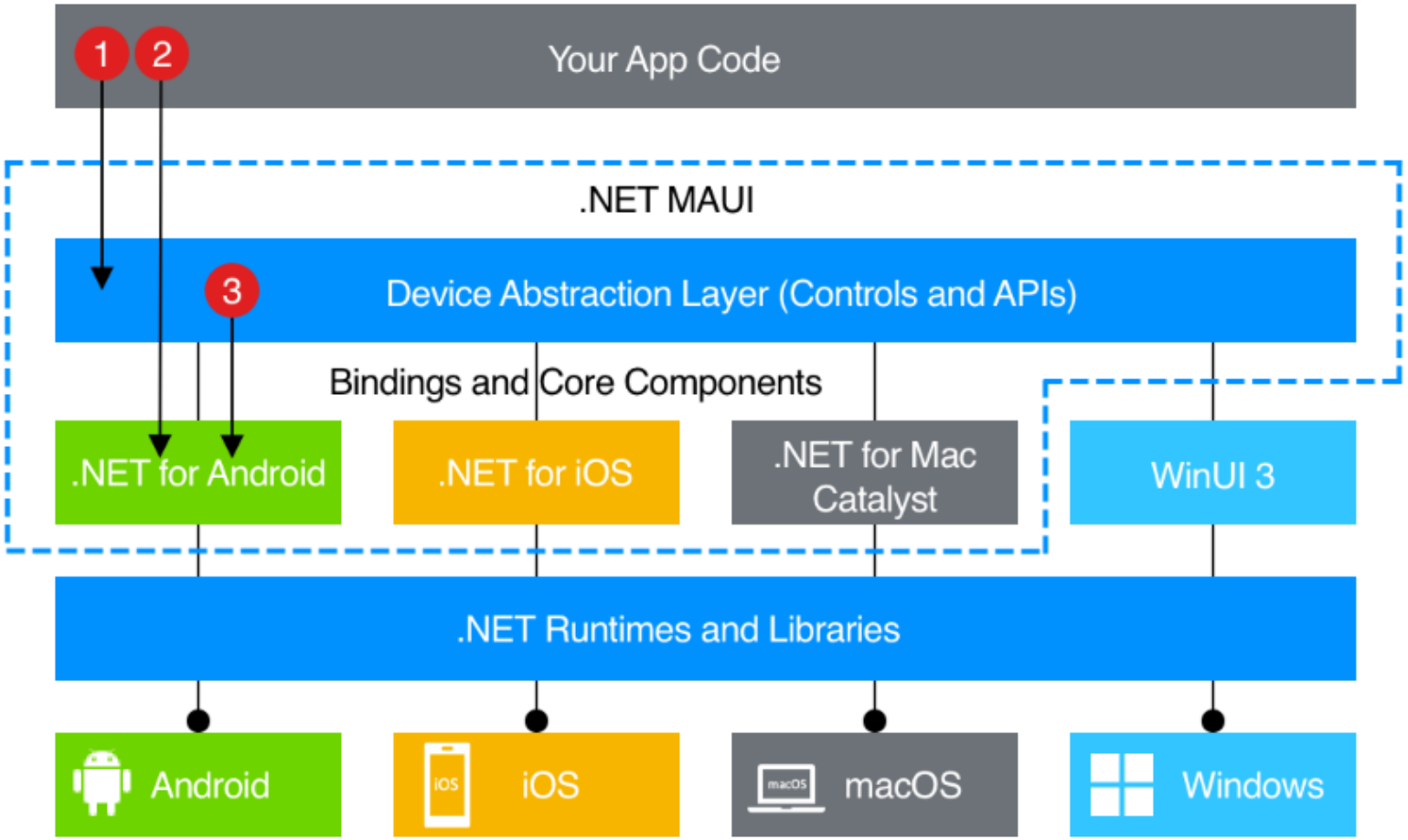
Software Engineer, Microsoft

.NET MAUI

- Build cross-platform apps for desktop and mobile platforms
- Single code-base
- Support for .NET runtime and libraries



.NET MAUI


1
2
Your App Code
.NET MAUI
3
Device Abstraction Layer (Controls and APIs)
Bindings and Core Components
.NET for Android
.NET for iOS
.NET for Mac Catalyst
WinUI 3
.NET Runtimes and Libraries
Android
iOS
macOS
Windows

.NET MAUI for Apple mobile

- MAUI provides support for Apple's SDK frameworks through Objective-C

- Newer frameworks are written in Swift, like:
  - SwiftUI
  - StoreKit
  - VisionKit
  - …

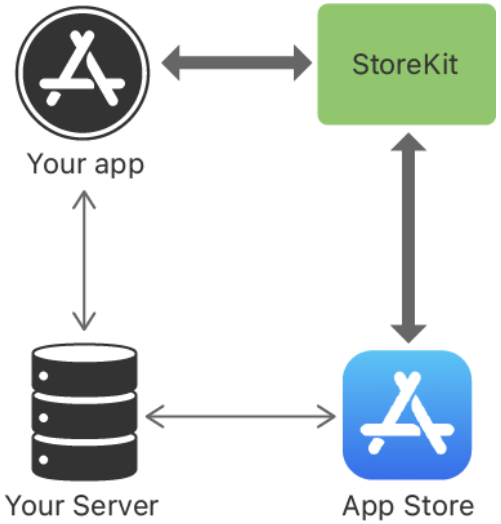
SwiftUI example

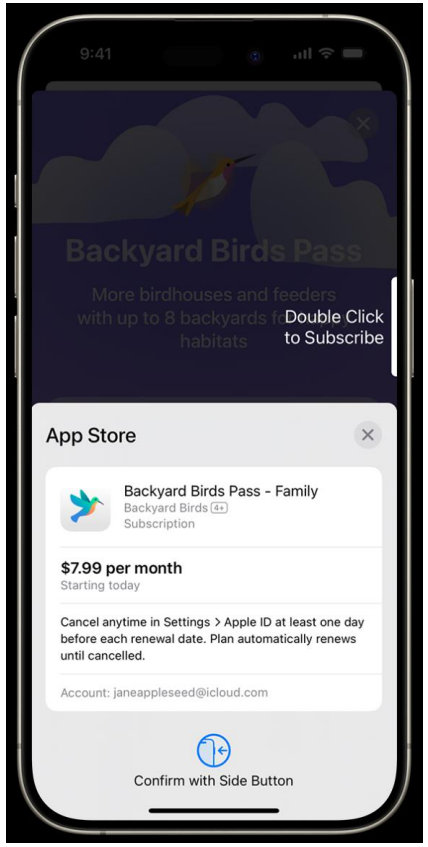
```swift
import SwiftUI

struct ContentView: View {
    @State private var isToggled = false

    var body: some View {
        VStack {
            Toggle("Toggle", isOn: $isToggled)
                .padding()
            if isToggled {
                Text("Toggle is ON")
                    .foregroundColor(.green)
            } else {
                Text("Toggle is OFF")
                    .foregroundColor(.red)
            }
        }
    }
}

struct ContentView_Previews: PreviewProvider {
    static var previews: some View {
        ContentView()
    }
}
```

```swift
import UIKit

class ViewController: UIViewController {
    let toggleSwitch = UISwitch()
    let label = UILabel()

    override func viewDidLoad() {
        super.viewDidLoad()

        toggleSwitch.addTarget(self, action: #selector(toggleSwitchChanged(_:))
        view.addSubview(toggleSwitch)

        label.textAlignment = .center
        view.addSubview(label)

        NSLayoutConstraint.activate([
            toggleSwitch.centerXAnchor.constraint(equalTo: view.centerXAnchor),
            toggleSwitch.centerYAnchor.constraint(equalTo: view.centerYAnchor),
            label.topAnchor.constraint(equalTo: toggleSwitch.bottomAnchor, cons
            label.leadingAnchor.constraint(equalTo: view.leadingAnchor),
            label.trailingAnchor.constraint(equalTo: view.trailingAnchor)
        ])
    }

    @objc func toggleSwitchChanged(_ sender: UISwitch) {
        if sender.isOn {
            label.text = "Toggle is ON"
            label.textColor = .green
        } else {
            label.text = "Toggle is OFF"
            label.textColor = .red
        }
    }
}
```

StoreKit example

```swift
func purchase(_ product: Product) async throws -> Transaction? {
    // Begin purchasing the `Product` the user selects.
    let result = try await product.purchase()

    switch result {
    case .success(let verification):
        // Check whether the transaction is verified. If it isn't,
        // this function rethrows the verification error.
        let transaction = try checkVerified(verification)

        // The transaction is verified. Deliver content to the user.
        await updateCustomerProductStatus()

        // Always finish a transaction.
        await transaction.finish()

        return transaction
    case .userCancelled, .pending:
        return nil
    default:
        return nil
    }
}
```





VisionKit example

```
31      func documentCameraViewController(_ controller: VNDocumentCameraViewController,
            didFinishWith scan: VNDocumentCameraScan) {
32          controller.dismiss(animated: true) { [weak self] in
33              self?.imageView.image = scan.imageOfPage(at: 0)
34
35              guard let strongSelf = self else { return }
36              UIAlertController.present(title: "Success!", message: "Document \(scan.title)
                    scanned with \(scan.pageCount) pages.", on: strongSelf)
37          }
38      }
39
40      func documentCameraViewControllerDidCancel(_ controller:
            VNDocumentCameraViewController) {
41          controller.dismiss(animated: true) { [weak self] in
42              self?.imageView.image = nil
43
44              guard let strongSelf = self else { return }
45              UIAlertController.present(title: "Cancelled", message: "User cancelled
                    operation.", on: strongSelf)
46          }
47      }
```

Swift vs Objective-C

- Objective-C
  - Dynamic typing
  - Message syntax `[myRectangle setWidth:20.0];`
  - Compatibility with C
  - Manual retain/release
- Swift
  - Static runtime with strong type system
  - Automatic reference counting
  - Optimized for speed

Goal: How to call into Swift SDK frameworks?

- Identify ABI differences between .NET and Swift
- Generate C# bindings that represent Swift types
- Deploy C# generated bindings as NuGet packages

```
using VisionKit;

ViewController barcodeScanner = DataScannerViewController();
barcodeScanner.StartScan();
barcodeScanner.SetScanCallbackWithCallback(result =>
{
    Console.WriteLine("Scanned items:");
    foreach (var item in result)
    {
        Console.WriteLine(item);
    }
});
```

ABI differences between .NET and Swift

- Swift has richer semantics than .NET and different callconv

```swift
protocol Container {
    associatedtype Item
    mutating func append(_ item: Item)
    var count: Int { get }
    subscript(i: Int) -> Item { get }
}
```

```swift
enum ArithmeticExpression {
    case number(Int)
    indirect case addition(ArithmeticExpression, ArithmeticExpression)
    indirect case multiplication(ArithmeticExpression, ArithmeticExpression)
}
```

```swift
struct IntStack: Container {
    // original IntStack implementation
    var items: [Int] = []
    mutating func push(_ item: Int) {
        items.append(item)
    }
    mutating func pop() -> Int {
        return items.removeLast()
    }
    // conformance to the Container protocol
    typealias Item = Int
    mutating func append(_ item: Int) {
        self.push(item)
    }
    var count: Int {
        return items.count
    }
    subscript(i: Int) -> Int {
        return items[i]
    }
}
```

```swift
func evaluate(_ expression: ArithmeticExpression) -> Int {
    switch expression {
    case let .number(value):
        return value
    case let .addition(left, right):
        return evaluate(left) + evaluate(right)
    case let .multiplication(left, right):
        return evaluate(left) * evaluate(right)
    }
}

print(evaluate(product))
// Prints "18"
```

Swift function signature physical lowering

- Swift ABI stability enables binary compatibility between applications and libraries compiled with different Swift versions
- Set of rules applied to the calling convention

```swift
@frozen public struct Point {
    let x: Double
    let y: Double
}
var myPoint = Point(x: 1.0, y: 2.0)
myPoint.x = 3.0 // This line results in a
```

```swift
struct MutableData {
    var values: [Int]
}
var dataA = MutableData(values: [1, 2, 3])
var dataB = dataA
dataB.values.append(4)
```

Swift function signature physical lowering

```
45  define protected swiftcc void @"output.FrozenPassThrough(vtype: output.Point) -> ()"(double %0, double %1) #0 !dbg !37 {
46  entry:
47    %vtype.debug = alloca %T6output5PointV, align 8
48    call void @llvm.dbg.declare(metadata ptr %vtype.debug, metadata !43, metadata !DIExpression()), !dbg !45
49    call void @llvm.memset.p0.i64(ptr align 8 %vtype.debug, i8 0, i64 16, i1 false)
50    call void @llvm.lifetime.start.p0(i64 16, ptr %vtype.debug), !dbg !46
51    %vtype.debug.x = getelementptr inbounds %T6output5PointV, ptr %vtype.debug, i32 0, i32 0, !dbg !48
52    %vtype.debug.x._value = getelementptr inbounds %TSd, ptr %vtype.debug.x, i32 0, i32 0, !dbg !48
53    store double %0, ptr %vtype.debug.x._value, align 8, !dbg !48
54    %vtype.debug.y = getelementptr inbounds %T6output5PointV, ptr %vtype.debug, i32 0, i32 1, !dbg !48
55    %vtype.debug.y._value = getelementptr inbounds %TSd, ptr %vtype.debug.y, i32 0, i32 0, !dbg !48
56    store double %1, ptr %vtype.debug.y._value, align 8, !dbg !48
57    ret void, !dbg !49
58  }
59
60  declare void @llvm.lifetime.start.p0(i64 immarg, ptr nocapture) #1
61
62  declare void @llvm.memset.p0.i64(ptr nocapture writeonly, i8, i64, i1 immarg) #2
63
64  declare void @llvm.dbg.declare(metadata, metadata, metadata) #3
65
66  define protected swiftcc void @"output.NonFrozenPassThrough(vtype: output.MutableData) -> ()"(ptr noalias nocapture dereferenceable(8) %0) #0 !dbg !50 {
67  entry:
68    %vtype.debug = alloca ptr, align 8
69    call void @llvm.dbg.declare(metadata ptr %vtype.debug, metadata !55, metadata !DIExpression()), !dbg !57
70    call void @llvm.memset.p0.i64(ptr align 8 %vtype.debug, i8 0, i64 8, i1 false)
71    %.values = getelementptr inbounds %T6output11MutableDataV, ptr %0, i32 0, i32 0, !dbg !58
72    %.values._buffer = getelementptr inbounds %TSa, ptr %.values, i32 0, i32 0, !dbg !58
73    %.values._buffer._storage = getelementptr inbounds %Ts22_ContiguousArrayBufferV, ptr %.values._buffer, i32 0, i32 0, !dbg !58
74    %1 = load ptr, ptr %.values._buffer._storage, align 8, !dbg !58
75    store ptr %1, ptr %vtype.debug, align 8, !dbg !60
76    ret void, !dbg !61
77  }
```

.NET function signature lowering for Swift

```
28
    Jakob Botsch Nielsen, 12 months ago | 1 author (Jakob Botsch Nielsen)
29  [StructLayout(LayoutKind.Sequential, Size = 8)]
    3 references
30  struct F0_S1
31  {
        1 reference
32      public ulong F0;
33
        1 reference
34      public F0_S1(ulong f0)
35      {
36          F0 = f0;
37      }
38  }
39
    Jakob Botsch Nielsen, 12 months ago | 1 author (Jakob Botsch Nielsen)
40  [StructLayout(LayoutKind.Sequential, Size = 4)]
    3 references
41  struct F0_S2
42  {
        1 reference
43      public float F0;
44
        1 reference
45      public F0_S2(float f0)
46      {
47          F0 = f0;
48      }
49  }
50
51  [UnmanagedCallConv(CallConvs = new Type[] { typeof(CallConvSwift) })]
52  [DllImport(SwiftLib, EntryPoint = "$s14SwiftAbiStress10swiftFunc02a02a12a22a32a42a52a62a7Sis5Int16V_s5Int32Vs6UInt64V
    1 reference
53  private static extern nint SwiftFunc0(short a0, int a1, ulong a2, ushort a3, F0_S0 a4, F0_S1 a5, byte a6, F0_S2 a7);
```

```
213     public static CORINFO_SWIFT_LOWERING LowerTypeForSwiftSignature(TypeDesc type)
214     {
215         if (!type.IsValueType || type is DefType { ContainsGCPointers: true })
216         {
217             Debug.Fail("Non-unmanaged types should not be passed directly to a Swift function.");
218             return new() { byReference = true };
219         }
220
221         LoweringVisitor visitor = new(type.Context.Target.PointerSize);
222         visitor.AddFields(type, addTrailingEmptyInterval: false);
223
224         List<(CorInfoType type, int offset)> loweredTypes = visitor.GetLoweredTypeSequence();
225
226         // If a type has a primitive sequence with more than 4 elements, Swift passes it by reference.
227         if (loweredTypes.Count > 4)
228         {
229             return new() { byReference = true };
230         }
231
232         CORINFO_SWIFT_LOWERING lowering = new()
233         {
234             byReference = false,
235             numLoweredElements = loweredTypes.Count
236         };
237
238         for (int i = 0; i < loweredTypes.Count; i++)
239         {
240             lowering.LoweredElements[i] = loweredTypes[i].type;
241             lowering.Offsets[i] = (uint)loweredTypes[i].offset;
242         }
243
244         return lowering;
245     }
```

Swift calling convention

ARM64

See Apple ARM64 Documentation, Procedure Call Standard for the Arm 64-bit Architecture.

Register usage

Register	Special	Purpose	C++	ObjC	Swift
`x0`		Integer argument 1 (1st return value)	`this`	`self`	
`x1`		Integer argument 2 (2nd return value)		`_cmd`	
`x2` - `x7`		Integer arguments 3-8 (3rd-8th return values)			
`x8`		Indirect result location register			
`x16`, `x17`	`ip0`, `ip1`	Scratch registers (used by dyld, can be used freely otherwise)			
`x18`		RESERVED **DO NOT USE**			
`x19`		Callee-saved register			
`x20`		Callee-saved register			`self`
`x21`		Callee-saved register			Error return
`x22`		Callee-saved register			Async context
`x23` - `x28`		Callee-saved registers			
`x29`	`fp`	Frame pointer			
`x30`	`lr`	Link register			
`sp`		Stack pointer			
`v0` - `v7`		Floating point/SIMD arguments 1-8 (also for return)			
`v8` - `v15`		Callee-saved registers (**lower 64-bits only**)			

System.Runtime.InteropServices.Swift Namespace

Reference

Feedback

Support types for .NET interop with Swift code.

Structs

Expand table

SwiftError	Represents the Swift error context, indicating that the argument is the error context.
SwiftIndirectResult	Represents the Swift return buffer context.
SwiftSelf	Represents the Swift 'self' context, indicating that the argument is the self context.
SwiftSelf<T>	Represents the Swift 'self' context when the argument is Swift frozen struct T, which is either enregistered into multiple registers, or passed by reference in the 'self' register.

Remarks

These types are used by low level interop and code-generation tools to enable interop between .NET code and Swift code.

Swift calling convention

```
[Fact]
0 references
public unsafe static void TestSwiftSelfContext()
{
    void* pointer = getInstance();
    SwiftSelf self = new SwiftSelf(pointer);
    Assert.True(self.Value != null, "Failed to obtain an instance of SwiftSelf from the Swift library.");

    int result = (int)getMagicNumber(self);
    Assert.True(result == 42, "The result from Swift does not match the expected value.");
}
```

```
66      [Fact]
        0 references
67      public unsafe static void TestSwiftErrorThrown()
68      {
69          const string expectedErrorMessage = "Catch me if you can!";
70          SetErrorMessageForSwift(expectedErrorMessage);
71
72          SwiftError error = new SwiftError();
73
74          // This will throw an error
75          conditionallyThrowError(1, ref error);
76          Assert.True(error.Value != null, "A Swift error was expected to be thrown.");
77
78          string errorMessage = GetErrorMessageFromSwift(error);
79          Assert.True(errorMessage == expectedErrorMessage, string.Format("The error message retrieved from Swift
80      }
```

Swift memory management

- GC vs ARC
- Swift types have ValueWitnessTable with InitWithCopy and Destroy functions
- Swift value types with reference properties are projected as C# classes with finalizer

```
114    ///   void (*destroy)(T *object, witness_t *self);
115    ///
116    /// Given a valid object of this type, destroy it, leaving it as an
117    /// invalid object.  This is useful when generically destroying
118    /// an object which has been allocated in-line, such as an array,
119    /// struct, or tuple element.
120    FUNCTION_VALUE_WITNESS(destroy,
121                           Destroy,
122                           VOID_TYPE,
123                           (MUTABLE_VALUE_TYPE, TYPE_TYPE))
124
125    ///   T *(*initializeWithCopy)(T *dest, T *src, M *self);
126    ///
127    /// Given an invalid object of this type, initialize it as a copy of
128    /// the source object.  Returns the dest object.
129    FUNCTION_VALUE_WITNESS(initializeWithCopy,
130                           InitializeWithCopy,
131                           MUTABLE_VALUE_TYPE,
132                           (MUTABLE_VALUE_TYPE, MUTABLE_VALUE_TYPE, TYPE_TYPE))
```

```
50      protected virtual void Dispose(bool disposing)
51      {
52          if (Interlocked.CompareExchange(ref _disposed, 1, 0) == 0)
53          {
54              var metadata = SwiftObjectHelper<SwiftArray<Element>>.GetTypeMetadata();
55
56              unsafe
57              {
58                  fixed (void* payload = &_buffer)
59                  {
60                      metadata.ValueWitnessTable->Destroy(payload, metadata);
61                  }
62              }
63              _disposed = 1;
64          }
65      }
```

.NET interop with Swift - Challenges

- Generics example

```swift
public
struct MutableData {
    var values: [Int]
}
@frozen
public struct Point {
    let x: Double
    let y: Double
}

@frozen
public struct GenericStruct<T> {
    let x: T
    let y: Double
}
```

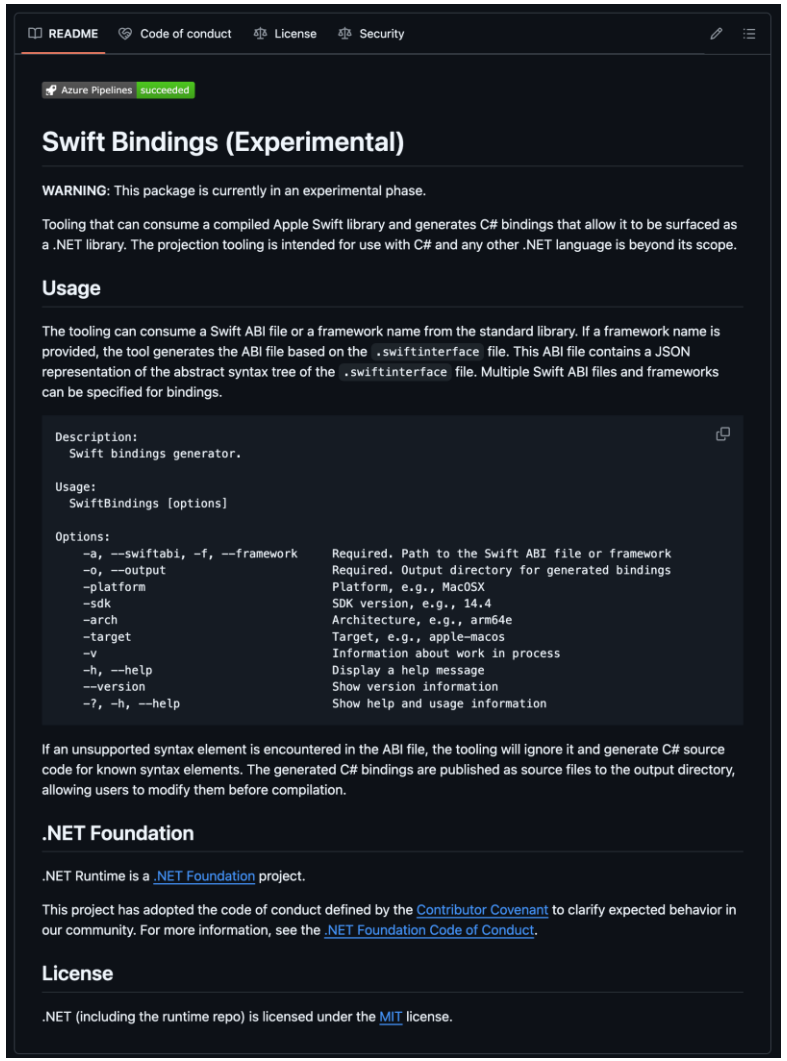
```llvm
define protected swiftcc void @"output.FrozenPassThrough(vtype: output.GenericStruct<output.Point>) -> ()"(double %0, double %1, double %2) #0 !dbg !37 {
entry:
  %vtype.debug = alloca %T6output13GenericStructVyAA5PointVG, align 8
  call void @llvm.dbg.declare(metadata ptr %vtype.debug, metadata !43, metadata !DIExpression()), !dbg !52
  call void @llvm.memset.p0.i64(ptr align 8 %vtype.debug, i8 0, i64 24, i1 false)
  call void @llvm.lifetime.start.p0(i64 24, ptr %vtype.debug), !dbg !53
  %vtype.debug.x = getelementptr inbounds %T6output13GenericStructVyAA5PointVG, ptr %vtype.debug, i32 0, i32 0, !dbg !55
  %vtype.debug.x.x = getelementptr inbounds %T6output5PointV, ptr %vtype.debug.x, i32 0, i32 0, !dbg !55
  %vtype.debug.x.x._value = getelementptr inbounds %TSd, ptr %vtype.debug.x.x, i32 0, i32 0, !dbg !55
  store double %0, ptr %vtype.debug.x.x._value, align 8, !dbg !55
  %vtype.debug.x.y = getelementptr inbounds %T6output5PointV, ptr %vtype.debug.x, i32 0, i32 1, !dbg !55
  %vtype.debug.x.y._value = getelementptr inbounds %TSd, ptr %vtype.debug.x.y, i32 0, i32 0, !dbg !55
  store double %1, ptr %vtype.debug.x.y._value, align 8, !dbg !55
  %vtype.debug.y = getelementptr inbounds %T6output13GenericStructVyAA5PointVG, ptr %vtype.debug, i32 0, i32 1, !dbg !55
  %vtype.debug.y._value = getelementptr inbounds %TSd, ptr %vtype.debug.y, i32 0, i32 0, !dbg !55
  store double %2, ptr %vtype.debug.y._value, align 8, !dbg !55
  ret void, !dbg !56
}

declare void @llvm.lifetime.start.p0(i64 immarg, ptr nocapture) #1

declare void @llvm.memset.p0.i64(ptr nocapture writeonly, i8, i64, i1 immarg) #2

declare void @llvm.dbg.declare(metadata, metadata, metadata) #3

define protected swiftcc void @"output.NonFrozenPassThrough(vtype: output.GenericStruct<output.MutableData>) -> ()"(ptr noalias nocapture dereferenceable(16) %0) #0 !dbg !57 {
entry:
  %vtype.debug = alloca %T6output13GenericStructVyAA11MutableDataVG, align 8
  call void @llvm.dbg.declare(metadata ptr %vtype.debug, metadata !62, metadata !DIExpression()), !dbg !71
  call void @llvm.memset.p0.i64(ptr align 8 %vtype.debug, i8 0, i64 16, i1 false)
  %.x = getelementptr inbounds %T6output13GenericStructVyAA11MutableDataVG, ptr %0, i32 0, i32 0, !dbg !72
  %.x.values = getelementptr inbounds %T6output11MutableDataV, ptr %.x, i32 0, i32 0, !dbg !72
  %.x.values._buffer = getelementptr inbounds %TSa, ptr %.x.values, i32 0, i32 0, !dbg !72
  %.x.values._buffer._storage = getelementptr inbounds %Ts22_ContiguousArrayBufferV, ptr %.x.values._buffer, i32 0, i32 0, !dbg !72
  %1 = load ptr, ptr %.x.values._buffer._storage, align 8, !dbg !72
  %.y = getelementptr inbounds %T6output13GenericStructVyAA11MutableDataVG, ptr %0, i32 0, i32 1, !dbg !72
  %.y._value = getelementptr inbounds %TSd, ptr %.y, i32 0, i32 0, !dbg !72
  %2 = load double, ptr %.y._value, align 8, !dbg !72
  call void @llvm.lifetime.start.p0(i64 16, ptr %vtype.debug), !dbg !72
  %vtype.debug.x = getelementptr inbounds %T6output13GenericStructVyAA11MutableDataVG, ptr %vtype.debug, i32 0, i32 0, !dbg !74
  %vtype.debug.x.values = getelementptr inbounds %T6output11MutableDataV, ptr %vtype.debug.x, i32 0, i32 0, !dbg !74
  %vtype.debug.x.values._buffer = getelementptr inbounds %TSa, ptr %vtype.debug.x.values, i32 0, i32 0, !dbg !74
  %vtype.debug.x.values._buffer._storage = getelementptr inbounds %Ts22_ContiguousArrayBufferV, ptr %vtype.debug.x.values._buffer, i32 0, i32 0, !dbg !74
  store ptr %1, ptr %vtype.debug.x.values._buffer._storage, align 8, !dbg !74
  %vtype.debug.y = getelementptr inbounds %T6output13GenericStructVyAA11MutableDataVG, ptr %vtype.debug, i32 0, i32 1, !dbg !74
  %vtype.debug.y._value = getelementptr inbounds %TSd, ptr %vtype.debug.y, i32 0, i32 0, !dbg !74
  store double %2, ptr %vtype.debug.y._value, align 8, !dbg !74
  ret void, !dbg !75
}
```

How to generate C# bindings?

- Create projection tooling that inputs Swift dynamic library and produces C# code

- How to contribute?

https://github.com/dotnet/runtimelab/tree/feature/swift-bindings



README Code of conduct License Security

Azure Pipelines succeeded

Swift Bindings (Experimental)

WARNING: This package is currently in an experimental phase.

Tooling that can consume a compiled Apple Swift library and generates C# bindings that allow it to be surfaced as a .NET library. The projection tooling is intended for use with C# and any other .NET language is beyond its scope.

Usage

The tooling can consume a Swift ABI file or a framework name from the standard library. If a framework name is provided, the tool generates the ABI file based on the `.swiftinterface` file. This ABI file contains a JSON representation of the abstract syntax tree of the `.swiftinterface` file. Multiple Swift ABI files and frameworks can be specified for bindings.

```
Description:
  Swift bindings generator.

Usage:
  SwiftBindings [options]

Options:
    -a, --swiftabi, -f, --framework     Required. Path to the Swift ABI file or framework
    -o, --output                        Required. Output directory for generated bindings
    -platform                           Platform, e.g., MacOSX
    -sdk                                SDK version, e.g., 14.4
    -arch                               Architecture, e.g., arm64e
    -target                             Target, e.g., apple-macos
    -v                                  Information about work in process
    -h, --help                          Display a help message
    --version                           Show version information
    -?, -h, --help                      Show help and usage information
```

If an unsupported syntax element is encountered in the ABI file, the tooling will ignore it and generate C# source code for known syntax elements. The generated C# bindings are published as source files to the output directory, allowing users to modify them before compilation.

.NET Foundation

.NET Runtime is a .NET Foundation project.

This project has adopted the code of conduct defined by the Contributor Covenant to clarify expected behavior in our community. For more information, see the .NET Foundation Code of Conduct.

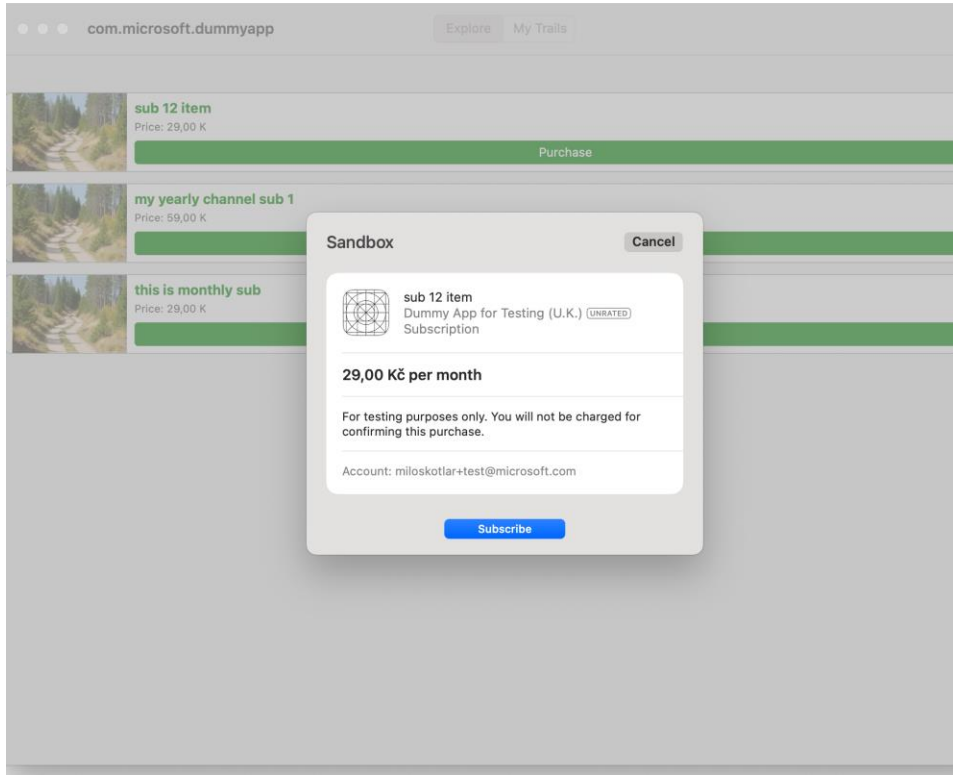
License

.NET (including the runtime repo) is licensed under the MIT license.

StoreKit bindings – InApp purchase

```
1 reference
public static async Task<IEnumerable<Trail>> GetStoreKitTrailData()
{
    var productsTask = Product.products(["sub12", "123456", "yearlysub"]);
    await productsTask;
    SwiftArray<Product> products = productsTask.Result;

    List<Trail> trails = new List<Trail>();
    Random random = Random.Shared;

    for (int i = 0; i < products.Count; ++i)
    {
        Product product = products[i];

        Console.WriteLine("Product retrieved");
        string trailName = product.DisplayName;
        string description = $"Price: {product.DisplayPrice}";
        Location startLocation = new Location(
            50.08804 + random.NextDouble() * 0.1,
            14.42076 + random.NextDouble() * 0.1
        );
        double distance = Math.Round(1 + random.NextDouble() * 20, 1);
        string difficulty = string.Empty;
        string terrain = string.Empty;

        trails.Add(new Trail(trailName, description, startLocation, distance, difficulty, terrain, product));
    }

    return trails;
}
```

```
// Add tap gesture recognizer to the purchase button
purchaseButton.TouchUpInside += async (sender, e) =>
{
    var purchaseTask = trailViewController.ProductItem.purchase();
    await purchaseTask;
};
stackView.AddArrangedSubview(trailView);
```



StoreKit bindings

StoreKit / Product / products(for:)

Type Method

products(for:)

Requests product data from the App Store.

iOS 15.0+ | iPadOS 15.0+ | macOS 12.0+ | tvOS 15.0+ | visionOS 1.0+ | watchOS 8.0+

```swift
static func products<Identifiers>(for identifiers: Identifiers) async throws -> [Product] w
```

```
1 reference
public static unsafe Task<SwiftArray<Product>> products(string [] identifiers)
{
    TaskCompletionSource<SwiftArray<Product>> task = new TaskCompletionSource<SwiftArray<Product>>();
    GCHandle handle = GCHandle.Alloc(task, GCHandleType.Normal);
    try
    {
        SwiftArray<SwiftString> swiftIdentifiers = new SwiftArray<SwiftString>();
        for (int i = 0; i < identifiers.Length; i++)
        {
            var str = new SwiftString(identifiers[i]);
            swiftIdentifiers.Append(str);
        }
        PInvoke_products((IntPtr)s_productsCallback, IntPtr.Zero, GCHandle.ToIntPtr(handle), swiftIdentifiers.Payload);

        return task.Task;
    }
    finally
    {
    }
}

1 reference
private static unsafe delegate* unmanaged[Swift]<ArrayBuffer, IntPtr, void> s_productsCallback = &productsOnComplete;
[UnmanagedCallersOnly(CallConvs = new[] { typeof(CallConvSwift) })]
1 reference
private static void productsOnComplete(ArrayBuffer result, IntPtr task)
{
    GCHandle handle = GCHandle.FromIntPtr(task);
    try
    {
        if (handle.Target is TaskCompletionSource<SwiftArray<Product>> tcs)
        {
            var taskResult = SwiftMarshal.MarshalFromSwift<SwiftArray<Product>>((SwiftHandle)new IntPtr(&result));
            tcs.TrySetResult(taskResult);
        }
    }
    finally
    {
        handle.Free();
    }
}

[UnmanagedCallConv(CallConvs = new Type[] { typeof(CallConvSwift) })]
[DllImport("__Internal", EntryPoint = "products_async")]
1 reference
internal static extern void PInvoke_products(IntPtr callback, IntPtr context, IntPtr task, ArrayBuffer identifiers);
```

StoreKit bindings

StoreKit / Product / purchase(options:)

Instance Method

purchase(options:)

Initiates a purchase for the product with the App Store and displays the confirmation sheet.

iOS 15.0+ | iPadOS 15.0+ | macOS 12.0+ | tvOS 15.0+ | watchOS 8.0+

```
@MainActor
func purchase(options: Set<Product.Purchase
Option> = []) async throws -> Product.PurchaseResult
```

```
1 reference
public unsafe Task purchase()
{
    TaskCompletionSource task = new TaskCompletionSource();
    GCHandle handle = GCHandle.Alloc(task, GCHandleType.Normal);

    PInvoke_productsPurchase((IntPtr)s_productsPurchaseCallback, IntPtr.Zero, GCHandle.ToIntPtr(handle), new SwiftSelf((void*)_payload));
    return task.Task;
}

1 reference
private static unsafe delegate* unmanaged[Cdecl]<IntPtr, void> s_productsPurchaseCallback = &productsPurchaseOnComplete;
[UnmanagedCallersOnly(CallConvs = new[] { typeof(CallConvCdecl) })]
1 reference
private static void productsPurchaseOnComplete(IntPtr task)
{
    GCHandle handle = GCHandle.FromIntPtr(task);
    try
    {
        if (handle.Target is TaskCompletionSource tcs)
        {
            tcs.TrySetResult();
        }
    }
    finally
    {
        handle.Free();
    }
}

[UnmanagedCallConv(CallConvs = new Type[] { typeof(CallConvSwift) })]
[DllImport("__Internal", EntryPoint = "purchase_async")]
1 reference
internal static extern void PInvoke_productsPurchase(IntPtr callback, IntPtr context, IntPtr task, SwiftSelf payload);

1 reference
public static unsafe Task<SwiftArray<Product>> products(string [] identifiers)
{
    TaskCompletionSource<SwiftArray<Product>> task = new TaskCompletionSource<SwiftArray<Product>>();
    GCHandle handle = GCHandle.Alloc(task, GCHandleType.Normal);
    try
    {
        SwiftArray<SwiftString> swiftIdentifiers = new SwiftArray<SwiftString>();
        for (int i = 0; i < identifiers.Length; i++)
        {
            var str = new SwiftString(identifiers[i]);
            swiftIdentifiers.Append(str);
        }
        PInvoke_products((IntPtr)s_productsCallback, IntPtr.Zero, GCHandle.ToIntPtr(handle), swiftIdentifiers.Payload);

        return task.Task;
    }
    finally
    {
    }
}
```