Going 100%*
Compose Multiplatform

by Richard Schattauer

* and all the things they didn’t tell you

Compose Multiplatform 1.6.11

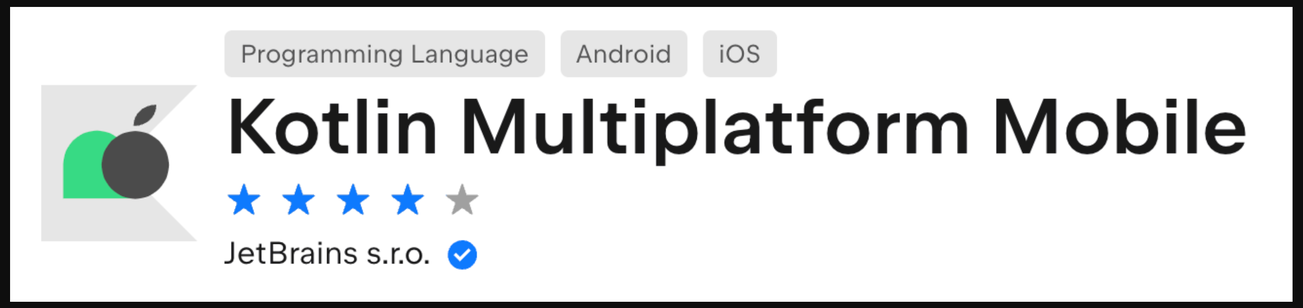
Prerequisites

Prerequisites

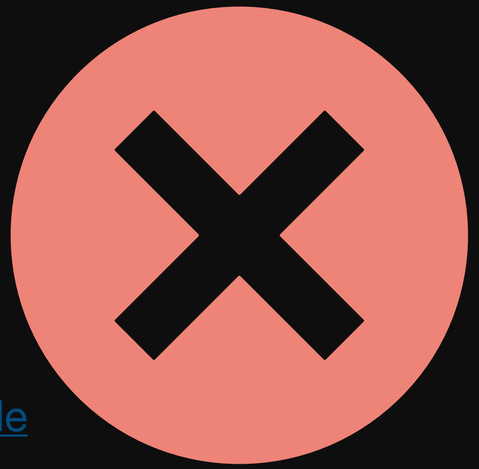
- All things related to UI and coding with compose multiplatform
- Other parts that will not be covered
  - Dependency injection
  - Networking
  - Persistence layers
  - ..

Wizard

Wizard



https://plugins.jetbrains.com/plugin/14936-kotlin-multiplatform-mobile



https://kmp.jetbrains.com/

Navigation

Navigation

- Jetpack Compose Navigation (Experimental)

  https://www.jetbrains.com/help/kotlin-multiplatform-dev/compose-navigation-routing.html

  > ⚠ The navigation library is currently Experimental. You're welcome to try it in your Compose Multiplatform projects. We would appreciate your feedback on GitHub ↗.

- Limitations

  - Deep links are not supported (not in the first release)
    https://github.com/JetBrains/compose-multiplatform/issues/85#issuecomment-2039108139

  - BackHandler and predictive back gestures are not supported on any platform besides Android
    https://github.com/JetBrains/compose-multiplatform/issues/4419

Navigation

- Third parties
  https://www.jetbrains.com/help/kotlin-multiplatform-dev/compose-navigation-routing.html#third-party-alternatives
  - Voyager
  - Decompose
  - Appyx
  - PreCompose

ViewModel & Lifecycle

ViewModel & Lifecycle

- Jetpack Compose ViewModel & Lifecycle
  https://www.jetbrains.com/help/kotlin-multiplatform-dev/compose-viewmodel.html
  https://www.jetbrains.com/help/kotlin-multiplatform-dev/compose-lifecycle.html

  ⚠ Support for the common `ViewModel` in Compose Multiplatform is Experimental.

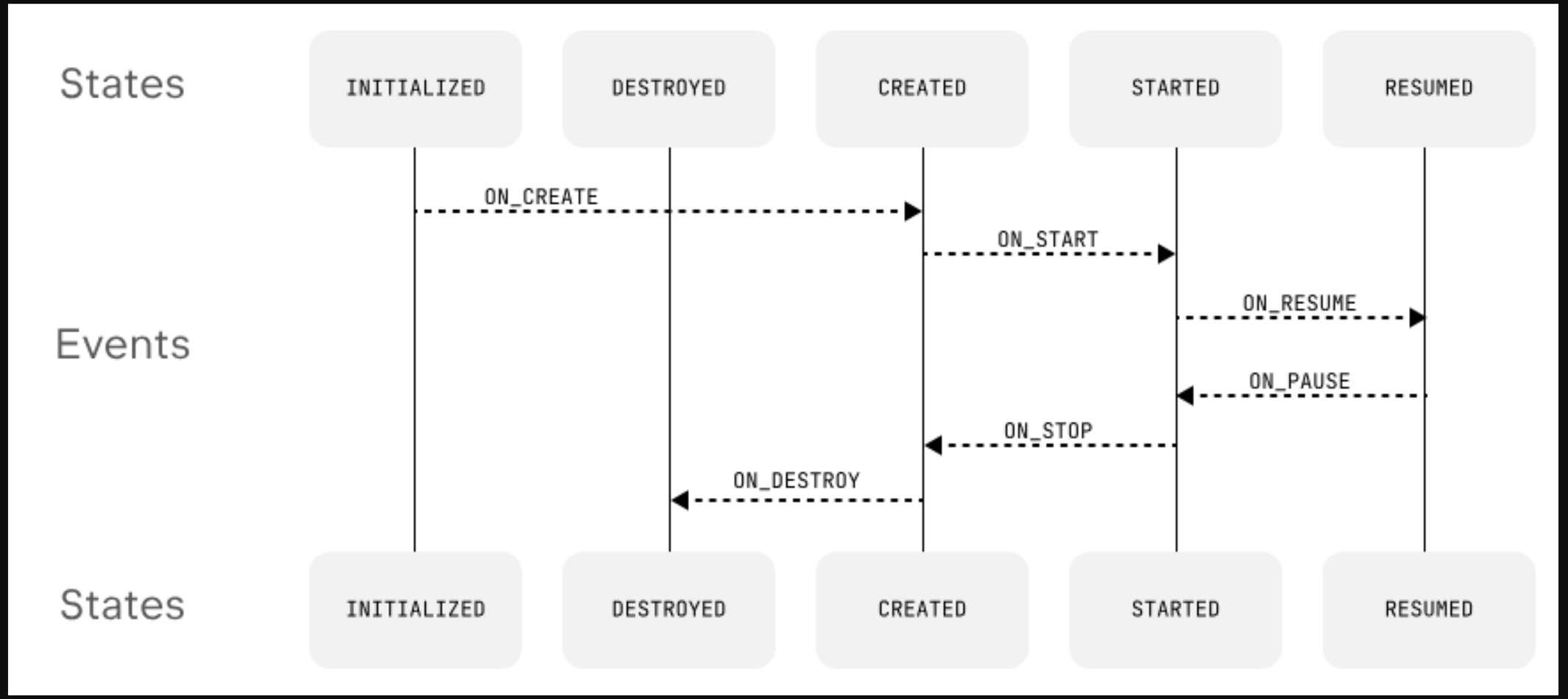
- Koin 3.6.0+ for ViewModel support
  https://github.com/InsertKoinIO/koin/issues/1826

- collectAsStateWithLifecycle & LifecycleEventEffect
  org.jetbrains.androidx.lifecycle:lifecycle-runtime-compose

```
val state by viewModel.state.collectAsStateWithLifecycle(
    lifecycleOwner = androidx.compose.ui.platform.LocalLifecycleOwner.current,
)
```

https://github.com/JetBrains/compose-multiplatform/issues/4749
https://issuetracker.google.com/issues/336842920

ViewModel & Lifecycle



https://www.jetbrains.com/help/kotlin-multiplatform-dev/compose-lifecycle.html

ViewModel & Lifecycle

Native events and notifications	Lifecycle event	Lifecycle state change
`viewDidDisappear`	`ON_STOP`	`STARTED` → `CREATED`
`viewWillAppear`	`ON_START`	`CREATED` → `STARTED`
`willResignActive`	`ON_PAUSE`	`RESUMED` → `STARTED`
`didBecomeActive`	`ON_RESUME`	`STARTED` → `RESUMED`
`didEnterBackground`	`ON_STOP`	`STARTED` → `CREATED`
`willEnterForeground`	`ON_START`	`CREATED` → `STARTED`
`viewControllerDidLeaveWindowHierarchy`	`ON_DESTROY`	`CREATED` → `DESTROYED`

https://www.jetbrains.com/help/kotlin-multiplatform-dev/compose-lifecycle.html

ViewModel & Lifecycle

- Third parties
  https://www.jetbrains.com/help/kotlin-multiplatform-dev/compose-navigation-routing.html#third-party-alternatives
  - Voyager
  - Decompose
  - Appyx
  - PreCompose

Resources

Resources

- commonMain/composeResources

```
compose.resources {
    publicResClass = true
    packageOfResClass = "my.sample.library.resources"
    generateResClass = always
}
```

https://www.jetbrains.com/help/kotlin-multiplatform-dev/compose-images-resources.html#configuration

- Qualifiers
  https://www.jetbrains.com/help/kotlin-multiplatform-dev/compose-images-resources.html#qualifiers
  - language two-letter (ISO 639-1) or a three-letter (ISO 639-2);
    plus two-letter ISO 3166-1-alpha-2 regional code
  - theme light or dark
  - density ldpi, mdpi, hdpi, xhdpi, xxhdpi, xxxhdpi

Resources

- Strings \n \t \uXXXX
- String Arrays
- Plurals zero, one, two, few, many, and other
- Fonts .ttf or .otf
- Drawables png, jpg, bmp, webp, xml (as on Android), svg (except Android)
- Raw

https://www.jetbrains.com/help/kotlin-multiplatform-dev/compose-images-resources.html#resource-usage
https://github.com/JetBrains/compose-multiplatform/tree/master/components/resources/demo

Resources

```
val Icons.Collapse: ImageVector
    get() {
        val current = _collapse
        if (current != null) return current

        return ImageVector.Builder(
            name = "androidx.compose.material.MaterialTheme.Collapse",
            defaultWidth = 24.0.dp,
            defaultHeight = 24.0.dp,
            viewportWidth = 24.0f,
            viewportHeight = 24.0f,
        ).apply { this: ImageVector.Builder
            path(
                fill = SolidColor(Color( color: 0xFF000000)),
            ) { this: PathBuilder
                moveTo(x = 12.6111f, y = 8.61572f)
                lineTo(x = 11.3889f, y = 8.61572f)
                curveTo(x1 = 10.9244f, y1 = 9.26885f, x2 = 10.3989f, y2 =
                curveTo(x1 = 9.20111f, y1 = 10.967f, x2 = 8.59f, y2 = 11.4
                curveTo(x1 = 7.68556f, y1 = 12.012f, x2 = 7.44111f, y2 = 1
                curveTo(x1 = 6.87889f, y1 = 12.4395f, x2 = 6.74444f, y2 =
                lineTo(x = 6.5f, y = 15.3845f)
                curveTo(x1 = 6.95222f, y1 = 15.1826f, x2 = 7.33111f, y2 =
                curveTo(x1 = 8.41889f, y1 = 14.3038f, x2 = 8.93222f, y2 =
```

SVG to Compose

https://github.com/rafaeltonholo/svg-to-compose
https://github.com/DevSrSouza/svg-to-compose

Image Loading

Image Loading

https://github.com/coil-kt/coil

https://github.com/Kamel-Media/Kamel

- 3.0+ (alpha)

Accessibility

Accessibility

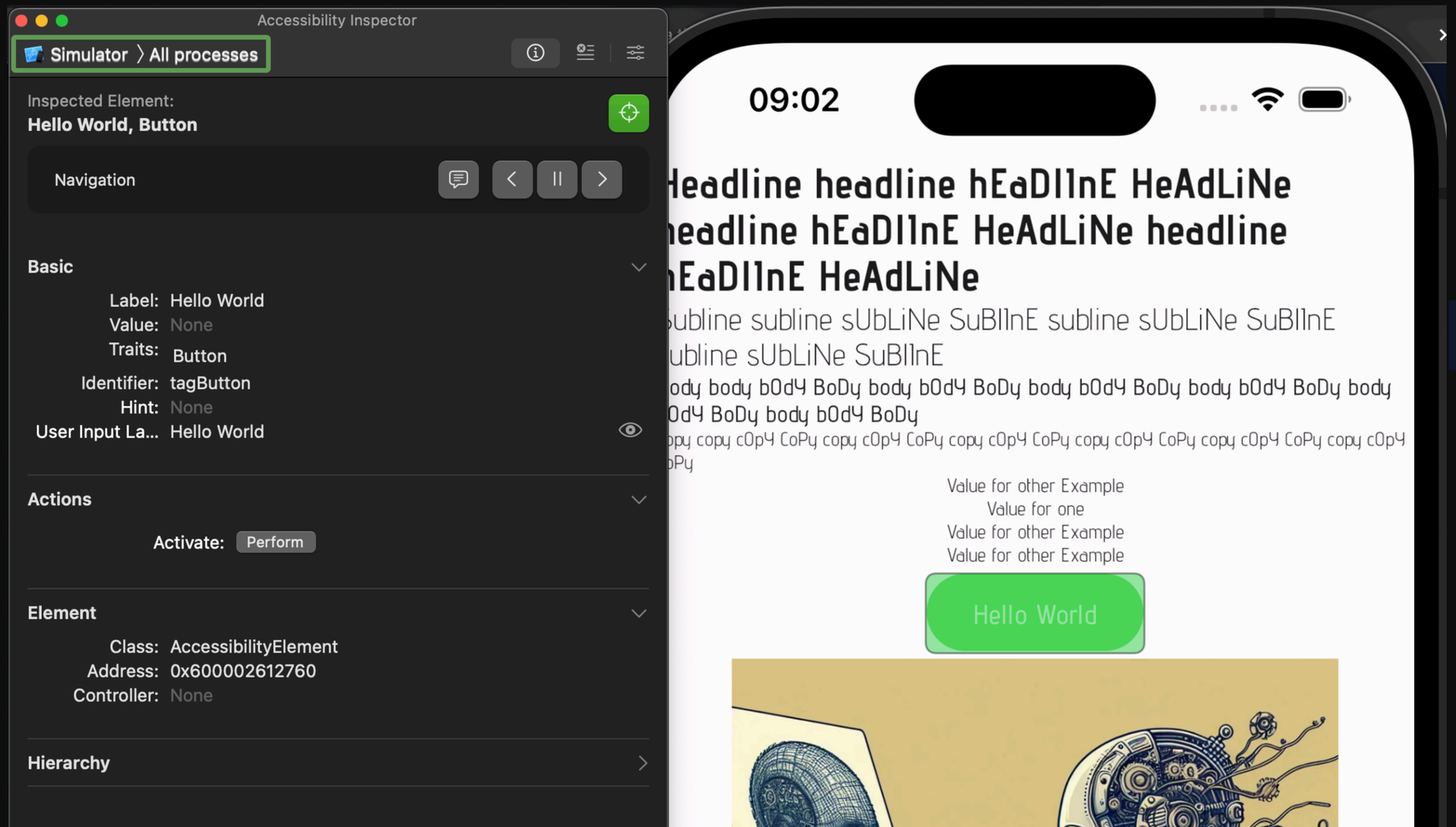
- Android TalkBack
- iOS VoiceOver
  https://www.jetbrains.com/help/kotlin-multiplatform-dev/compose-ios-accessibility.html

```
@OptIn(ExperimentalComposeApi::class)
fun mainViewController(): UIViewController = ComposeUIViewController(
    configure = {
        accessibilitySyncOptions = AccessibilitySyncOptions.Always(debugLogger = null)
    },
) {
    Root()
}
```

https://github.com/JetBrains/compose-multiplatform/issues/4401

Accessibility

- iOS testTag = accessibilityIdentifier




```
tapOn:
  id: tagButton
```

Accessibility

- Android semantics { testTagsAsResourceId = true }
- Setting at root composable entry may not be enough
- Watch out for (Alert)Dialogs!

```
expect fun Modifier.testTagsAsResourceId(): Modifier
```

- /androidMain

```
@OptIn(ExperimentalComposeUiApi::class)
actual fun Modifier.testTagsAsResourceId(): Modifier {
    return this then Modifier.semantics { testTagsAsResourceId = true }
}
```

Input & Keyboard

Input & Keyboard

- KeyboardActions
- KeyboardOptions
  - keyboardType
  - imeAction
- BasicTextField2 basic support for desktop, other platforms are planned for 1.7.0

Modifiers

Modifiers

- Some modifiers are missing usually those that are android specific
- Examples: motionEventSpy & pointerInteropFilter

Window Size Classes

Window Size Classes

Support material3-window-size-class #2404

Open **manosbatsis** opened this issue on Oct 14, 2022 · 10 comments

https://github.com/JetBrains/compose-multiplatform/issues/2404

manosbatsis commented on Oct 14, 2022

See https://developer.android.com/reference/kotlin/androidx/compose/material3/windowsizeclass/WindowSizeClass

☺ 👍 32 👀 1

chrisbanes commented on Jun 25, 2023

FYI: I built https://github.com/chrisbanes/material3-windowsizeclass-multiplatform as a stop-gap until support comes to Compose Multiplatform.

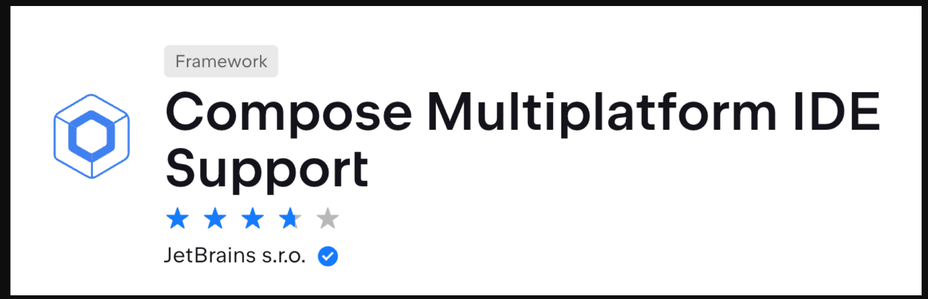
☺ 👍 12 🎉 2 ❤️ 11

Platform	Supported	Sample
Android	✅	android-app
iOS	✅	ios-app
Desktop (JVM)	✅	desktop-app
Web	✅ (experimental)	web-app

https://github.com/chrisbanes/material3-windowsizeclass-multiplatform

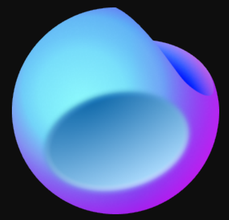
IDE

IDE



- Android Studio / IntelliJ
- Requires Plugin
  https://plugins.jetbrains.com/plugin/16541-compose-multiplatform-ide-support
- No direct swift support within IDE

IDE



Fleet

- Still in preview
- Free as long as its in preview
- Under heavy development
- Requires specific maximum Android Gradle Plugin version!
- Develop Swift code from within Fleet using Smart Mode

Previews

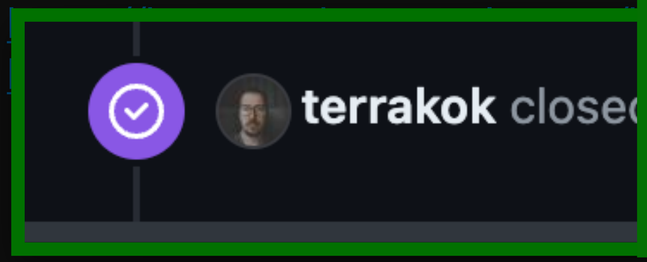
Previews

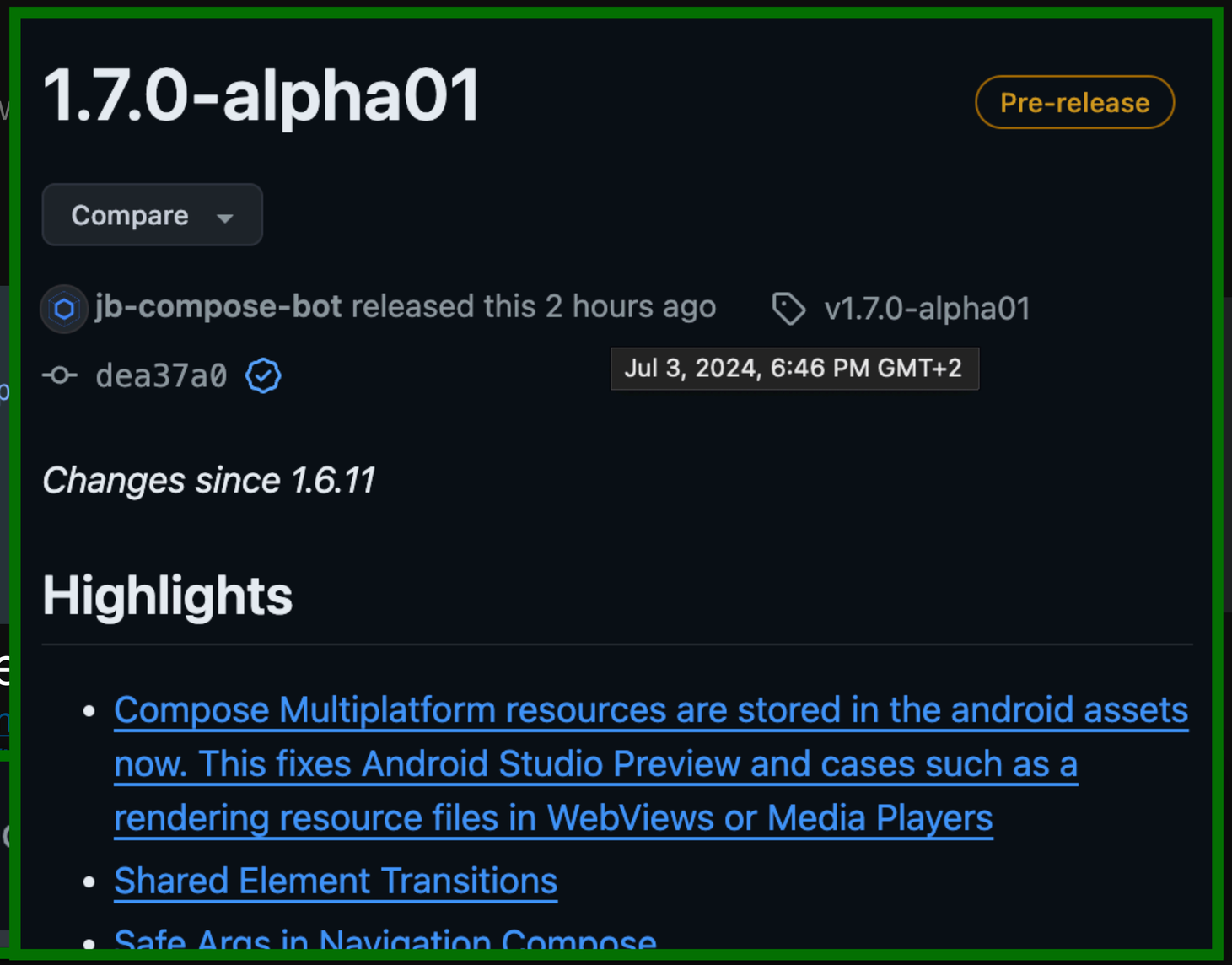
Android Studio live preview

- /androidMain

```
@Composable
@androidx.compose.ui.tooling.p
fun HomeScreenPreview() {
    AppTheme {
        HomeScreen()
    }
}
```

- org.jetbrains.compose
https://github.com/JetBrains/com



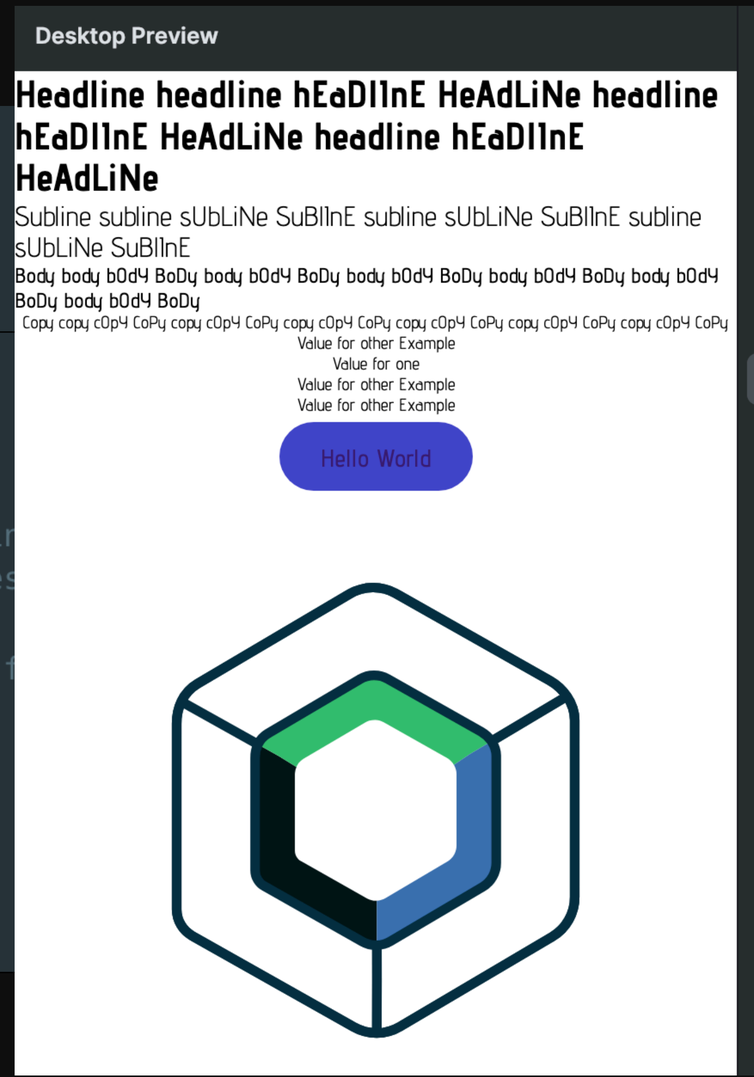


Previews

Android Studio static preview

```
@androidx.compose.desktop.ui.tooling.preview.Preview
@Composable
private fun PreviewWithinIntelliJ() {
    AppTheme {
        HomeScreen()
    }
}
```

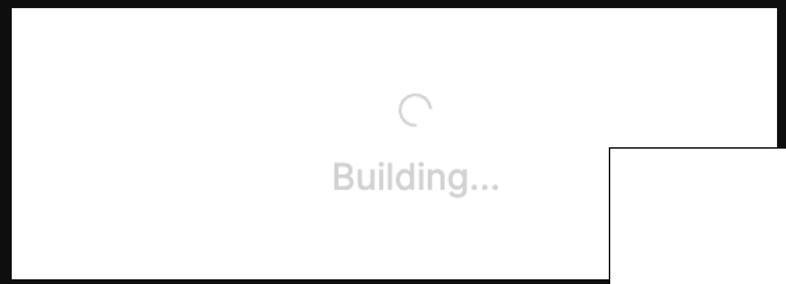
```
package androidx.compose.desktop.ui.tooling.preview

/**
 * This exists to show a preview of a composable function in
 * https://github.com/JetBrains/compose-multiplatform/issues
 * is resolved.
 * Although this is not a live preview like we are used to f
useful to see a static preview.
 */
@OptIn(ExperimentalMultiplatform::class)
@OptionalExpectation
expect annotation class Preview()
```

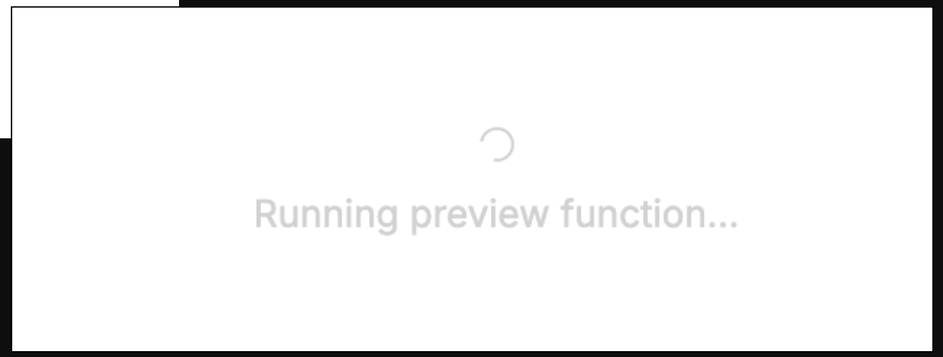


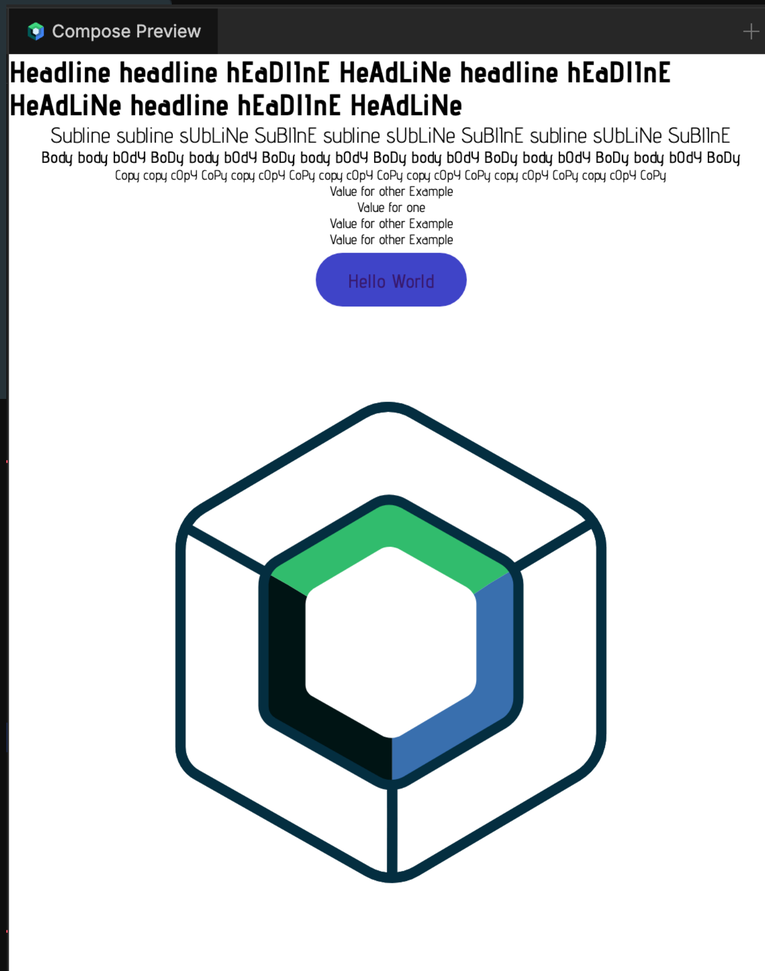
Previews

Fleet static preview that updates automatically

```
@org.jetbrains.compose.ui.tooling.preview.Preview
@Composable
private fun PreviewWithinFleet() {
    AppTheme {
        HomeScreen()
    }
}
```

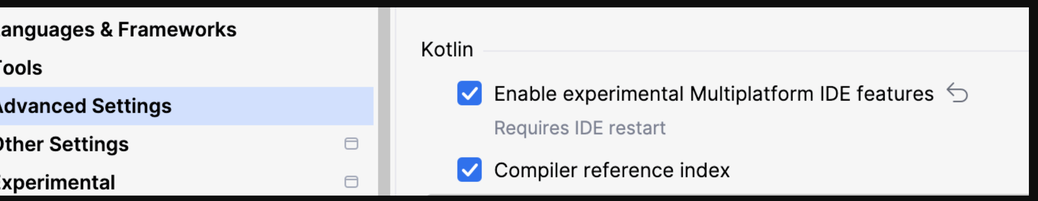


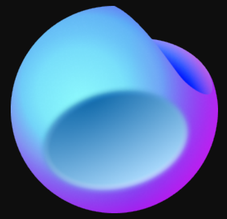


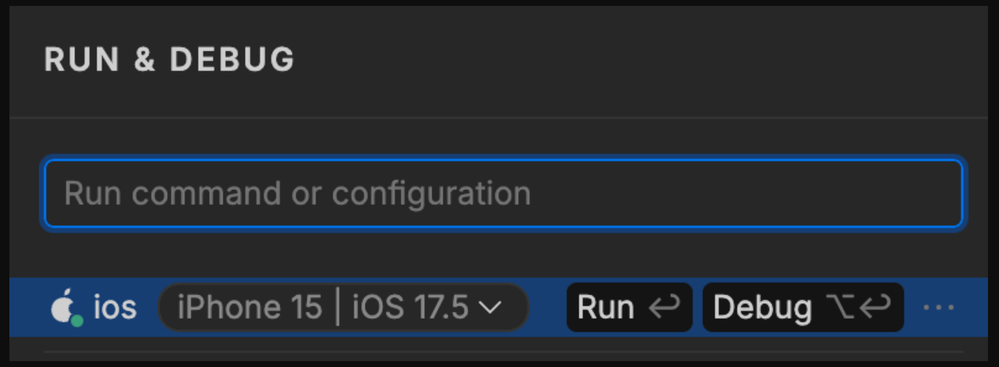


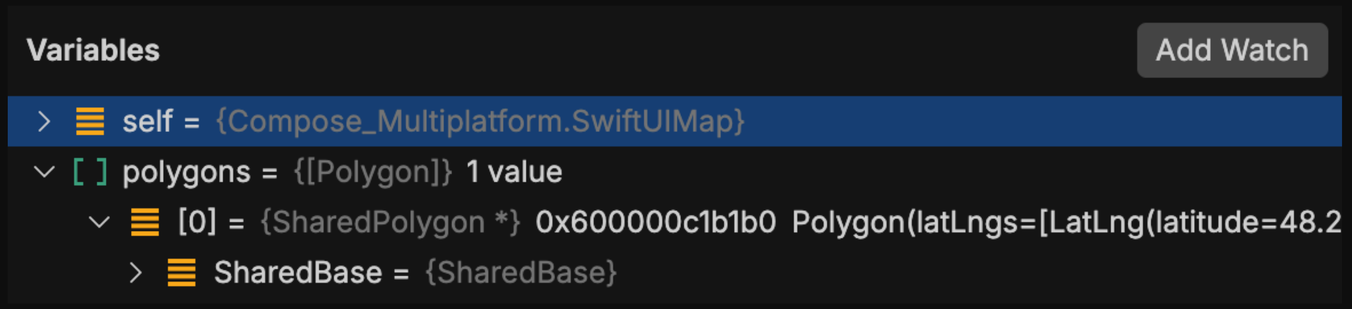
Debugging

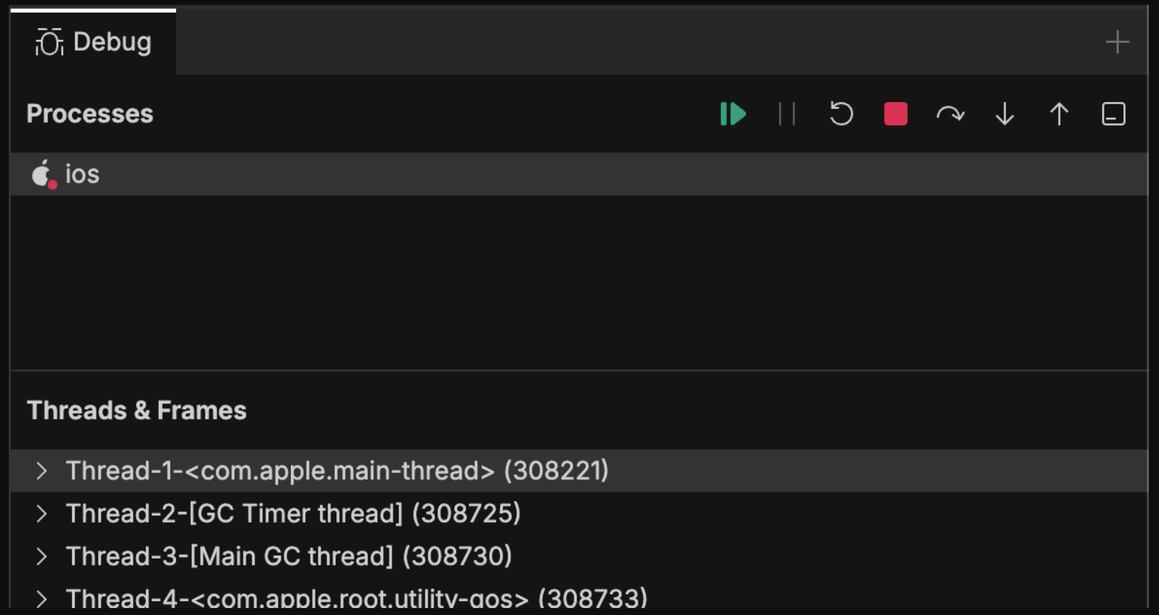
Debugging


anguages & Frameworks
ools
dvanced Settings
ther Settings
xperimental
Kotlin
Enable experimental Multiplatform IDE features
Requires IDE restart
Compiler reference index




RUN & DEBUG
Run command or configuration
ios
iPhone 15 | iOS 17.5
Run
Debug


Variables
Add Watch
self = {Compose_Multiplatform.SwiftUIMap}
polygons = {[Polygon]} 1 value
[0] = {SharedPolygon *} 0x600000c1b1b0 Polygon(latLngs=[LatLng(latitude=48.2
SharedBase = {SharedBase}


Debug
Processes
ios
Threads & Frames
Thread-1-<com.apple.main-thread> (308221)
Thread-2-[GC Timer thread] (308725)
Thread-3-[Main GC thread] (308730)
Thread-4-<com.apple.root.utility-qos> (308733)

Native UI (iOS)

Native UI (iOS)

- iosMain supports Cocoapods
  https://kotlinlang.org/docs/multiplatform-ios-dependencies.html

- No SwiftPackageManager
  https://youtrack.jetbrains.com/issue/KT-53877/Support-Swift-Package-Manager-in-Kotlin-Multiplatform

```
Kotlin supports interoperability with Objective-C dependencies and Swift dependencies
if their APIs are exported to Objective-C with the @objc attribute. Pure Swift
dependencies are not yet supported.
```

- Sharing Swift Code
  https://dev.to/ttypic/going-swiftly-using-a-swift-only-libraries-in-your-kotlin-multiplatform-app-1ml9
  https://github.com/KodeinKoders/playground-SwiftLib-in-KMMLib
  https://kotlinlang.org/docs/native-objc-interop.html#mappings

Native UI (iOS)

- Kotlin plugin to support Swift Package dependencies

  https://github.com/PaGr0m/kotlin-spm-plugin

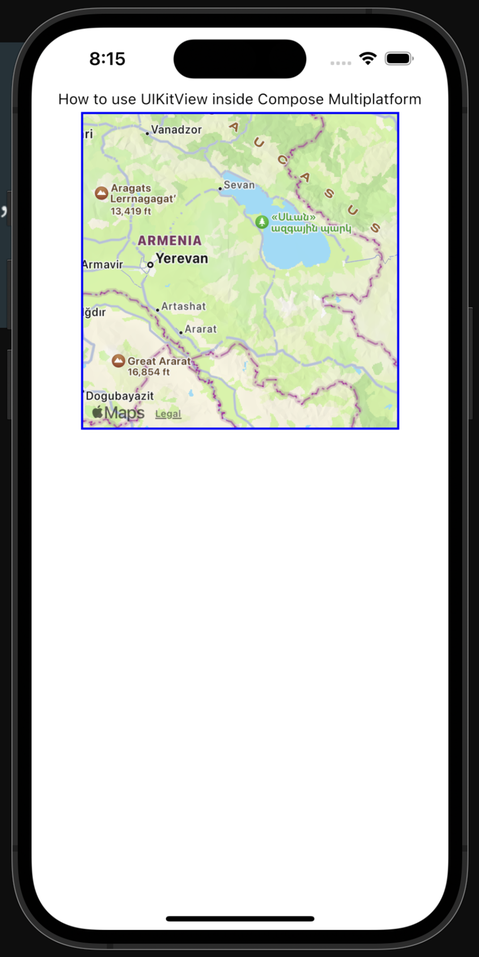
  Two-way interoperability between Kotlin dependencies and Kotlin-based Swift Package Multiplatform

- NOTE: At the moment, Kotlin is not directly compatible with Swift. Therefore, you can only connect Objective-C libraries that have a Swift package file.

Native UI (iOS)

- Use UIKitView just like AndroidView

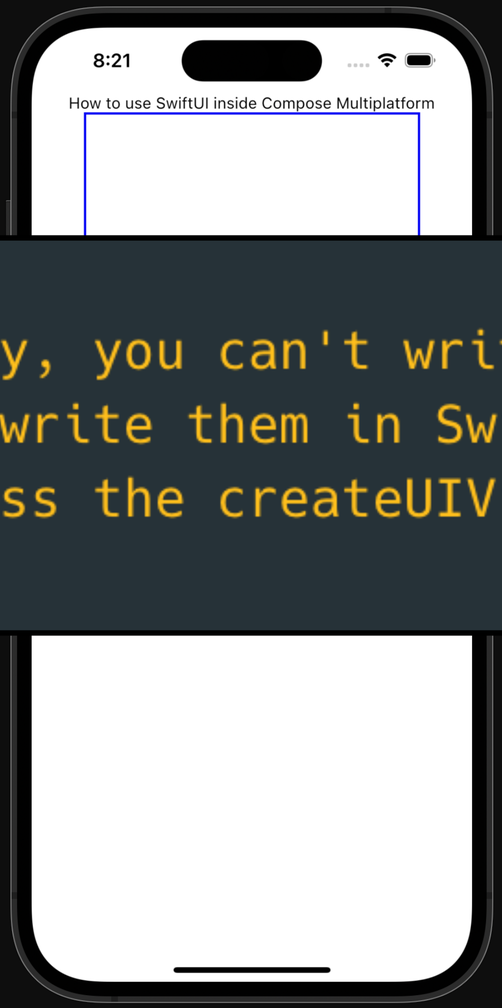
```
UIKitView(
    factory = { MKMapView() },
    modifier = Modifier.size(300.dp),
)
```




```
@Composable
actual fun VideoPlayer(url: String, modifier: Modifier) {
    val player = remember { AVPlayer(uRL = NSURL.URLWithString(url)!!) }
    val playerLayer = remember { AVPlayerLayer() }
    val avPlayerViewController = remember { AVPlayerViewController() }
    avPlayerViewController.player = player
    avPlayerViewController.showsPlaybackControls = true
    playerLayer.player = player

    UIKitView(
        factory = {
            UIView().apply {
                addSubview(avPlayerViewController.view)
            }
        },
        onResize = { view: UIView, rect: CValue<CGRect> ->
            CATransaction.begin()
            CATransaction.setValue(true, kCATransactionDisableActions)
            view.layer.setFrame(rect)
            playerLayer.setFrame(rect)
            avPlayerViewController.view.layer.frame = rect
            CATransaction.commit()
        },
        update = {
            player.play()
            avPlayerViewController.player?.play()
        },
        modifier = modifier,
    )
    DisposableEffect(Unit) {
        onDispose {
            player.pause()
        }
    }
}
```

Native UI (iOS)

- SwiftUI within Compose




```
@OptIn(ExperimentalForeignApi::class)
fun ComposeEntryPointWithUIViewController(
    createUIViewController: () -> UIViewController
): UIViewController = ComposeUIViewController {
    Column(
        Modifier
            .fillMaxSize()
            .windowInsetsPadding(WindowInsets.systemBars),
```

Currently, you can't write SwiftUI structures directly in Kotlin. Instead, you have to write them in Swift and pass them to a Kotlin function. In your Swift code, pass the createUIViewController to your entry point function code.

```
Main_iosKt.ComposeEntryPointWithUIViewController(createUIViewController: {
() -> UIViewController in
    let swiftUIView = VStack {
        Text("SwiftUI in Compose Multiplatform")
    }
    return UIHostingController(rootView: swiftUIView)
})
```

Native UI (iOS)

Currently, you can't write SwiftUI structures directly in Kotlin. Instead, you have to write them in Swift and pass them to a Kotlin function. In your Swift code, pass the createUIViewController to your entry point function code.

```
UIKitView(
    factory = { factory.view },
    update = {
        ... // How ?
    },
    modifier = modifier,
)
```

```
UIKitViewController(
        factory = { factory.viewController },
        update = {
            ... // How ?
        },
        modifier = modifier,
    )
```

Native UI (iOS)

```kotlin
@Composable
expect fun Map(
    modifier: Modifier = Modifier,
    contentPadding: PaddingValues = PaddingValues(all = 0.dp),
    polygons: ImmutableList<Polygon> = persistentListOf(),
)
```

```kotlin
@Composable
actual fun Map(
  modifier: Modifier,
  contentPadding: PaddingValues,
  polygons: ImmutableList<Polygon>,
) {
    val useUIKitInsteadOfSwiftUI = false
    if (useUIKitInsteadOfSwiftUI) {
        MapWithUIKit(contentPadding, polygons, modifier)
    } else {
        MapWithSwiftUI(contentPadding, polygons, modifier)
    }
}
```

Sample

https://github.com/rschattauer/compose_multiplatform

GitHub

Native UI (iOS)

```kotlin
@Composable
expect fun Map(
    modifier: Modifier = Modifier,
    contentPadding: PaddingValues = PaddingValues(all = 0.dp),
    polygons: ImmutableList<Polygon> = persistentListOf(),
)
```

```kotlin
@Composable
actual fun Map(
  modifier: Modifier,
  contentPadding: PaddingValues,
  polygons: ImmutableList<Polygon>,
) {
    val useUIKitInsteadOfSwiftUI = false
    if (useUIKitInsteadOfSwiftUI) {
        MapWithUIKit(contentPadding, polygons, modifier)
    } else {
        MapWithSwiftUI(contentPadding, polygons, modifier)
    }
}
```

Sample

https://github.com/rschattauer/compose_multiplatform

GitHub

Native UI (iOS)

```
@Composable
private fun MapWithSwiftUI(
    contentPadding: PaddingValues,
    polygons: ImmutableList<Polygon>,
    modifier: Modifier = Modifier,
) {
    val polygonsState = remember { MutableStateFlow(polygons) }
    val contentPaddingState = remember { MutableStateFlow(contentPadding) }
    val factory = remember { mapWithSwiftUIViewFactory(polygonsState.asStateFlow(), contentPaddingState.asStateFlow()) }
    UIKitViewController(
        factory = { factory.viewController },
        update = {
            polygonsState.update { polygons }
            contentPaddingState.update { contentPadding }
        },
        modifier = modifier,
    )
}

internal lateinit var mapWithSwiftUIViewFactory: (
    polygonsState: StateFlow<ImmutableList<Polygon>>,
    contentPaddingState: StateFlow<PaddingValues>,
) -> MapWithSwiftUIViewFactory

interface MapWithSwiftUIViewFactory {
    val viewController: UIViewController
}
```

Native UI (iOS)

```
@Composable
private fun MapWithSwiftUI(
    contentPadding: PaddingValues,
    polygons: ImmutableList<Polygon>,
    modifier: Modifier = Modifier,
) {
    val polygonsState = remember { MutableStateFlow(polygons) }
    val contentPaddingState = remember { MutableStateFlow(contentPadding) }
    val factory = remember { mapWithSwiftUIViewFactory(polygonsState.asStateFlow(), contentPaddingState.asStateFlow()) }
    UIKitViewController(
        factory = { factory.viewController },
        update = {
            polygonsState.update { polygons }
            contentPaddingState.update { contentPadding }
        },
        modifier = modifier,
    )
}
```

```
internal lateinit var mapWithSwiftUIViewFactory: (
    polygonsState: StateFlow<ImmutableList<Polygon>>,
    contentPaddingState: StateFlow<PaddingValues>,
) -> MapWithSwiftUIViewFactory

interface MapWithSwiftUIViewFactory {
    val viewController: UIViewController
}
```

Native UI (iOS)

```kotlin
@Composable
private fun MapWithSwiftUI(
    contentPadding: PaddingValues,
    polygons: ImmutableList<Polygon>,
    modifier: Modifier = Modifier,
) {
    val polygonsState = remember { MutableStateFlow(polygons) }
    val contentPaddingState = remember { MutableStateFlow(contentPadding) }
                                                                    low()) }
```

```kotlin
internal lateinit var mapWithSwiftUIViewFactory: (
    polygonsState: StateFlow<ImmutableList<Polygon>>,
    contentPaddingState: StateFlow<PaddingValues>,
) -> MapWithSwiftUIViewFactory

interface MapWithSwiftUIViewFactory {
    val viewController: UIViewController
}
```

```
}
```

Native UI (iOS)

```
@Composable
private fun MapWithSwiftUI(
    contentPadding: PaddingValues,
    polygons: ImmutableList<Polygon>,
    modifier: Modifier = Modifier,
) {
    val polygonsState = remember { MutableStateFlow(polygons) }
    val contentPaddingState = remember { MutableStateFlow(contentPadding) }
    val factory = remember { mapWithSwiftUIViewFactory(polygonsState.asStateFlow(), contentPaddingState.asStateFlow()) }
    UIKitViewController(
        factory = { factory.viewController },
        update = {
            polygonsState.update { polygons }
            contentPaddingState.update { contentPadding }
        },
        modifier = modifier,
    )
}
```

```
internal lateinit var mapWithSwiftUIViewFactory: (
    polygonsState: StateFlow<ImmutableList<Polygon>>,
    contentPaddingState: StateFlow<PaddingValues>,
) -> MapWithSwiftUIViewFactory

interface MapWithSwiftUIViewFactory {
    val viewController: UIViewController
}
```

Native UI (iOS)

```
@Composable
private fun MapWithSwiftUI(
    contentPadding: PaddingValues,
    polygons: ImmutableList<Polygon>,
    modifier: Modifier = Modifier,
) {
    val polygonsState = remember { MutableStateFlow(polygons) }
    val contentPaddingState = remember { MutableStateFlow(contentPadding) }
    val factory = remember { mapWithSwiftUIViewFactory(polygonsState.asStateFlow(), contentPaddingState.asStateFlow()) }
    UIKitViewController(
        factory = { factory.viewController },
        update = {
            polygonsState.update { polygons }
            contentPaddingState.update { contentPadding }
        },
        modifier = modifier,
    )
}

internal lateinit var mapWithSwiftUIViewFactory: (
    polygonsState: StateFlow<ImmutableList<Polygon>>,
    contentPaddingState: StateFlow<PaddingValues>,
) -> MapWithSwiftUIViewFactory

interface MapWithSwiftUIViewFactory {
    val viewController: UIViewController
}
```

```
fun setViewFactories(
    mapWithUIKitViewFactory: () -> MapWithUIKitViewFactory,
    mapWithSwiftUIViewFactory: (
        polygonsState: StateFlow<ImmutableList<Polygon>>,
        contentPaddingState: StateFlow<PaddingValues>,
    ) -> MapWithSwiftUIViewFactory,
) {
    ui.component.mapWithUIKitViewFactory = mapWithUIKitViewFactory
    ui.component.mapWithSwiftUIViewFactory = mapWithSwiftUIViewFactory
}
```

Native UI (iOS)

```
@Composable
private fun MapWithSwiftUI(
    contentPadding: PaddingValues,
    polygons: Im
    modifier: Mo
) {
    val polygons
    val contentP
    val factory
    UIKitViewCon
        factory
        update =
            poly
            cont
        },
        modifier
    )
}

internal lateini
    polygonsStat
    contentPaddi
) -> MapWithSwif

interface MapWit
    val viewCont
}
```

```
fun setViewFactories(
    mapWithUIKitViewFactory: () -> MapWithUIKitViewFactory,
    mapWithSwiftUIViewFactory: (
        polygonsState: StateFlow<ImmutableList<Polygon>>,
        contentPaddingState: StateFlow<PaddingValues>,
    ) -> MapWithSwiftUIViewFactory,
) {
    ui.component.mapWithUIKitViewFactory = mapWithUIKitViewFactory
    ui.component.mapWithSwiftUIViewFactory = mapWithSwiftUIViewFactory
}
```

Native UI (iOS)

```
@Composable
private fun MapWithSwiftUI(
    contentPadding: PaddingValues,
    polygons: ImmutableList<Polygon>,
    modifier: Modifier = Modifier,
) {
    val polygonsState = remember { MutableStateFlow(polygons) }
    val contentPaddingState = remember { MutableStateFlow(contentPadding) }
    val factory = remember { mapWithSwiftUIViewFactory(polygonsState.asStateFlow(), contentPaddingState.asStateFlow()) }
    UIKitViewController(
        factory = { factory.viewController },
        update = {
            polygonsState.update { polygons }
            contentPaddingState.update { contentPadding }
        },
        modifier = modifier,
    )
}

internal lateinit var mapWithSwiftUIViewFactory: (
    polygonsState: StateFlow<ImmutableList<Polygon>>,
    contentPaddingState: StateFlow<PaddingValues>,
) -> MapWithSwiftUIViewFactory

interface MapWithSwiftUIViewFactory {
    val viewController: UIViewController
}
```

```
fun setViewFactories(
    mapWithUIKitViewFactory: () -> MapWithUIKitViewFactory,
    mapWithSwiftUIViewFactory: (
        polygonsState: StateFlow<ImmutableList<Polygon>>,
        contentPaddingState: StateFlow<PaddingValues>,
    ) -> MapWithSwiftUIViewFactory,
) {
    ui.component.mapWithUIKitViewFactory = mapWithUIKitViewFactory
    ui.component.mapWithSwiftUIViewFactory = mapWithSwiftUIViewFactory
}
```

Native UI (iOS)

```kotlin
@Composable
private fun MapWithSwiftUI(
    contentPadding: PaddingValues,
    polygons: ImmutableList<Polygon>,
    modifier: Modifier = Modifier,
) {
    val polygonsState = remember { MutableStateFlow(polygons) }
    val contentPaddingState = remember { MutableStateFlow(contentPadding) }
    val factory = remember { mapWithSwiftUIViewFactory(polygonsState.asStateFlow(), contentPaddingState.asStateFlow()) }
    UIKitViewController(
        factory = { factory.viewController },
        update = {
            polygonsState.update { polygons }
            contentPaddingState.update { contentPadding }
        },
        modifier = modifier,
    )
}

internal lateinit var mapWithSwiftUIViewFactory: (
    polygonsState: StateFlow<ImmutableList<Polygon>>,
    contentPaddingState: StateFlow<PaddingValues>,
) -> MapWithSwiftUIViewFactory

interface MapWithSwiftUIViewFactory {
    val viewController: UIViewController
}
```

```kotlin
                                                    KitViewFactory,

        polygonsState: StateFlow<ImmutableList<Polygon>>,
        contentPaddingState: StateFlow<PaddingValues>,
    ) -> MapWithSwiftUIViewFactory,
) {
    ui.component.mapWithUIKitViewFactory = mapWithUIKitViewFactory
    ui.component.mapWithSwiftUIViewFactory = mapWithSwiftUIViewFactory
}
```

Native UI (iOS)

```
@Composable
private fun MapWithSwiftUI(
    contentPadding: PaddingValues,
    polygons: ImmutableList<Polygon>,
    modifier: Modifier = Modifier,
) {
val polygonsState = remember { MutableStateFlow(polygons) }
val contentPaddingState = remember { MutableStateFlow(contentPadding) }
val factory = remember { mapWithSwiftUIViewFactory(polygonsState.asStateFlow(), contentPaddingState.as
UIKitViewController(
    factory = { factory.viewController },
    update = {
        polygonsState.update { polygons }
        contentPaddingState.update { contentPadding }
    },
    modifier = modifier,
)
interface MapWithSwiftUIViewFactory {
    val viewController: UIViewController
}
```

Native UI (iOS)

```kotlin
@Composable
private fun MapWithSwiftUI(
    contentPadding: PaddingValues,
    polygons: ImmutableList<Polygon>,
    modifier: Modifier = Modifier,
) {
    val polygonsState = remember { MutableStateFlow(polygons) }
    val contentPaddingState = remember { MutableStateFlow(contentPadding) }
    val factory = remember { mapWithSwiftUIViewFactory(polygonsState.asStateFlow(), contentPaddingState.asStateFlow()) }
    UIKitViewController(
        factory = { factory.viewController },
        update = {
            polygonsState.update { polygons }
            contentPaddingState.update { contentPadding }
        },
        modifier = modifier,
    )
}

internal lateinit var mapWithSwiftUIViewFactory: (
    polygonsState: StateFlow<ImmutableList<Polygon>>,
    contentPaddingState: StateFlow<PaddingValues>,
) -> MapWithSwiftUIViewFactory

interface MapWithSwiftUIViewFactory {
    val viewController: UIViewController
}
```

```kotlin
                                                    KitViewFactory,

        polygonsState: StateFlow<ImmutableList<Polygon>>,
        contentPaddingState: StateFlow<PaddingValues>,
    ) -> MapWithSwiftUIViewFactory,
) {
    ui.component.mapWithUIKitViewFactory = mapWithUIKitViewFactory
    ui.component.mapWithSwiftUIViewFactory = mapWithSwiftUIViewFactory
}
```

Native UI (iOS)

```
func mapWithSwiftUIViewFactory(
    polygonsState: SkieSwiftStateFlow<[shared.Polygon]>,
    contentPaddingState: SkieSwiftStateFlow<Foundation_layoutPaddingValues>
) -> MapWithSwiftUIViewFactory {
    return MapSwiftUIViewContainer(
        polygonsState: polygonsState,
        contentPaddingState: contentPaddingState
    )
}

class MapSwiftUIViewContainer: MapWithSwiftUIViewFactory {

    var viewController: UIViewController

    init(
      polygonsState: SkieSwiftStateFlow<[shared.Polygon]>,
      contentPaddingState: SkieSwiftStateFlow<Foundation_layoutPaddingValues
    ) {
        self.viewController = UIHostingController(rootView: SwiftUIMap(polygonsState: polygonsState, contentPaddingState: contentPaddingState))
    }
}

struct SwiftUIMap: View {

    var polygonsState: SkieSwiftStateFlow<[shared.Polygon]>
    var contentPaddingState: SkieSwiftStateFlow<Foundation_layoutPaddingValues>

    init(
        polygonsState: SkieSwiftStateFlow<[shared.Polygon]>,
        contentPaddingState: SkieSwiftStateFlow<Foundation_layoutPaddingValues>
    ) {
        self.polygonsState = polygonsState
        self.contentPaddingState = contentPaddingState
    }

    @State var polygons: Polygons?
    @State var contentPadding: Foundation_layoutPaddingValues?
    @State var viewport = Viewport.camera(center: .init(latitude: 48.2082, longitude: 16.3719), zoom: 14, bearing: 0, pitch: 0)
```

Native UI (iOS)

```
func mapWithSwiftUIViewFactory(
    polygonsState: SkieSwiftStateFlow<[shared.Polygon]>,
    contentPaddingState: SkieSwiftStateFlow<Foundation_layoutPaddingValues>
) -> MapWithSwiftUIViewFactory {
    return MapSwiftUIViewContainer(
        polygonsState: polygonsState,
        contentPaddingState: contentPaddingState
    )
}

class MapSwiftUIViewContainer: MapWithSwiftUIViewFactory {

    var viewController: UIViewController

    init(
      polygonsState: SkieSwiftStateFlow<[shared.Polygon]>,
      contentPaddingState: SkieSwiftStateFlow<Foundation_layoutPaddingValues
    ) {
        self.viewController = UIHostingController(rootView: SwiftUIMap(polygonsState: polygonsState, contentPaddingState: contentPaddingState))
    }
}

struct SwiftUIMap: View {

    var polygonsState: SkieSwiftStateFlow<[shared.Polygon]>
    var contentPaddingState: SkieSwiftStateFlow<Foundation_layoutPaddingValues>

    init(
        polygonsState: SkieSwiftStateFlow<[shared.Polygon]>,
        contentPaddingState: SkieSwiftStateFlow<Foundation_layoutPaddingValues>
    ) {
        self.polygonsState = polygonsState
        self.contentPaddingState = contentPaddingState
    }

    @State var polygons: Polygons?
    @State var contentPadding: Foundation_layoutPaddingValues?
    @State var viewport = Viewport.camera(center: .init(latitude: 48.2082, longitude: 16.3719), zoom: 14, bearing: 0, pitch: 0)
```

Native UI (iOS)

```
func mapWithSwiftUIViewFactory(
    polygonsState: SkieSwiftStateFlow<[shared.Polygon]>,
    contentPaddingState: SkieSwiftStateFlow<Foundation_layoutPaddingValues>
) -> MapWithSwiftUIViewFactory {
    return MapSwiftUIViewContainer(
        polygonsState: polygonsState,
        contentPaddingState: contentPaddingState
    )
}
```

```
    }
}

struct SwiftUIMap: View {

    var polygonsState: SkieSwiftStateFlow<[shared.Polygon]>
    var contentPaddingState: SkieSwiftStateFlow<Foundation_layoutPaddingValues>

    init(
        polygonsState: SkieSwiftStateFlow<[shared.Polygon]>,
        contentPaddingState: SkieSwiftStateFlow<Foundation_layoutPaddingValues>
    ) {
        self.polygonsState = polygonsState
        self.contentPaddingState = contentPaddingState
    }

    @State var polygons: Polygons?
    @State var contentPadding: Foundation_layoutPaddingValues?
    @State var viewport = Viewport.camera(center: .init(latitude: 48.2082, longitude: 16.3719), zoom: 14, bearing: 0, pitch: 0)
```

Native UI (iOS)

```
func mapWithSwiftUIViewFactory(
    polygonsState: SkieSwiftStateFlow<[shared.Polygon]>,
    contentPaddingState: SkieSwiftStateFlow<Foundation_layoutPaddingValues>
) -> MapWithSwiftUIViewFactory {
    return MapSwiftUIViewContainer(
        polygonsState: polygonsState,
        contentPaddingState: contentPaddingState
    )
}

class MapSwiftUIViewContainer: MapWithSwiftUIViewFactory {

    var viewController: UIViewController

    init(
      polygonsState: SkieSwiftStateFlow<[shared.Polygon]>,
      contentPaddingState: SkieSwiftStateFlow<Foundation_layoutPaddingValues
    ) {
        self.viewController = UIHostingController(rootView: SwiftUIMap(polygonsState: polygonsState, contentPaddingState: contentPaddingState))
    }
}

struct SwiftUIMap: View {

    var polygonsState: SkieSwiftStateFlow<[shared.Polygon]>
    var contentPaddingState: SkieSwiftStateFlow<Foundation_layoutPaddingValues>

    init(
        polygonsState: SkieSwiftStateFlow<[shared.Polygon]>,
        contentPaddingState: SkieSwiftStateFlow<Foundation_layoutPaddingValues>
    ) {
        self.polygonsState = polygonsState
        self.contentPaddingState = contentPaddingState
    }

    @State var polygons: Polygons?
    @State var contentPadding: Foundation_layoutPaddingValues?
    @State var viewport = Viewport.camera(center: .init(latitude: 48.2082, longitude: 16.3719), zoom: 14, bearing: 0, pitch: 0)
```

Native UI (iOS)

```
func mapWithSwiftUIViewFactory(
    polygonsState: SkieSwiftStateFlow<[shared.Polygon]>,
    contentPaddingState: SkieSwiftStateFlow<Foundation_layoutPaddingValues>
) -> MapWithSwiftUIViewFactory {
    return MapSwiftUIViewContainer(
        polygonsState: polygonsState,
        contentPaddingState: contentPaddingState
    )
}
```

```
class MapSwiftUIViewContainer: MapWithSwiftUIViewFactory {

    var viewController: UIViewController

    init(
      polygonsState: SkieSwiftStateFlow<[shared.Polygon]>,
      contentPaddingState: SkieSwiftStateFlow<Foundation_layoutPaddingValues
    ) {
        self.viewController = UIHostingController(rootView: SwiftUIMap(polygonsState: polygonsState, contentPaddingState: contentPaddingState))
    }
}
```

```
struct SwiftUIMap: View {

    var polygonsState: SkieSwiftStateFlow<[shared.Polygon]>
    var contentPaddingState: SkieSwiftStateFlow<Foundation_layoutPaddingValues>

    init(
        polygonsState: SkieSwiftStateFlow<[shared.Polygon]>,
        contentPaddingState: SkieSwiftStateFlow<Foundation_layoutPaddingValues>
    ) {
        self.polygonsState = polygonsState
        self.contentPaddingState = contentPaddingState
    }

    @State var polygons: Polygons?
    @State var contentPadding: Foundation_layoutPaddingValues?
    @State var viewport = Viewport.camera(center: .init(latitude: 48.2082, longitude: 16.3719), zoom: 14, bearing: 0, pitch: 0)
```

Native UI (iOS)

```
func mapWithSwiftUIViewFactory(
    polygonsState: SkieSwiftStateFlow<[shared.Polygon]>,
    contentPaddingState: SkieSwiftStateFlow<Foundation_layoutPaddingValues>
) -> MapWithSwiftUIViewFactory {
    return MapSwiftUIViewContainer(
        polygonsState: polygonsState
```

```
class MapSwiftUIViewContainer: MapWithSwiftUIViewFactory {

    var viewController: UIViewController

    init(
      polygonsState: SkieSwiftStateFlow<[shared.Polygon]>,
      contentPaddingState: SkieSwiftStateFlow<Foundation_layoutPaddingValues
    ) {
        self.viewController = UIHostingController(rootView: SwiftUIMap(polygonsState: polygonsState, contentPadding
    }
}
```

```
    init(
        polygonsState: SkieSwiftStateFlow<[shared.Polygon]>,
        contentPaddingState: SkieSwiftStateFlow<Foundation_layoutPaddingValues>
    ) {
        self.polygonsState = polygonsState
        self.contentPaddingState = contentPaddingState
    }

    @State var polygons: Polygons?
    @State var contentPadding: Foundation_layoutPaddingValues?
    @State var viewport = Viewport.camera(center: .init(latitude: 48.2082, longitude: 16.3719), zoom: 14, bearing: 0, pitch: 0)
```

Native UI (iOS)

```
func mapWithSwiftUIViewFactory(
    polygonsState: SkieSwiftStateFlow<[shared.Polygon]>,
    contentPaddingState: SkieSwiftStateFlow<Foundation_layoutPaddingValues>
) -> MapWithSwiftUIViewFactory {
    return MapSwiftUIViewContainer(
        polygonsState: polygonsState,
        contentPaddingState: contentPaddingState
    )
}
```

```
class MapSwiftUIViewContainer: MapWithSwiftUIViewFactory {

    var viewController: UIViewController

    init(
      polygonsState: SkieSwiftStateFlow<[shared.Polygon]>,
      contentPaddingState: SkieSwiftStateFlow<Foundation_layoutPaddingValues
    ) {
        self.viewController = UIHostingController(rootView: SwiftUIMap(polygonsState: polygonsState, contentPaddingState: contentPaddingState))
    }
}
```

```
struct SwiftUIMap: View {

    var polygonsState: SkieSwiftStateFlow<[shared.Polygon]>
    var contentPaddingState: SkieSwiftStateFlow<Foundation_layoutPaddingValues>

    init(
        polygonsState: SkieSwiftStateFlow<[shared.Polygon]>,
        contentPaddingState: SkieSwiftStateFlow<Foundation_layoutPaddingValues>
    ) {
        self.polygonsState = polygonsState
        self.contentPaddingState = contentPaddingState
    }

    @State var polygons: Polygons?
    @State var contentPadding: Foundation_layoutPaddingValues?
    @State var viewport = Viewport.camera(center: .init(latitude: 48.2082, longitude: 16.3719), zoom: 14, bearing: 0, pitch: 0)
```

Native UI (iOS)

```
func mapWithSwiftUIViewFactory(
    polygonsState: SkieSwiftStateFlow<[shared.Polygon]>,
    contentPaddingState: SkieSwiftStateFlow<Foundation_layoutPaddingValues>
) -> MapWithSwiftUIViewFactory {
    return MapSwiftUIViewContainer(
        polygonsState: polygonsState,
        contentPaddingState: contentPaddingState
    )
}

class MapSwiftUIViewContainer: MapWithSwiftUIViewFactory {

    var viewController: UIViewController

    init(
      polygonsState: SkieSwiftStateFlow<[shared.Polygon]>,
      contentPaddingState: SkieSwiftStateFlow<Foundation_layoutPaddingValues
    ) {
        self.viewController = UIHostingController(rootView: SwiftUIMap(polygonsState: polygonsState, contentPaddingState: contentPaddingState))
    }
}
```

```
struct SwiftUIMap: View {

    var polygonsState: SkieSwiftStateFlow<[shared.Polygon]>
    var contentPaddingState: SkieSwiftStateFlow<Foundation_layoutPaddingValues>

    init(
        polygonsState: SkieSwiftStateFlow<[shared.Polygon]>,
        contentPaddingState: SkieSwiftStateFlow<Foundation_layoutPaddingValues>
    ) {
        self.polygonsState = polygonsState
        self.contentPaddingState = contentPaddingState
    }

    @State var polygons: Polygons?
    @State var contentPadding: Foundation_layoutPaddingValues?
    @State var viewport = Viewport.camera(center: .init(latitude: 48.2082, longitude: 16.3719), zoom: 14, bearing: 0, pitch: 0)
```

Native UI (iOS)

```
func mapWithSwiftUIViewFactory(
    polygonsState: SkieSwiftStateFlow<[shared.Polygon]>,
    contentPaddingState: SkieSwiftStateFlow<Foundation_layoutPaddingValues>
) -> MapWithSwiftUIViewFactory {
    return MapSwiftUIViewContainer(
        polygonsState: polygonsState,
        contentPaddingState: contentPaddingState
    )
}

class MapSwiftUIViewContainer: MapWithSwiftUIViewFactory {
```

```
struct SwiftUIMap: View {

    var polygonsState: SkieSwiftStateFlow<[shared.Polygon]>
    var contentPaddingState: SkieSwiftStateFlow<Foundation_layoutPaddingValues>

    init(
        polygonsState: SkieSwiftStateFlow<[shared.Polygon]>,
        contentPaddingState: SkieSwiftStateFlow<Foundation_layoutPaddingValues>
    ) {
        self.polygonsState = polygonsState
        self.contentPaddingState = contentPaddingState
    }

    @State var polygons: Polygons?
    @State var contentPadding: Foundation_layoutPaddingValues?
    @State var viewport = Viewport.camera(center: .init(latitude: 48.2082, longitude: 16.3719), zoom: 14, bearing: 0, pitch: 0)
```

Native UI (iOS)

```
func mapWithSwiftUIViewFactory(
    polygonsState: SkieSwiftStateFlow<[shared.Polygon]>,
    contentPaddingState: SkieSwiftStateFlow<Foundation_layoutPaddingValues>
) -> MapWithSwiftUIViewFactory {
    return MapSwiftUIViewContainer(
        polygonsState: polygonsState,
        contentPaddingState: contentPaddingState
    )
}

class MapSwiftUIViewContainer: MapWithSwiftUIViewFactory {

    var viewController: UIViewController

    init(
      polygonsState: SkieSwiftStateFlow<[shared.Polygon]>,
      contentPaddingState: SkieSwiftStateFlow<Foundation_layoutPaddingValues
    ) {
        self.viewController = UIHostingController(rootView: SwiftUIMap(polygonsState: polygonsState, contentPaddingState: contentPaddingState))
    }
}
```

```
struct SwiftUIMap: View {

    var polygonsState: SkieSwiftStateFlow<[shared.Polygon]>
    var contentPaddingState: SkieSwiftStateFlow<Foundation_layoutPaddingValues>

    init(
        polygonsState: SkieSwiftStateFlow<[shared.Polygon]>,
        contentPaddingState: SkieSwiftStateFlow<Foundation_layoutPaddingValues>
    ) {
        self.polygonsState = polygonsState
        self.contentPaddingState = contentPaddingState
    }

    @State var polygons: Polygons?
    @State var contentPadding: Foundation_layoutPaddingValues?
    @State var viewport = Viewport.camera(center: .init(latitude: 48.2082, longitude: 16.3719), zoom: 14, bearing: 0, pitch: 0)
```

Native UI (iOS)

```
func mapWithSwiftUIViewFactory(
    polygonsState: SkieSwiftStateFlow<[shared.Polygon]>,
    contentPaddingState: SkieSwiftStateFlow<Foundation_layoutPaddingValues>
) -> MapWithSwiftUIViewFactory {
    return MapSwiftUIViewContainer(
        polygonsState: polygonsState,
        contentPaddingState: contentPaddingState
    )
}

class MapSwiftUIViewContainer: MapWithSwiftUIViewFactory {

    var viewController: UIViewController

    init(
      polygonsState: SkieSwiftStateFlow<[shared.Polygon]>,
      contentPaddingState: SkieSwiftStateFlow<Foundation_layoutPaddingValues
    ) {
        self.viewController = UIHostingController(rootView: SwiftUIMap(polygonsState: polygonsState, contentPaddingState: contentPaddingState))
    }
}

struct SwiftUIMap: View {

    var polygonsState: SkieSwiftStateFlow<[shared.Polygon]>
    var contentPaddingState: SkieSwiftStateFlow<Foundation_layoutPaddingValues>

    init(
        polygonsState: SkieSwiftStateFlow<[shared.Polygon]>,
        contentPaddingState: SkieSwiftStateFlow<Foundation_layoutPaddingValues>
    ) {
        self.polygonsState = polygonsState
        self.contentPaddingState = contentPaddingState
    }

    @State var polygons: Polygons?
    @State var contentPadding: Foundation_layoutPaddingValues?
    @State var viewport = Viewport.camera(center: .init(latitude: 48.2082, longitude: 16.3719), zoom: 14, bearing: 0, pitch: 0)
```

Native UI (iOS)

```
    @State var polygons: Polygons?
    @State var contentPadding: Foundation_layoutPaddingValues?
    @State var viewport = Viewport.camera(center: .init(latitude: 48.2082, longitude: 16.3719), zoom: 14, bearing: 0, pitch: 0)

    var body: some View {
        Map(viewport: $viewport) {
            ForEvery($polygons.wrappedValue?.polygons ?? [], id: \.self) { polygon in
                ...
                PolygonAnnotation(polygon: poly)
            }
        }
        .ornamentOptions(
            OrnamentOptions(
                logo: LogoViewOptions(margins: CGPoint(x: CGFloat(8.0), y: CGFloat($contentPadd...))),
                attributionButton: AttributionButtonOptions(margins: CGPoint(x: CGFloat(8.0), y: CGFloat($contentPadd...)))
            )
        )
        .task {
            for await polygons in polygonsState {
                ...
                self.polygons = Polygons(polygons: polygons)
            }
        }
        .task {
            for await contentPadding in contentPaddingState { self.contentPadding = contentPadding }
        }
    }
}
```

Native UI (iOS)

```
    @State var polygons: Polygons?
    @State var contentPadding: Foundation_layoutPaddingValues?
    @State var viewport = Viewport.camera(center: .init(latitude: 48.2082, longitude: 16.3719), zoom: 14, bearing: 0, pitch: 0)

    var body: some View {
        Map(viewport: $viewport) {
            ForEvery($polygons.wrappedValue?.polygons ?? [], id: \.self) { polygon in
                ...
                PolygonAnnotation(polygon: poly)
            }
        }
        .ornamentOptions(
            OrnamentOptions(
                logo: LogoViewOptions(margins: CGPoint(x: CGFloat(8.0), y: CGFloat($contentPadd...))),
                attributionButton: AttributionButtonOptions(margins: CGPoint(x: CGFloat(8.0), y: CGFloat($contentPadd...)))
            )
        )
        .task {
            for await polygons in polygonsState {
                ...
                self.polygons = Polygons(polygons: polygons)
            }
        }
        .task {
            for await contentPadding in contentPaddingState { self.contentPadding = contentPadding }
        }
    }
}
```

Native UI (iOS)

```
@State var polygons: Polygons?
@State var contentPadding: Foundation_layoutPaddingValues?
@State var viewport = Viewport.camera(center: .init(latitude: 48.2082, longitude: 16.3719)
```

```
                ...
                PolygonAnnotation(polygon: poly)
            }
        }
        .ornamentOptions(
            OrnamentOptions(
                logo: LogoViewOptions(margins: CGPoint(x: CGFloat(8.0), y: CGFloat($contentPadd...))),
                attributionButton: AttributionButtonOptions(margins: CGPoint(x: CGFloat(8.0), y: CGFloat($contentPadd...)))
            )
        )
        .task {
            for await polygons in polygonsState {
                ...
                self.polygons = Polygons(polygons: polygons)
            }
        }
        .task {
            for await contentPadding in contentPaddingState { self.contentPadding = contentPadding }
        }
    }
}
```

Native UI (iOS)

```
    @State var polygons: Polygons?
    @State var contentPadding: Foundation_layoutPaddingValues?
    @State var viewport = Viewport.camera(center: .init(latitude: 48.2082, longitude: 16.3719), zoom: 14, bearing: 0, pitch: 0)

    var body: some View {
        Map(viewport: $viewport) {
            ForEvery($polygons.wrappedValue?.polygons ?? [], id: \.self) { polygon in
                ...
                PolygonAnnotation(polygon: poly)
            }
        }
        .ornamentOptions(
            OrnamentOptions(
                logo: LogoViewOptions(margins: CGPoint(x: CGFloat(8.0), y: CGFloat($contentPadd...))),
                attributionButton: AttributionButtonOptions(margins: CGPoint(x: CGFloat(8.0), y: CGFloat($contentPadd...)))
            )
        )
        .task {
            for await polygons in polygonsState {
                ...
                self.polygons = Polygons(polygons: polygons)
            }
        }
        .task {
            for await contentPadding in contentPaddingState { self.contentPadding = contentPadding }
        }
    }
}
```

Native UI (iOS)

```
    @State var polygons: Polygons?
    @State var contentPadding: Foundation_layoutPaddingValues?
    @State var viewport = Viewport.camera(center: .init(latitude: 48.2082, longitude: 16.3719), zoom: 14, bearing: 0, pitch: 0)

    var body: some View {
        Map(viewport: $viewport) {
            ForEvery($polygons.wrappedValue?.polygons ?? [], id: \.self) { polygon in
                ...
                PolygonAnnotation(polygon: poly)
            }
        }
        .ornamentOptions(
            OrnamentOptions(
                logo: LogoViewOptions(margins: CGPoint(x: CGFloat(8.0), y: CGFloat($contentPadd...))),
                attributionButton: AttributionButtonOptions(margins: CGPoint(x: CGFloat(8.0), y: CGFloat($contentPadd...)))
            )
        )
        .task {
            for await polygons in polygonsState {
                ...
                self.polygons = Polygons(polygons: polygons)
            }
        }
        .task {
            for await contentPadding in contentPaddingState { self.contentPadding = contentPadding }
        }
    }
}
```

Native UI (iOS)

```
@State var polygons: Polygons?
@State var contentPadding: Foundation_layoutPaddingValues?
@State var viewport = Viewport.camera(center: .init(latitude: 48.2082, longitude: 16.3719), zoom: 14, bearing: 0, pitch: 0)

var body: some View {
    Map(viewport: $viewport) {
        ForEvery($polygons.wrappedValue?.polygons ?? [], id: \.self) { polygon in
            ...
            PolygonAnnotation(polygon: poly)
        }
    }
    .ornamentOptions(
        OrnamentOptions(
            logo: LogoViewOptions(margins: CGPoint(x: CGFloat(8.0), y: CGFloat($contentPadd...))),
            attributionButton: AttributionButtonOptions(margins: CGPoint(x: CGFloat(8.0), y: CGFloat($contentPadd...)))
        )
    )
```

```
.task {
    for await polygons in polygonsState {
        ...
        self.polygons = Polygons(polygons: polygons)
    }
}
.task {
    for await contentPadding in contentPaddingState { self.contentPadding = contentPadding }
}
```

Native UI (iOS)

```
    @State var polygons: Polygons?
    @State var contentPadding: Foundation_layoutPaddingValues?
    @State var viewport = Viewport.camera(center: .init(latitude: 48.2082, longitude: 16.3719), zoom: 14, bearing: 0, pitch: 0)

    var body: some View {
        Map(viewport: $viewport) {
            ForEvery($polygons.wrappedValue?.polygons ?? [], id: \.self) { polygon in
                ...
                PolygonAnnotation(polygon: poly)
            }
        }
        .ornamentOptions(
            OrnamentOptions(
                logo: LogoViewOptions(margins: CGPoint(x: CGFloat(8.0), y: CGFloat($contentPadd...))),
                attributionButton: AttributionButtonOptions(margins: CGPoint(x: CGFloat(8.0), y: CGFloat($contentPadd...)))
            )
        )
        .task {
            for await polygons in polygonsState {
                ...
                self.polygons = Polygons(polygons: polygons)
            }
        }
        .task {
            for await contentPadding in contentPaddingState { self.contentPadding = contentPadding }
        }
    }
}
```

Native UI (iOS)

```
    @State var polygons: Polygons?
    @State var contentPadding: Foundation_layoutPaddingValues?
    @State var viewport = Viewport.camera(center: .init(latitude: 48.2082, longitude: 16.3719), zoom: 14, bearing: 0, pitch: 0)

    var body: some View {
        Map(viewport: $viewport) {
            ForEvery($polygons.wrappedValue?.polygons ?? [], id: \.self) { polygon in
                ...
                PolygonAnnotation(polygon: poly)
            }
        }
        .ornamentOptions(
            OrnamentOptions(
                logo: LogoViewOptions(margins: CGPoint(x: CGFloat(8.0), y: CGFloat($contentPadd...))),
                attributionButton: AttributionButtonOptions(margins: CGPoint(x: CGFloat(8.0), y: CGFloat($contentPadd...)))
            )
        )
        .task {
            for await polygons in polygonsState {
                ...
                self.polygons = Polygons(polygons: polygons)
            }
        }
        .task {
            for await contentPadding in contentPaddingState { self.contentPadding = contentPadding }
        }
    }
}
```

Native UI (iOS)

```
    @State var polygons: Polygons?
    @State var contentPadding: Foundation_layoutPaddingValues?
    @State var viewport = Viewport.camera(center: .init(latitude: 48.2082, longitude: 16.3719), zoom: 14, bearing: 0, pitch: 0)

    var body: some View {
```

```
Map(viewport: $viewport) {
    ForEvery($polygons.wrappedValue?.polygons ?? [], id: \.self) { polygon in
        ...
        PolygonAnnotation(polygon: poly)
    }
}
```

```
            for await polygons in polygonsState {
                ...
                self.polygons = Polygons(polygons: polygons)
            }
        }
        .task {
            for await contentPadding in contentPaddingState { self.contentPadding = contentPadding }
        }
    }
}
```

Native UI (iOS)

```
    @State var polygons: Polygons?
    @State var contentPadding: Foundation_layoutPaddingValues?
    @State var viewport = Viewport.camera(center: .init(latitude: 48.2082, longitude: 16.3719), zoom: 14, bearing: 0, pitch: 0)

    var body: some View {
        Map(viewport: $viewport) {
            ForEvery($polygons.wrappedValue?.polygons ?? [], id: \.self) { polygon in
                ...
                PolygonAnnotation(polygon: poly)
            }
        }
        .ornamentOptions(
            OrnamentOptions(
                logo: LogoViewOptions(margins: CGPoint(x: CGFloat(8.0), y: CGFloat($contentPadd...))),
                attributionButton: AttributionButtonOptions(margins: CGPoint(x: CGFloat(8.0), y: CGFloat($contentPadd...)))
            )
        )
        .task {
            for await polygons in polygonsState {
                ...
                self.polygons = Polygons(polygons: polygons)
            }
        }
        .task {
            for await contentPadding in contentPaddingState { self.contentPadding = contentPadding }
        }
    }
}
```

Native UI (iOS)

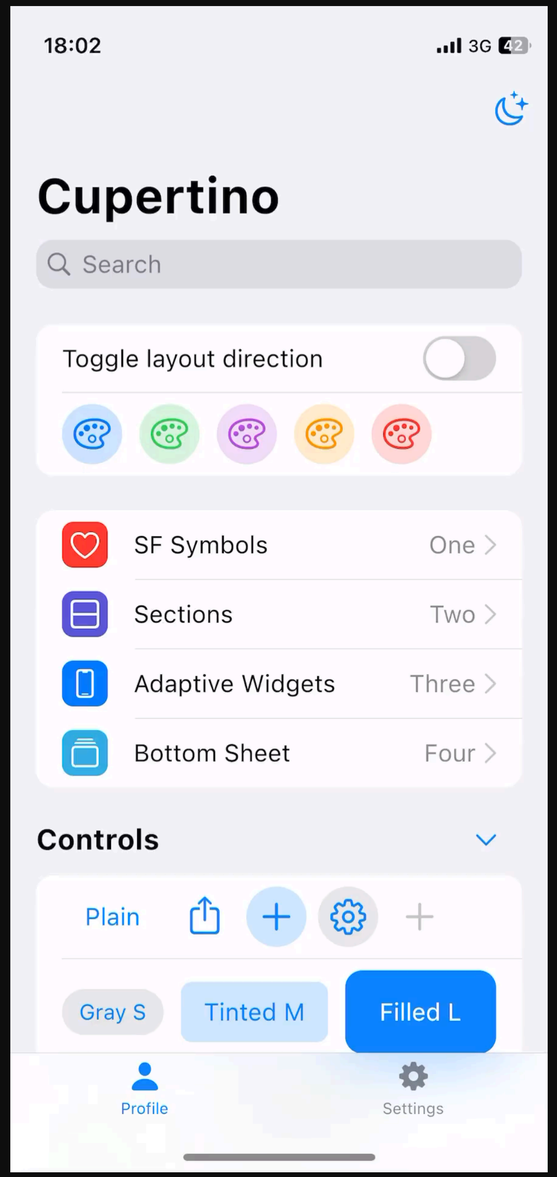
```
fun MapViewController(
    mapFactory: (viewModel: MapViewModel) -> UIViewController
) = ComposeUIViewController {
    val viewModel: MapViewModel = // resolve the ViewModel
    UIKitViewController(
        modifier = Modifier,
        factory = { mapFactory(viewModel) }, // Pass the ViewModel on instantiation
        update = {}
    )
}
```

```
private func myMapViewFactory(viewModel: MapViewModel) -> UIViewController {
      let myMapView = MyMapView(viewModel: viewModel) // Instantiate the SwiftUI View
      return UIHostingController(rootView: myMapView) // Use the view to construct a ViewController
  }
```

https://touchlab.co/jetpack-compose-ios-interop

Real Native UI (iOS)

Real Native UI (iOS)



Compose Cupertino

https://github.com/alexzhirkevich/compose-cupertino

The current state

The current state

- IDE decision
  - Android Studio, Fleet and/or Xcode
- You will (always?) be 2-3 months behind Android’s compose
  - Google’s willingness for KMP reflects on Compose Multiplatform
- JetBrains is going full speed on Compose Multiplatform and Fleet
- There is currently no “perfect” way to have Native UI within your app
- Compose Multiplatform 1.7.0 in alpha

How to get your iOS colleagues on board?

How to get your iOS colleagues on board?

- Kotlin isn’t as far off from Swift
- Compose ≈ SwiftUI (but backwards compatible)
- Native code will not go away
- New iOS features can receive more attention

Keeping up to date

Keeping up to date

- Websites to bookmark
  https://blog.jetbrains.com/kotlin/category/multiplatform/
  https://github.com/JetBrains/compose-multiplatform/releases
  https://www.youtube.com/c/kotlin

- People to follow

  https://berlin.droidcon.com/speakers/
  https://kotlinconf.com/speakers/

- Preferring to chat?
  https://kotlinlang.org/community/
  https://slack-chats.kotlinlang.org/

- Looking for some libraries?
  https://github.com/terrakok/kmp-awesome
  https://klibs.io/

Thank you

Thank you

Sample

https://github.com/rschattauer/compose_multiplatform

Thank you

GitHub

Sample

https://github.com/rschattauer/compose_multiplatform