Toggly Swift / iOS

iOS, macOS, tvOS, watchOS · SwiftUI, UIKit, Combine, async/await

Install (SwiftPM)

Use the published `ios-sdk-v1.8.3` tag or later (root package manifest). Earlier tags without that manifest cannot install from the repo URL. See docs.toggly.io/sdks/ios#installation.

PACKAGE.SWIFT

```
dependencies: [
  .package(
    url: "https://github.com/ops-ai/" +
         "Toggly.FeatureManagement.git",
    revision: "ios-sdk-v1.8.3")
]

.target(name: "App", dependencies: [
  .product(name: "TogglyCore",
           package: "Toggly.FeatureManagement"),
  .product(name: "TogglySwiftUI",
           package: "Toggly.FeatureManagement"),
  .product(name: "TogglyUIKit",
           package: "Toggly.FeatureManagement"),
  .product(name: "TogglyCombine",
           package: "Toggly.FeatureManagement"),
])
```

Configure

```
import TogglyCore

@main
struct MyApp: App {
  init() {
    Toggly.configure(config: TogglyConfig(
      appKey:      Env.togglyKey,
      environment: "Production",
      identity:    nil,                  // anonymous
      featureDefaults: [
        "NewCheckout": false,
        "BetaUI":      false,
      ],
      refreshInterval: 180,
      useSignedDefinitions: true,
      connectTimeout: 5, requestTimeout: 30
    ))
    Task { await Toggly.shared.initialize() }
  }
  var body: some Scene {
    WindowGroup { ContentView() } } }
```

Evaluate (async/await)

```
if await Toggly.shared.isFeatureOn("NewUI") {
  // ...
}

let on = await Toggly.shared.evaluateFeatureGate(
  featureKeys: ["a","b"],
  requirement: .all,
  negate: false)
```

Variants

```
// TogglyConfig(enableVariants: true, ...)
if let v = await Toggly.shared
    .getVariant("checkout-experiment") {
  switch v.name {
  case "Treatment": showTreatment()
  default: showControl()
  }
}
// @FeatureVariant("key") var variant // SwiftUI
```

Identity & rollouts

```
// login
await Toggly.shared.setIdentity("user-123")

// logout - reverts to anonymous device id
await Toggly.shared.setIdentity(nil)

// manual refresh
await Toggly.shared.refresh()
```

SwiftUI — property wrapper

```
import TogglySwiftUI

struct ContentView: View {
  @FeatureFlag("NewUI")          var newUI
  @FeatureFlag("dark", defaultValue: true) var dark
  @FeatureGate(["a","b"], requirement: .all)
                                 var bothOn

  var body: some View {
    if newUI { NewView() }
    if bothOn { CombinedView() }
  }
}
```

SwiftUI — modifiers & view

```
Button("Beta") { print("Beta tapped") }
  .featureFlag("Beta")          // hide if off

Text("Old")
  .featureFlag("LegacyMode", negate: true) // hide if

FeatureView("dark-mode") {
  DarkModeSettings()
}
FeatureView("dark-mode", negate: true) {
  Text("Coming soon!")
}
```

UIKit

```
import TogglyUIKit

class HomeVC: FeatureFlagViewController {
  override func viewDidLoad() {
    super.viewDidLoad()
    observeFeature("NewUI")
  }
  override func featureFlagDidChange(
      _ key: String, isEnabled: Bool) {
    newButton.isHidden = !isEnabled
  }
}

// or one-shot
Task {
  if await FeatureFlagAsync.isEnabled("NewUI") {
    showNewUI()
  }
}
```

Combine

```
import TogglyCombine

class VM: ObservableObject {
  @Published var newUI = false
  var bag = Set<AnyCancellable>()

  init() {
    TogglyPublishers.featureFlag("NewUI")
      .assign(to: &$newUI)
    TogglyPublishers.featureGate(
        ["a","b"], requirement: .any)
      .sink { [weak self] in
        self?.handle($0) }.store(in: &bag)
  }
}
```

Toggly Swift / iOS

iOS, macOS, tvOS, watchOS · SwiftUI, UIKit, Combine, async/await

Events

```
await Toggly.shared.on { event in
  switch event {
  case .featureChanged(let c):
    print("\(c.featureKey): \(c.newValue)")
  case .definitionsRefreshed:
    print("flags refreshed")
  default: break
  }
}
```

Custom storage

```
actor KeychainStorage: TogglyStorage {
  func get(_ key: String) async → String? { … }
  func set(_ key: String,
           value: String) async { … }
  func delete(_ key: String) async { … }
  func clear() async { … }
}

Toggly.configure(config: TogglyConfig(
  appKey: KEY, storage: KeychainStorage()))
```

Built-in: `MemoryStorage` (default), `UserDefaultsStorage`.

Signed definitions

```
Toggly.configure(config: TogglyConfig(
  appKey: KEY, environment: "Production",
  useSignedDefinitions: true))
```

ECDSA P-256 / ES256 over `data|timestamp`; JWKS cached up to 30 days; key rotation handled automatically.

Testing

```
Toggly.reset()                      // between tests
Toggly.configure(config: TogglyConfig(
  featureDefaults: [
    "feature1": true,
    "feature2": false,
  ]))
await Toggly.shared.initialize()
```

Debug info

```
let d = await Toggly.shared.getDebugInfo()
print("identity: \(d.identity)")
print("lastSynced:  \(d.lastSynced)")
print("lastError:   \(d.lastError)")
```

App lifecycle

- Foreground refreshes flags automatically.
- Background suspends the refresh timer.
- ETag-aware — 304 Not Modified responses keep cache hot.
- Per-identity cache survives app launches.

Pitfalls

- Call `Toggly.shared.initialize()` from `App.init` in a `Task`; SwiftUI views can render before flags load — use `showFeatureDuringEvaluation` or `featureDefaults` to avoid flicker.
- `@FeatureFlag` is reactive — it subscribes to flag state changes automatically.
- On watchOS, set a longer `refreshInterval` and rely on cached flags — bandwidth is metered.
- Don't instantiate multiple `TogglyConfig`s — use `Toggly.configure` once, then `Toggly.shared`.
- Pin App Key out of bundle — read from `xcconfig` / build settings, never hard-code in code reviews.

Resources

- Docs: `docs.toggly.io/sdks/ios`
- SwiftPM: `Toggly.FeatureManagement`
- GitHub: `github.com/ops-ai/Toggly.FeatureManagement`
- iOS 14+, macOS 11+, tvOS 14+, watchOS 7+