FLOWX.AI

PLATFORM DEEP DIVE / Core components / Renderer SDKs / ios-renderer

Contents

PLATFORM DEEP DIVE / Core components / Renderer SDKs / Using the iOS Renderer

iOS Project Requirements

The minimum requirements are:

- iOS 14
- Swift 5.0

Installing the library

The iOS Renderer is available through Cocoapods and Swift Package Manager.

Swift Package Manager

In Xcode, click `File` → `Add Packages...`, enter FlowX repo's URL `https://github.com/flowx-ai/flowx-ios-sdk`. Set the dependency rule to `Up To Next Major` and add package.

If you are developing a framework and use FlowX as a dependency, add to your `Package.swift` file:

```
dependencies: [
    .package(url: "https://github.com/flowx-ai/flowx-ios-sdk", .upToNextMajor(from: "0.96.0"))
]
```

Cocoapods

Prerequisites

- Cocoapods gem installed

Cocoapods private trunk setup

Add the private trunk repo to your local Cocoapods installation with the command:

```
pod repo add flowx-specs git@github.com:flowx-ai/flowx-ios-specs.git
```

Adding the dependency

Add the source of the private repository in the Podfile

```
source 'git@github.com/flowx-ios-specs.git'
```

Add the pod and then run `pod install`

```
pod 'FlowX'
```

Library dependencies

The iOS Renderer library depends on the following libraries:

- Socket.IO-Client-Swift
- Alamofire
- SVProgressHUD
- SDWebImageSwiftUI

Configuring the library

The SDK has 2 configurations, available through shared instances.

It is recommended to call the configuration methods at app launch.

Otherwise, make sure you do it before the start of any FlowX process.

FXConfig

This config is used for general purpose properties.

Properties

Name	Description	Type	Requirement
baseURL	The base URL used for REST networking	String	Mandatory
imageBaseURL	The base URL used for media library images	String	Mandatory

Name	Description	Type	Requirement
language	The language used for retrieving enumerations and substitution tags	String	Mandatory. Defaults to "en"
stepViewType	The type of the custom step view class	FXStepViewProtocol.Type	Optional
logEnabled	Value indicating whether console logging is enabled. Default is false	Bool	Optional

Sample

```
FXConfig.sharedInstance.configure { (config) in
    config.baseURL = myBaseURL
```

```
    config.imageBaseURL = myImageBaseURL
    config.language = "en"
    config.logEnabled = true
    config.stepViewType = CustomStepView.self
}
```

FXSessionConfig

This config is used for providing networking or auth session-specific properties.

The library expects a session instance managed by the container app. Request adapting and retrying are handled by the container app.

Properties

Name	Description	Type
sessionManager	Alamofire session instance used for REST networking	Session
token	JWT authentication access token	String

Sample

```
FXSessionConfig.sharedInstance.configure { config in
    config.sessionManager = mySessionManager
    config.token = myAccessToken
}
```

Using the library

The library's public APIs are called using the shared instance of FlowX, `FlowX.sharedInstance`.

How to start and end FlowX session

After all the configurations are set, you can start a FlowX session by calling the `startSession()` method.

This is optional, as the session starts lazily when the first process is started.

`FlowX.sharedInstance.startSession()`

When you want to end a FlowX session, you can call the `endSession()` method. This also does a complete clean-up of the started processes.

You might want to use this method in a variety of scenarios, for instance when the user logs out.

`FlowX.sharedInstance.endSession()`

How to start a process

You can start a process by calling the method below.

The container app is responsible with presenting the navigation controller holding the process navigation.

```
public func startProcess(navigationController:
UINavigationController,
```

```
                            name: String,
                            params: [String: Any]?,
                            isModal: Bool = false,
                            showLoader: Bool = false)
```

`navigationController` - the instance of UINavigationController which will hold the process navigation stack

`name` - the name of the process

`params` - the start parameters, if any

`isModal` - a boolean indicating whether the process navigation is modally displayed. When the process navigation is displayed modally, a close bar button item is displayed on each screen displayed throughout the process navigation.

`showLoader` - a boolean indicating whether the loader should be displayed when starting the process.

Sample

```
FlowX.sharedInstance.startProcess(navigationController:
processNavigationController,
                                  name: processName,
                                  params: startParams,
                                  isModal: true
                                  showLoader: true)

self.present(processNavigationController, animated: true,
completion: nil)
```

How to resume a process

You can resume a process by calling the method below.

```
public func continueExistingProcess(uuid: String,
                                    name: String,
                                    navigationController:
UINavigationController,
                                    isModal: Bool = false) {
```

`uuid` - the UUID string of the process

`name` - the name of the process

`navigationController` - the instance of UINavigationController which will hold the process navigation stack

`isModal` - a boolean indicating whether the process navigation is modally displayed. When the process navigation is displayed modally, a close bar button item is displayed on each screen displayed throughout the process navigation.

How to end a process

You can manually end a process by calling the `stopProcess(name: String)` method.

This is useful when you want to explicitly ask the FlowX shared instance to clean up the instance of the process sent as parameter.

For example, it could be used for modally displayed processes that are dismissed by the user, in which case the `dismissRequested(forProcess process:`

`String, navigationController: UINavigationController)` method of the FXDataSource will be called.

Sample

```
FlowX.sharedInstance.stopProcess(name: processName)
```

How to run an action from a custom component

The custom components which the container app provides will contain FlowX actions to be executed. In order to run an action you need to call the following method:

```
public func runAction(action: ProcessActionModel,
                      params: [String: Any]? = nil)
```

`action` - the `ProcessActionModel` action object

`params` - the parameters for the action

How to run an upload action from a custom component

```
public func runUploadAction(action: ProcessActionModel,
                            image: UIImage)
```

`action` - the `ProcessActionModel` action object

`image` - the image to upload

```
public func runUploadAction(action: ProcessActionModel,
                            fileURL: URL)
```

`action` - the `ProcessActionModel` action object

`fileURL` - the local URL of the image

Getting a substitution tag value by key

```
public func getTag(withKey key: String) -> String?
```

All substitution tags will be retrieved by the SDK before starting the first process and will be stored in memory.

Whenever the container app needs a substitution tag value for populating the UI of the custom components, it can request the substitution tag using the method above, providing the key.

Getting a media item url by key

```
public func getMediaItemURL(withKey key: String) -> String?
```

All media items will be retrieved by the SDK before starting the first process and will be stored in memory.

Whenever the container app needs a media item url for populating the UI of the custom components, it can request the url using the method above, providing the key.

```
public func getTag(withKey key: String) -> String?
```

All substitution tags will be retrieved by the SDK before starting the first process and will be stored in memory.

Whenever the container app needs a substitution tag value for populating the UI of the custom components, it can request the substitution tag using the method above, providing the key.

Handling authorization token changes

When the access token of the auth session changes, you can update it in the renderer using the `func updateAuthorization(token: String)` method.

FXDataSource

The library offers a way of communication with the container app through the `FXDataSource` protocol.

The data source is a public property of FlowX shared instance.

`public weak var dataSource: FXDataSource?`

```
public protocol FXDataSource: AnyObject {
    func controllerFor(componentIdentifier: String) ->
FXController?

    func viewFor(componentIdentifier: String) -> FXView?

    func viewFor(componentIdentifier: String,
```

```
customComponentViewModel: FXCustomComponentViewModel) ->
AnyView?

    func navigationController() -> UINavigationController?

    func errorReceivedForAction(name: String?)

    func validate(validatorName: String, value: String) ->
Bool

    func dismissRequested(forProcess process: String,
navigationController: UINavigationController)

}
```

- `func controllerFor(componentIdentifier: String) -> FXController?`

This method is used for providing a custom component UIKit view controller, identified by the componentIdentifier argument.

- `func viewFor(componentIdentifier: String) -> FXView?`

This method is used for providing a custom UIKit view, identified by the componentIdentifier argument.

- `func viewFor(componentIdentifier: String, customComponentViewModel: FXCustomComponentViewModel) -> AnyView?`

This method is used for providing a custom SwiftUI view, identified by the componentIdentifier argument. A view model is provided as an ObservableObject

to be added as @ObservedObject inside the SwiftUI view.

- `func navigationController() -> UINavigationController?`

This method is used for providing a navigation controller. It can be either a custom `UINavigationController` class, or just a regular `UINavigationController` instance themed by the container app.

- `func errorReceivedForAction(name: String?)`

This method is called when an error occurs after an action is executed.

- `func validate(validatorName: String, value: String) -> Bool`

This method is used for custom validators. It provides the name of the validator and the value to be validated. The method returns a boolean indicating whether the value is valid or not.

- `func dismissRequested(forProcess process: String, navigationController: UINavigationController)`

This method is called, on a modally displayed process navigation, when the user attempts to dismiss the modal navigation. Typically it is used when you want to present a confirmation pop-up.

The container app is responsible with dismissing the UI and calling the stop process APIs.

FXController

FXController is an open class, which helps the container app provide UIKit custom component screens to the renderer. It needs to be subclassed for each custom screen.

```
open class FXController: UIViewController {

    internal(set) public var data: [String: Any]?
    internal(set) public var actions: [ProcessActionModel]?

    open func titleForScreen() -> String? {
        return nil
    }

    open func populateUI(data: [String: Any]) {

    }

    open func updateUI(data: [String: Any]) {

    }

}
```

- `internal(set) public var data: [String: Any]?`

`data` is a dictionary property, containing the data model for the custom component.

- `internal(set) public var actions: [ProcessActionModel]?`

`actions` is the array of actions provided to the custom component.

- `func titleForScreen() -> String?`

This method is used for setting the screen title. It is called by the renderer when the view controller is displayed.

- `func populateUI(data: [String: Any])`

This method is called by the renderer, after the controller has been presented, when the data is available.

This will happen asynchronously. It is the container app's responsibility to make sure that the initial state of the view controller does not have default/residual values displayed.

- `func updateUI(data: [String: Any])`

This method is called by the renderer when an already displayed view controller needs to update the data shown.

FXView

FXView is a protocol that helps the container app provide custom UIKit subviews of a generated screen to the renderer. It needs to be implemented by `UIView` instances. Similar to `FXController` it has data and actions properties and a populate method.

```
public protocol FXView: UIView {
    var data: [String: Any]? { get set }
    var actions: [ProcessActionModel]? { get set }
```

```
    func populateUI(data: [String: Any]?)
}
```

- `var data: [String: Any]?`

`data` is a dictionary property containing the data model needed by the custom view.

- `var actions: [ProcessActionModel]?`

`actions` is the array of actions provided to the custom view.

- `func populateUI(data: [String: Any]?)`

This method is called by the renderer after the screen containing the view has been displayed.

It is the container app's responsibility to make sure that the initial state of the view does not have default/residual values displayed.

NOTE: It is mandatory for views implementing the FXView protocol to provide the intrinsic content size. Sample:

```
override var intrinsicContentSize: CGSize {
    return CGSize(width: UIScreen.main.bounds.width, height:
100)
}
```

FXCustomComponentViewModel

`FXCustomComponentViewModel` is a class implementing the `ObservableObject` protocol. It is used for managing the state of custom SwiftUI views. It has two published properties, for data and actions.

```
    @Published public var data: [String: Any] = [:]
    @Published public var actions: [ProcessActionModel] = []
```

Example

```
struct SampleView: View {

    @ObservedObject var viewModel:
FXCustomComponentViewModel

    var body: some View {
        Text("Lorem")
    }
}
```

Was this page helpful?