playAWARDS iOS SDK Usage Guide

The playAWARDS iOS SDK allows access into the Loyalty Center to redeem points earned by a user. The playAWARDS iOS SDK can be integrated into a Unity application or a native iOS application. This technical documentation outlines how to integrate the SDK into an existing iOS application.

Opening and Closing Your Loyalty Center

The SDK module is called `RewardsStoreSDK`. Use the following function to start the store session:

```
StartStore(initData: RewardsStoreInitData)
```

The `RewardsStoreInitData` has a list of properties to initialize. playAWARDS offers a `RewardsStoreInitDataBuilder` builder to help you configure your sessions if working at a native level, or you have access to managed objects.

Note: This code is provided as an example.

```
@objc public class RewardsStoreInitDataBuilder : NSObject {
    /// Required. Constructs the object
    /// - Parameters:
    ///   - loginJwt: Jwt provided by the game (generated by your game server)
    ///   - environmentPtr: String of your environment. Can be DEVELOP, STAGE, or
PROD
    @objc public init(_ loginJwt: String, _ environmentPtr: String)


    /// Required. Sets the Loyalty handler callback of the current init data
instance
    /// - Parameter handler: The delegate, must be a void(double)
    /// - Returns: The builder instance
    @objc public func WithLoyaltyHandler(_ handler: LoyaltyDelegate)


    /// Required. Sets the myVip handler callback of the current init data
instance
    /// - Parameter handler: The delegate, must be a void(char\*)
    /// - Returns: The builder instance
    @objc public func WithMyVipHandler(_ handler: MyVipDelegate)

    /// Required. Sets the anonymous Loyalty Point to show in the store (can be 0
for any authenticated user)
    /// - Parameter anonymousLP: Amount of Loyalty Point for the anonymous user
    /// - Returns: The builder instance
    @objc public func WithAnonymousLP(_ anonymousLP: Int64)

    /// Required. Determines whether or not the preloading is active
    /// - Parameter preload: Whether the store should preload
    /// - Returns: The builder instance
    @objc public func ShouldPreload(_ preload: Bool)

    /// Required. Sets deeplinking properties
    /// - Parameters:
    ///   - type: deeplink type url
    ///   - section: deeplink section url
    ///   - elementId: element id url
    /// - Returns: The builder instance
    @objc public func WithDeepLink(_ type: String?, _ section: String?, _
elementId: String?)

    /// Optional. Sets the language for localization of the store
    /// - Parameter language: String to determine which language to use (defaults
to "EN" if not provided or not a valid language)
    /// - Returns: The builder instance
    @objc public func SetLanguage(_ language: String)

    /// Required. Constructs the RewardsStoreInitData instance
    /// - Returns: The internal instance of RewardsStoreInitData
    @objc public func Build()
}
```

Set up the `RewardsStoreInitDataBuilder` properties and functions one-by-one.

Do the following to open and close your Loyalty Center:

1. Retrieve Your Signing Key From the playAWARDS Game Console
2. Initialize a Store Session With playAWARDS
3. Retrieve Loyalty Point Balance and Tier Status Updates From playAWARDS
4. Send Anonymous Loyalty Points to the Store
5. Preload the Store
6. Set Up the Ability to Deeplink to Locations Within the Loyalty Center
7. Select a Language for the Store
8. Open the Loyalty Center

1. Retrieve Your Signing Key From the playAWARDS Game Console

Your JSON Web Token (JWT) must be signed by a signing key. Storing this key in its integrity is crucial for the handshake to occur. This key should be considered a secret variable and must not be exposed in any public-facing methods within an API nor in the client side of your game.

If you do not have access to the playAWARDS Game Console, request access from PLAYSTUDIOS IT.

Visit the Getting Connected page to access your signing key.

2. Initialize a Store Session With playAWARDS

Set up the following properties:

- `loginJwt: String`: *Required*.

    - Token used to authorize the session of the Rewards Store.

- `environmentPtr: String`: *Required*.

    - This parameter represents the system environment:
        - **STAGING** for Staging.
        - **PROD** for Production.

    Note: STAGING calls the Staging environment. The player database on Staging is different from the player database on Production. Ensure you are using it only for local testing purposes.

Getting a Valid Store Session

To be able to log in to the store, playAWARDS uses a `loginJwt`, a JSON Web Token (JWT). This token must be created on your own game server for security reasons.

The JWT should be generated using the HS256 algorithm and must contain certain specific information within its claims to allow our internal API calls to recognize the user session as valid.

The following table contains the information the token should include:

Claim	Description
`partnerApplicationId`	Your game's unique identifier within the playAWARDS services (provided by playAWARDS), and specific to your environment. This means that you could have a different value for both **STAGING** and **PROD**.
`playerId`	Unique identifier for the specific player instance trying to access the store. Note that this could be an echo of the `supportCode` claim for anonymous users.
`supportCode`	The given support code for this instance of the player (taken as `playerId` for anonymous users).
`identityProvider`	The service that was interfaced with to identify a player, for example, Facebook, Google, Apple, and so forth.
`iat`	Unix Timestamp (EPOCH) for the token's issued date and time.
`exp`	Unix Timestamp (EPOCH) for the token's expiration date and time.

Signing Keys, Environments and Application IDs

In order for your `loginJwt` to be valid, the stated properties should all match and be aware of each other. This means:

- Your signing key must be valid for your environment.

- The `partnerApplicationId` must be deployed in the desired environment.

    Note: If your `partnerApplicationId` contains the words **STAGING** or **PROD**, that does not mean that the `partnerApplicationId` is deployed to that specific environment. The naming is just for your convenience and ease of identification; it does not get tied into the actual referenced environment.

You should now be able to construct the following:

```
let builder = RewardsStoreInitDataBuilder("ey...", "PROD") // loginJWT is String
```

3. Retrieve Loyalty Point Balance and Tier Status Updates From playAWARDS

There are two useful callbacks or handlers you can attach to your functions to receive responses from the SDK when the store is closed or a Reward is purchased. The functions need to be written by the game so that the game knows how to handle the new updated values when they are sent.

- `loyaltyHandler: UnsafePointer<CDouble>`: *Required*.
    - While the player was in the Loyalty Center, their balance might have changed. For example, maybe the player purchased a Reward. Therefore, when the Loyalty Center is closed, playAWARDS must send you the updated Loyalty Point balance. playAWARDS passes this balance to your game's handler, and you can immediately reflect the updated Loyalty Point balance in your game UI.
- `myVipHandler: UnsafePointer<CChar>`: *Required*.
    - While the player was in the Loyalty Center, they might have changed their Loyalty Tier and Tier Point balance. Therefore, when the Loyalty Center is closed, playAWARDS must send you the updated Loyalty Tier and Tier Point balance. playAWARDS passes this to your game's handler, and you can immediately reflect the updated Loyalty Tier and Tier Point balance in your game UI.

Your code should look like the following:

```
let builder = RewardsStoreInitDataBuilder("ey...", "PROD") // loginJWT is String
builder.WithLoyaltyHandler(loyaltyHandlerF)
builder.WithMyVipHandler(vipCallbackF)

func loyaltyHandlerF(_: UnsafePointer<CDouble>){
    // Handle the updated Loyalty Point balance when they are sent.
}
func vipCallbackF(_: UnsafePointer<CChar>){
    // Handle the updated myVIP status when they are sent.
}
```

4. Send Anonymous Loyalty Points to the Store

The purpose of this step is to ensure playAWARDS can properly display the Loyalty Point balance of non-authenticated or anonymous players. If playAWARDS does not have this information, you must provide it.

How you implement this step depends on whether or not your game manages and stores your own anonymous Loyalty Points, or if Loyalty Point storage is done through the playAWARDS Loyalty Engine:

- If you manage anonymous Loyalty Points, and therefore playAWARDS does not have these balances, then you are responsible to pass through the anonymous Loyalty Point balance.

- If you do not manage anonymous Loyalty Points, and playAWARDS has this balance available, then you can simply pass through the value **0**.

  If you have questions about this, reach out to your Integration Manager.

- `anonLP: Int64`: *Required*.
  - Sends playAWARDS the anonymous player's Loyalty Point balance. If the player is authenticated, the value can be **0**.

Your code should look like the following:

```
let builder = RewardsStoreInitDataBuilder("ey...", "PROD") // loginJWT is String
builder.WithLoyaltyHandler(loyaltyHandlerF)
builder.WithMyVipHandler(vipCallbackF)
builder.WithAnonymousLP(100) // Pass an Int64 number if you manage the anonymous
Loyalty Point balance or 0

func loyaltyHandlerF(_: UnsafePointer<CDouble>){
    // Handle the updated Loyalty Point balance when they are sent.
}
func vipCallbackF(_: UnsafePointer<CChar>){
    // Handle the updated myVIP status when they are sent.
}
```

5. Preload the Store

Preload should be called at the start of the game. This is to ensure the game assets are preloaded to render the store to make load time much faster when the store opens.

- `preload: Bool`: *Required*.
  - The value must be **true** to ensure a better UI experience for the player.

Your code should look like the following:

```
let builder = RewardsStoreInitDataBuilder("ey...", "PROD") // loginJWT is String
builder.WithLoyaltyHandler(loyaltyHandlerF)
builder.WithMyVipHandler(vipCallbackF)
builder.WithAnonymousLP(100) // Pass an Int64 number if you manage the anonymous
Loyalty Point balance or 0
builder.ShouldPreload(true) // Must be set to true

func loyaltyHandlerF(_: UnsafePointer<CDouble>){
    // Handle the updated Loyalty Point balance when they are sent.
}
func vipCallbackF(_: UnsafePointer<CChar>){
    // Handle the updated myVIP status when they are sent.
}
```

6. Set Up the Ability to Deeplink to Locations Within the Loyalty Center

playAWARDS offers the deeplink functionality so that the game can take players to a specific Reward, their wallet, their account, and so forth. For more details, see Deeplinking and User Direction.

- `deeplinkType: String`: *Required*.
  - This parameter is for specifying the type of the deeplink. Possible values are `home`, `wallet`, `account`, `award`, or `section`. Use lowercase for these values.
- `deeplinkSection: String`: *Optional*.
  - This parameter indicates the section to which you want to navigate as long as the `deeplinkType` parameter has the value `section`. Possible values are `destination`, `partner`, and so forth. Use lowercase for these values.
- `deeplinkElementId: String`: *Optional*.
  - This parameter indicates the ID of the section or the ID of the Reward to which you want to navigate.

Your code should look like the following if you want to use deeplink:

```
let builder = RewardsStoreInitDataBuilder("ey..."", "PROD") // loginJWT is String
builder.WithLoyaltyHandler(loyaltyHandlerF)
builder.WithMyVipHandler(vipCallbackF)
builder.WithAnonymousLP(100) // Pass an Int64 number if you manage the anonymous
Loyalty Point balance or 0
builder.ShouldPreload(true) // Must be set to true
builder.withDeepLink("section", "destination", "4") // An example of the deeplink
that takes the player to the destination view with ID 4

func loyaltyHandlerF(_: UnsafePointer<CDouble>){
    // Handle the updated Loyalty Point balance when they are sent.
}
func vipCallbackF(_: UnsafePointer<CChar>){
    // Handle the updated myVIP status when they are sent.
}
```

Note: In order to deeplink from a marketing campaign or anywhere outside of your application, you could build the ability, for example, through a Singular link, to accept the deeplink parameters in your application URI. Then, convert the deeplink parameters into the expected playAWARDS deeplink protocol/parameters.

7. Select a Language for the Store

playAWARDS can localize in up to 16 set languages.

Note: This step is optional. If you choose to localize your Loyalty Center, know that there might be a cost to you!

playAWARDS offers language localization for the Rewards Store. For more details, see Language Localization.

- `language: String`: *Optional*.
  - The value must be one of the languages that playAWARDS offers.

Your code should look like the following if you want to use language localization:

```
let builder = RewardsStoreInitDataBuilder("ey...", "PROD") // loginJWT is String
builder.WithLoyaltyHandler(loyaltyHandlerF)
builder.WithMyVipHandler(vipCallbackF)
builder.WithAnonymousLP(100) // Pass an Int64 number if you manage the anonymous
Loyalty Point balance or 0
builder.ShouldPreload(true) // Must be set to true
builder.withDeepLink("section", "destination", "4") // An example of the deeplink
that takes the player to the destination view with ID 4
builder.setLanguage("ja") // Must be one of the languages available

func loyaltyHandlerF(_: UnsafePointer<CDouble>){
    // Handle the updated Loyalty Point balance when they are sent.
}
func vipCallbackF(_: UnsafePointer<CChar>){
    // Handle the updated myVIP status when they are sent.
}
```

8. Open the Loyalty Center

Build the `initData` to call the `StartStore` function:

```
let builder = RewardsStoreInitDataBuilder("ey...", "PROD") // loginJWT is String
builder.WithLoyaltyHandler(loyaltyHandlerF)
builder.WithMyVipHandler(vipCallbackF)
builder.WithAnonymousLP(100) // Pass an Int64 number if you manage the anonymous
Loyalty Point balance or 0
builder.ShouldPreload(true) // Must be set to true
builder.withDeepLink("section", "destination", "4") // An example of the deeplink
that takes the player to the destination view with ID 4
builder.setLanguage("ja") // Must be one of the languages available

let initData = builder.build()

// Initiate the preloading of the store to make load time much faster when the
store opens.
// This should be called at the start of the game.
StartStore(initData)

// Show the store after being preloaded by StartStore.
// Do not call this function before StartStore.
// This should be called when a button or an action is invoked to open the store.
ShowStoreFromPreload("section", "destination", "4", anonLP, "ja")

func loyaltyHandlerF(_: UnsafePointer<CDouble>){
    // Handle the updated Loyalty Point balance when they are sent.
}
func vipCallbackF(_: UnsafePointer<CChar>){
    // Handle the updated myVIP status when they are sent.
}
```

Congratulations! You have successfully integrated the playAWARDS iOS SDK into your application.

Loyalty Center SDK Privacy Manifest

Note: Typically, you merge your privacy manifest with ours. The playAWARDS privacy manifest is currently empty, so there is no work for you to do at this time.

Apple's new development guidelines state the importance of users' privacy and developer transparency, which is why since April 2024, all applications are required to declare their privacy manifest to get approval for the App Store. This is crucial to know since the Loyalty Center SDK makes use of certain user data for Rewards customization, analytics, and to improve the overall user experience inside the game.

Our Privacy Manifest

You can find the **PrivacyInfo.xcprivacy** file at the root of the **RewardsStoreSDK.framework** folder. The file is an XML declaring what information we intend to track and use and with what purpose. Refer to this file for any questions regarding the store's data tracking needs as well as if any manual merging is required with your own application's **.xcprivacy** file.