Swift Wallet SDK for Stellar

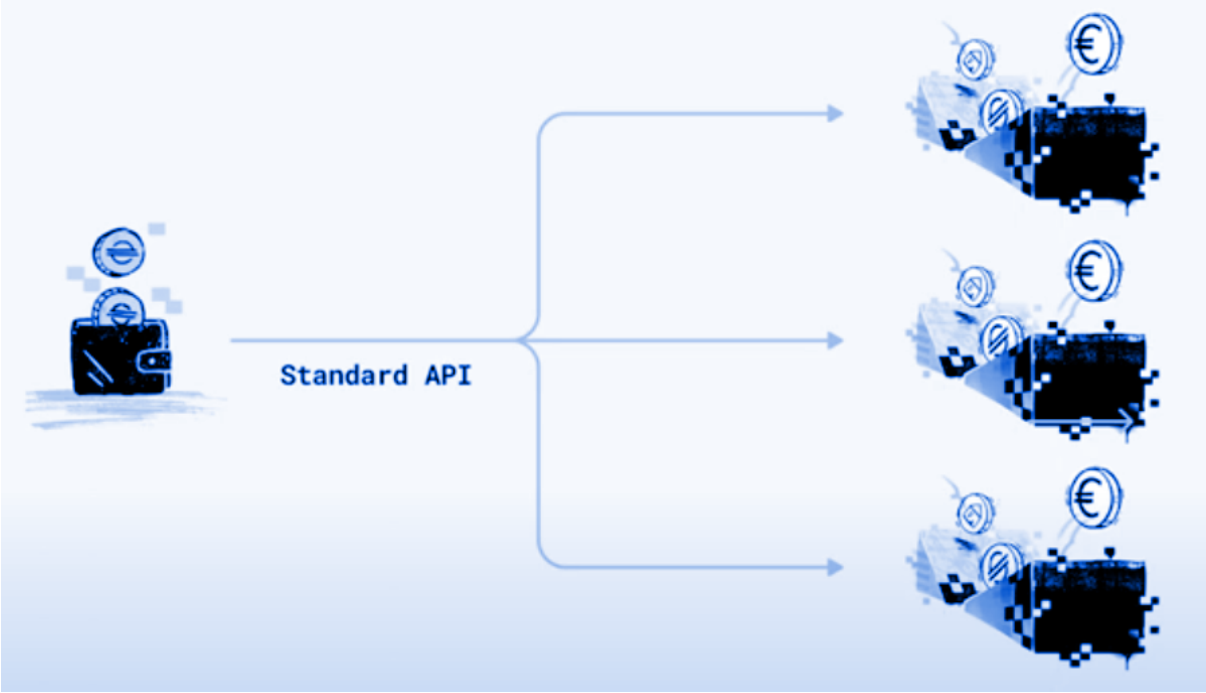
Architecture Document

Created by Soneso on 19. Oct 2024

Table of Contents

Introduction

Interoperability between payment applications such as wallets and on/off ramps such as Stellar anchors has always been one of the most important pillars for Stellar. That's why the Stellar Development Foundation (SDF) started defining standards for wallets and anchors very early on to facilitate their interoperability and to maximize the adoption.



The basic consideration here is that, from the wallet developer's point of view, it should make no difference whether they integrate with one anchor or with several different ones.

Since the anchors should all offer the same standard API, the effort of integrating with a single anchor or with multiple anchors should, in an ideal world, only be a matter of configuration and not of writing specific source code for the integration of each of the anchors.

The power of this interoperability standard is that you code once and ideally with very little modifications you can support multiple companies and you can provide a very competitive service in terms of your wallet offering.

However, because these standards were defined very early on to maximize adoption, with very little information about the most important user cases, the consequence was that they had to be extended again and again over time to reflect the wallet and anchor builder’s needs.

But evolving the standard is not an easy thing, you don't want to break anyone who is already compatible, so you need to create this backwards compatible evolution system that ends up becoming almost like a patchwork where you either grow by having a fixed set of endpoints with a lot of optional parameters or you have a bunch of endpoints that do slightly different things. It is probably impossible to get as close to a good API as if you had defined that from the go considering all the use cases and accommodated all of them. As a result, these standards have become very complex, extensive, and interconnected over time. See for example SEP-06 or SEP-24.

Besides of premature definition, SDF also needs to communicate about those changes to make sure that the developers can accommodate those changes and additions to have a fully compatible interoperable system. But because Stellar has an open ecosystem, it is hard to reach all developers in advance to tell them that they need to accommodate new use cases, new endpoints, and other changes.

The standard API for interoperability between wallets and anchors was defined by SDF via so-called Stellar Ecosystem Proposals (SEPs). And the original recommendation for developers to become interoperable was to read the SEPs and implement them. However, since these are very complex, extensive and cover many edge cases, this led to many different interpretations during implementation.

That's why SDF came up with the idea of implementing so-called wallet SDKs about two years ago. These are software libraries that implement the needed functionality and that developers can integrate into their program, so as not to keep reinventing the wheel over and over again. These SDKs also have many other benefits, as we will see below.

The integration of a wallet SDK makes it very easy for the developer of a client application to become interoperable. They do not have to develop the communication with the anchor themselves, but only use the SDK. Furthermore, this helps to remain interoperable, because

every time the standard evolves, a new version of the SDK is also released, and the developer only must update to use the new version of the library. It is a common, natural process for software developers.

Because the functionality is already implemented, developers can become interoperable much faster and at a much lower cost than if they were to implement the functionality from scratch themselves. This means that developers have more time to focus on implementing their business logic instead of constantly reinventing the wheel by having to implement Stellar details.

Scope of this document

Wallet SDKs are currently offered for three different programming languages. The Stellar Development Foundation created and maintained the TypeScript and the Kotlin Wallet SDKs. And the Flutter (dart) Wallet SDK was developed and is maintained by me (Soneso). You can find their documentation on the official Stellar docs website under: "Build a Wallet with the Wallet SDK".

As you can see in the Stellar docs, all three SDKs offer the same functionality and the same interfaces for integration. These were defined by SDF when the TypeScript and Kotlin SDKs were implemented.

All three have a similar architecture, so why write an architecture document for the next programming language (swift) when the architecture is already given and has already been implemented three times?

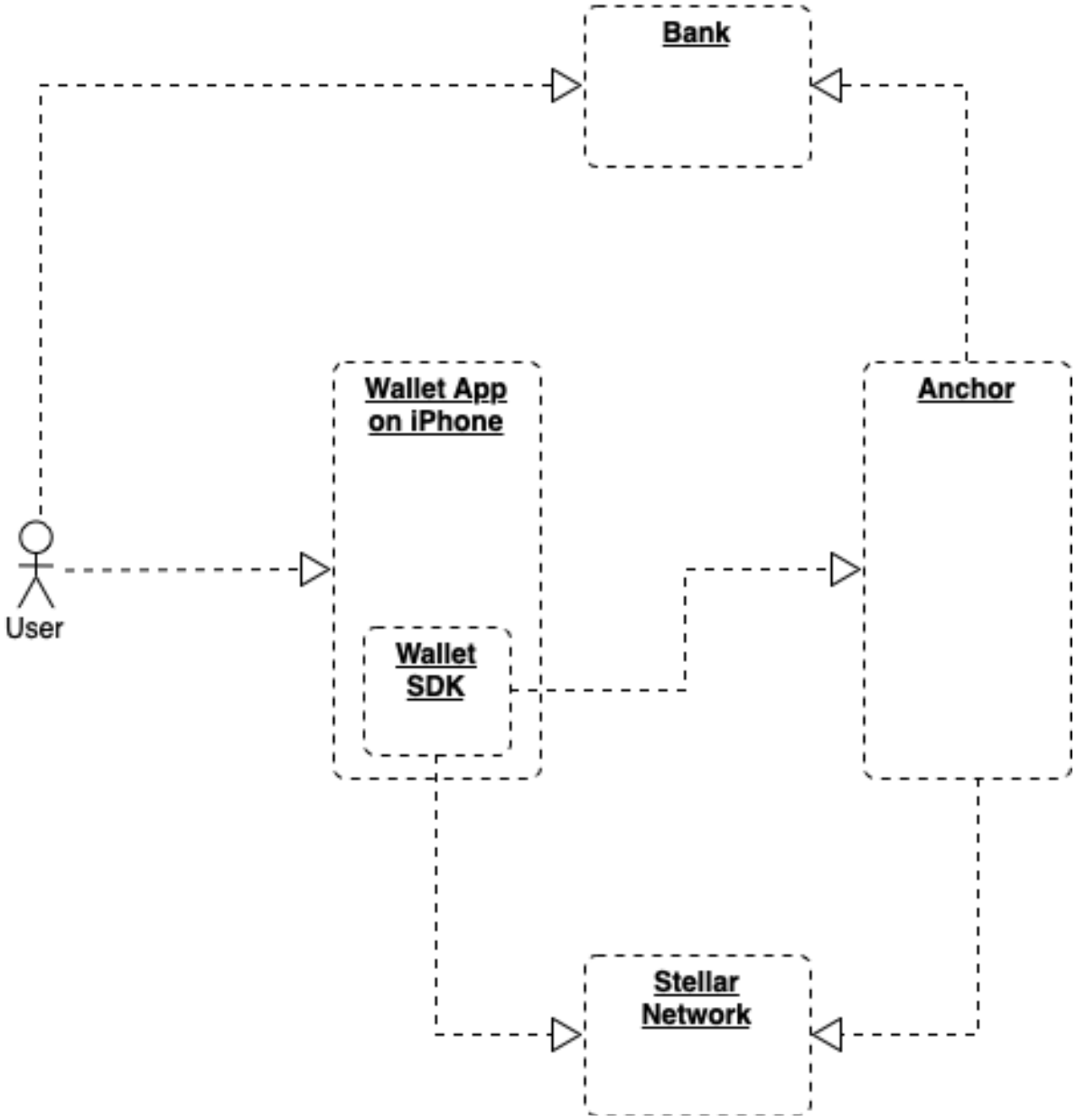
Well, the main reason is that developing such an SDK is a large and time-consuming task. This raises the question of financing. To support the financing of new projects, the SDF has launched a program called SCF (Stellar Community Fund). To be able to apply to the SCF program, a presentation of the project must of course be made in advance and various forms must be completed. This also includes the provision of an architecture document to assess whether the proposed project is in line with the SCF goals, and the developers are sufficiently familiar with the Stellar technology.

But the main advantage is that the architecture document helps interested developers who want to contribute to the open-source Swift Wallet SDK. It helps them to understand faster and better how the SDK is structured. This enables them to provide high quality and more targeted support for further development and troubleshooting.

Context

As described in the introduction, the Swift SDK is used by Swift developers who want to develop payment applications, such as a wallet. The main focus of the SDK is to abstracts the Stellar-specific functionality needed to communicate with on/off ramps such as Anchors. It also handles the needed communication with the Stellar Network.

The Swift programming language is used to implement native applications for Apple products such as iPhones, iPads, Macs, etc.



Structure overview

Main entry point of the SDK is the Wallet class. It provides services to build wallet applications on the Stellar network. To provide the services it uses a Stellar configuration and an optional app configuration that are passed to the constructor on initialization.

The Wallet class provides following services:

- Stellar: for interaction with the Stellar Network
- Anchor: for building on/off ramps with anchors
- Recovery: for implementing account recovery
- Sep7Uri: for parsing and construction Stellar SEP-7 (delegated signing) URIs.

In the following chapters, we will discuss the individual services.

Stellar Network

The Stellar class provides methods and services for interaction with the Stellar Network. A configured instance of the Stellar class is provided via the Wallet class (See chapter Structure Overview).

The functionality to be offered here allows developers to easily build, sign, sponsor and send different types of Stellar transactions to the Stellar Network. The account services facilitate the work with Stellar accounts, and one will also be able to access the underlying iOS SDK that can be used to fetch all kind of data from the Stellar Network provided by Horizon.

The following features will be available: Create Account, Modify Account, Modify Assets (Trustlines), Swap Assets, Transfer (Payments), Path Pay, Manage Sell and Buy Offers, Account Merge, Fund Testnet Account, Building Advanced Transactions, Sponsoring Transactions, Fee Bump Transactions, Accessing Horizon SDK, Submit Transaction, Using XDR to Send Transaction Data. The corresponding test cases and documentation must be implemented. See also official Stellar Wallet SDK docs.

Anchor

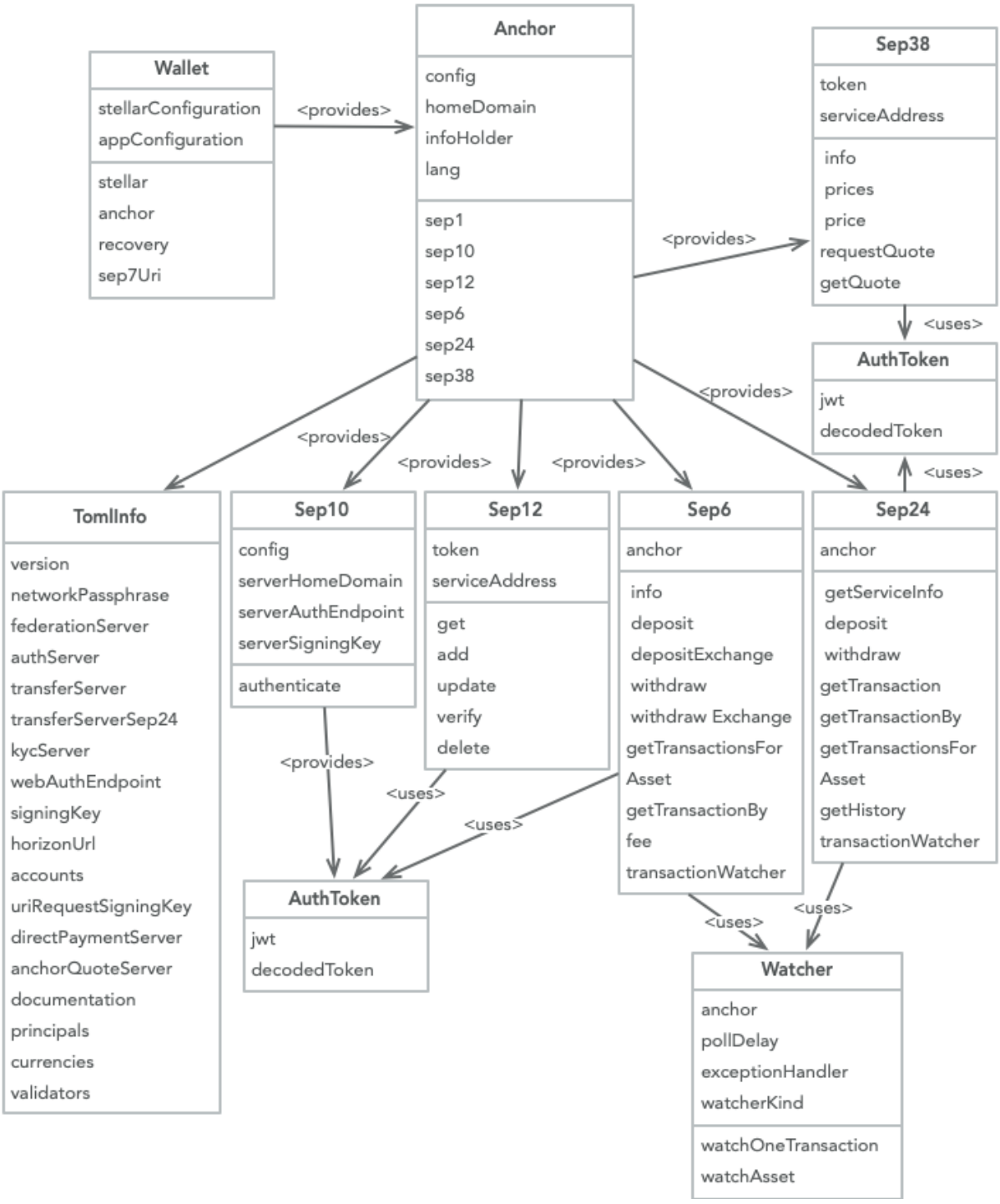
The Anchor class provides methods and services for interaction with Anchors covering the implementation defined by the involved Stellar Ecosystem Proposals (SEPs).

The SDK will offer support for the following SEPs:

- SEP-10 (Stellar authentication) & SEP-01 (Fetching and parsing the Anchors public information): Wallets connect to anchors using a standard way of authentication via the Stellar network defined by SEP-10. The SDK will offer all the components required for SEP-10 Authentication with Anchors: Loader & parser for SEP-01 (Stellar Toml). The SEP-10 authentication process will be covered including Default Signer, Client Domain Signer, Validations, JWT tokens handling, preparation of Auth Tokens to be used for the successive SEP's to be covered. The corresponding test cases and documentation must be implemented. See also official Stellar Wallet SDK docs.
- SEP-24 (Hosted Deposit and Withdrawal): The SEP-24 standard defines the standard way for anchors and wallets to interact on behalf of users. Wallets use this standard to facilitate exchanges between on-chain assets (such as stablecoins) and off-chain assets (such as fiat, or other network assets such as BTC). The following features will be available: Withdrawal, Deposit, Fetching Single Transactions, Transaction History, Transactions Watcher, SEP-09 KYC fields. Access to these features is made possible via an API that is predefined for all Wallet SDKs (see docs). Test cases and documentation will be available.
- SEP-38 (Quotes service): The SEP-38 standard defines a way for anchors to provide quotes for the exchange of an off-chain asset and a different on-chain asset, and vice versa. The following features will be available: Info, Get Prices, Get Price, Post Quote, Get Quote, Asset Identification Format. Access to these features is made possible via an API that is predefined for all Wallet SDKs (see docs). Test cases and documentation will be available.
- SEP-12 (KYC API: defines a standard way for stellar clients to upload KYC (or other) information to anchors and other services. Our SEP-06 implementation will use this protocol, but it can serve as a stand-alone service as well. The following features will be available: Add customer, Get Customer, Update Customer, Delete Customer, Verify Provided fields, Upload binary data. Access to these features is made possible via an API that is predefined for all Wallet SDKs (see docs). Test cases and documentation will be available.
- SEP-06 (Programmatic Deposit and Withdrawal): The SEP-6 standard defines a way for anchors and wallets to interact on behalf of users. Wallets use this standard to facilitate exchanges between on-chain assets (such as stablecoins) and off-chain assets (such as fiat, or other network assets such as BTC). See also the Wallet SDK

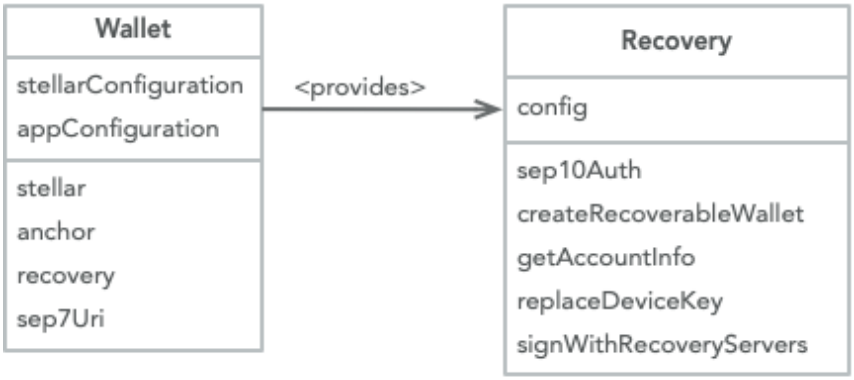
docs here. The following features will be available: Get Anchor Info, Deposit, Deposit-Exchange, Providing KYC Info, Withdrawal, Withdrawal-Exchange, Get Exchange Info, Fetching Transactions, Tracking Transactions. Access to these features is made possible via an API that is predefined for all Wallet SDKs. Test cases and documentation will be available.

Furthermore, the SDK will offer a service that simplifies the monitoring of anchor transactions, so that the user can be informed about the status updates of the transactions and can act accordingly.



Recovery

The SEP-30 standard defines the standard way for an individual (e.g., a user or wallet) to regain access to their Stellar account after losing its private key without providing any third-party control of the account. The Recovery class implements this functionality. During this flow the wallet communicates with one or more recovery signer servers to register the wallet for a later recovery if it's needed. See Wallet SDK docs here. The following features will be available: Create Recoverable Account, Get Account Info, Recover Wallet. Access to these features is made possible via an API that is predefined for all Wallet SDKs. Test cases and documentation will be available.



Sep7: URI Scheme for delegated signing

The SEP-07 standard introduces a URI Scheme that can be used to generate a URI that will serve as a request to sign a transaction. The URI (request) will typically be signed by the user's trusted wallet where she stores her secret key(s). The Sep7Uri class will provide the functionality needed to easily parse and construct SEP-07 Stellar URIs. See also Stellar Wallet SDK docs here. Test cases and documentation will be available.