NEXT GENERATION INTERNET

GNU Taler for Developers

Iván Ávalos

COSIN'24

What is Taler?

`https://taler.net/en/features.html`

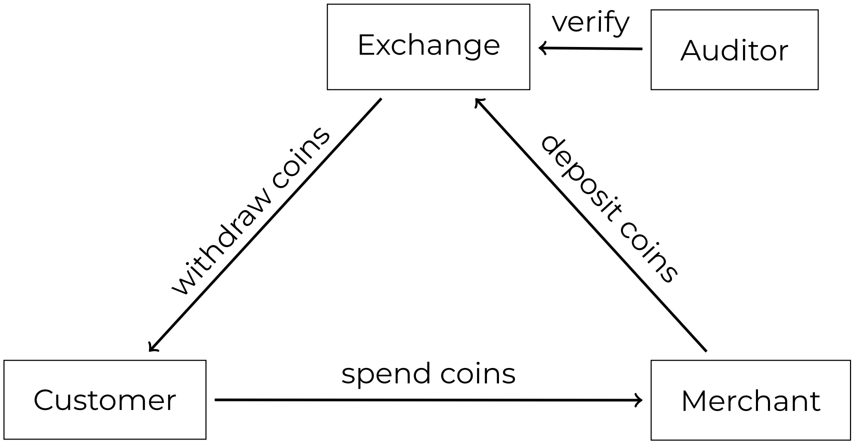
Taler is

- a Free/Libre software *payment system* infrastructure project
- ...with a surrounding software ecosystem
- ...and a company (Taler Systems S.A.) and community that wants to deploy it as widely as possible.

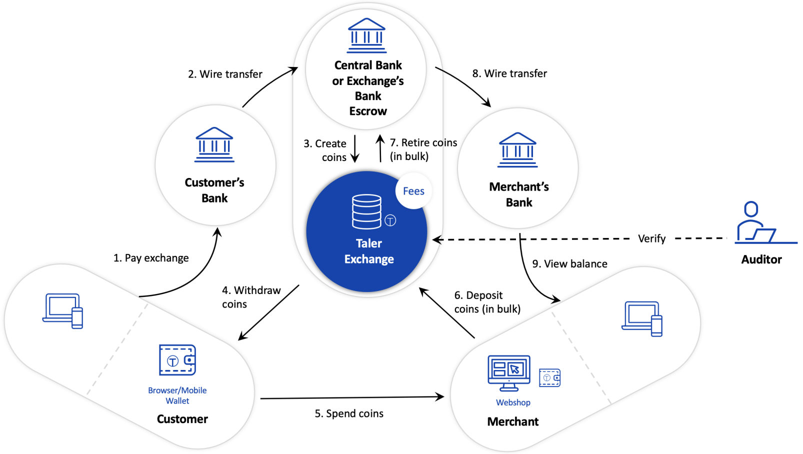
However, Taler is

- *not* a currency
- *not* a long-term store of value
- *not* a network or instance of a system
- *not* decentralized
- *not* based on proof-of-work or proof-of-stake
- *not* a speculative asset / "get-rich-quick scheme"

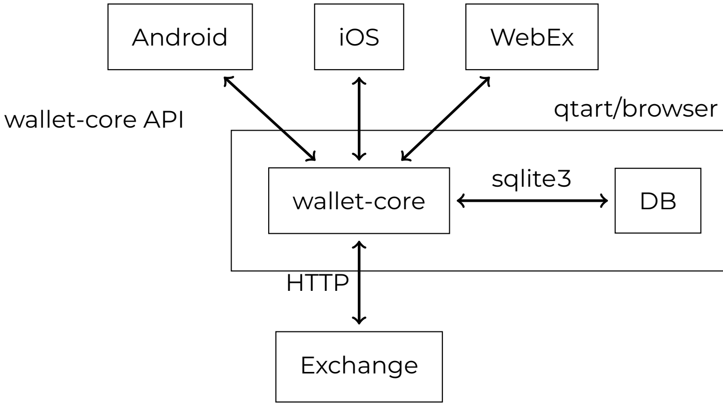
Taler overview



Architecture of Taler



Wallet architecture



GNU Taler wallet

wallet-core

wallet-core is the component that powers the Taler wallets across different platforms. It is written in TypeScript and it implements of all the core functionality required by the wallets. It takes care of the following:

- database management (SQLite3)
- task shepherding
- cryptography
- wallet operations
- communication with the exchange

Most of its functionality is exposed via **requests**. Apps using wallet-core can interact with it by calling different request methods, passing parameters, and then e.g. rendering in the UI the data contained in the response or showing an error message.

GNU Taler wallet

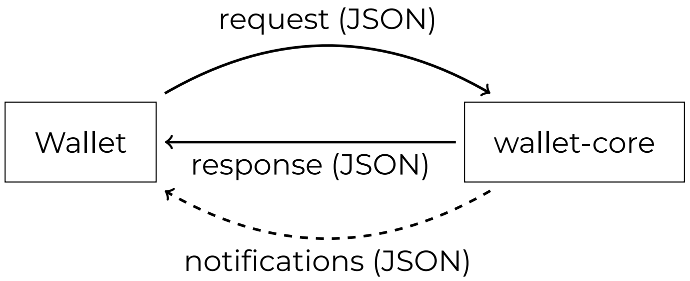
qtart

qtart (**Q**uickJS **TA**ler **R**un**T**ime) is a QuickJS-based runtime that embeds wallet-core into a native library for usage outside of the browser, such as the mobile apps or any future desktop app.

- Based on the acclaimed QuickJS JavaScript engine.
- Implements native modules for cryptography.
- Supports native HTTP networking (with multi-threading).
- Provides access to the wallet-core API via a simple callback-based interface.
- Keeps us from having to rewrite wallet-core for every platform!

Wallet-core API

Introduction



- Documentation: `https://docs.taler.net/wallet/wallet-core.html`

Wallet-core API

Request structure

Field	Type	Description
`id`	integer	request ID
`operation`	string	API operation
`args`	object	request arguments

Example

```
{
  "id": 0,
  "operation": "init",
  "args": { "logLevel": "INFO" }
}
```

Wallet-core API

Response structure

Field	Type	Description
`type`	string	either `response` or `error`
`id`	integer	request ID
`operation`	string	API operation
`result`	object	response data

Example

```
{ "type": "response",
  "id": 0,
  "operation": "init",
  "result": {...} }
```

Wallet-core API

Notification structure

Field	Type	Description
`type`	string	will be `notification`
`payload`	object	notification data

Example

```
{
  "type": "notification",
  "payload": {
    "type": "task-observability-event"
  }
}
```

Wallet-core API
Error structure

An error can be contained inside a response or a notification, and includes the following data, in some cases along with extra fields:

Field	Type	Description
`code`	integer	GANA error code
`when`	timestamp?	time when it occurred
`hint`	string?	error message

Example

```
{ "code": 7001,
  "hint": "could not resolve host: demo.taler.net",
  "when": { "t_ms": 1718726899827 } }
```

GNU Taler wallet

Building wallet-core

1. Install Python, Node.js, NPM and pnPM (`https://pnpm.io/`)
2. Clone the Git repository (`https://git.taler.net/wallet-core.git`)
3. Run the bootstrap script

   ```
   $ ./bootstrap
   ```

4. Run the configuration script

   ```
   $ ./configure
   ```

5. Build all the components!

   ```
   $ make
   ```

Note: the relevant `.mjs` file for building qtart will be created under `packages/taler-wallet-embedded/dist/taler-wallet-core-qjs.mjs`.

GNU Taler wallet

Building web extension

In order to build the web extension, please follow the steps in the previous slide, and then run the following command:

```
$ make webextension
```

This will generate two files under `packages/taler-wallet-webextension`:

- `extension/v2/taler-wallet-webextension-$VERSION.zip`
- `extension/v3/taler-wallet-webextension-$VERSION.zip`

Those files are the final packaged extensions. Depending on the manifest version supported by your browser, you should install either `v2` (e.g. Firefox) or `v3` (e.g. Chromium/Chrome).

GNU Taler wallet

Building qtart (Android)

1. Install Docker and Docker Compose
2. Clone the Git repository (`https://git.taler.net/quickjs-tart.git`)
3. Copy into the root the `.mjs` file produced when building wallet-core.
4. Descend into the `docker-android` directory.
5. Create an empty `.env` file.
6. Run the following command:

```
$ docker-compose run --rm quickjs
```

A local Maven repository will be created under the `.m2/repository` directory relative to the Git repository root. The absolute path to this directory should be added as a URL to the project-level `build.gradle` file of the Android app, under `allprojects/repositories`.

GNU Taler wallet
Building Android app

1. Install Android Studio.
2. Clone the Git repository (`https://git.taler.net/taler-android.git`).
3. Open the project with Android Studio.
4. Build qtart from source (optional).
   - Build wallet-core from source.
   - Copy the resulting `.mjs` file to qtart.
   - Run the dockerized qtart build.
   - Add local Maven repository to the Android project.
5. Build and run the Android app.

GNU Taler wallet

Building iOS app

1. Install Xcode (in macOS).
2. Under the same directory:
   - Clone the iOS app Git repository (`https://git.taler.net/taler-ios.git`)
   - Clone the qtart Git repository (`https://git.taler.net/quickjs-tart.git`)
3. Build wallet-core from source.
4. Copy the resulting `.mjs` file to qtart.
5. Open the iOS project with Xcode.
6. Build and run the iOS app.

Wallet-core CLI

The CLI can be used to test wallet-core features quickly. In order to install (only) the wallet-core CLI and other CLI tools, run the following command after setting up the wallet-core repository:

```
$ make install-tools
```

Useful commands:

```
$ taler-wallet-cli --help             # print help message
$ taler-wallet-cli transactions       # print transaction list
$ taler-wallet-cli handle-uri $URI # handle a Taler URI
$ taler-wallet-cli advanced withdraw-manually \
    --exchange https://exchange.demo.taler.net/ \
    --amount KUDOS:5                  # perform manual withdrawal
$ taler-wallet-cli run-pending        # attempt to finish all pending tasks
$ taler-wallet-cli run-until-done     # run until all work is done
```

Wallet-core CLI

It is also possible to call wallet-core API requests directly from the CLI, even when there is not a command for it:

```
$ taler-wallet-cli api getWithdrawalDetailsForAmount \
      '{"exchangeBaseUrl":"https://exchange.demo.taler.net/",
        "amount":"KUDOS:10"}'
```

Wallet-core CLI

By design, wallet-core CLI only performs background tasks during each execution, and when it completes the requested action, it quits. However, it is also possible to run it as a daemon and run commands in a client-server fashion, as shown below:

Run the wallet as a daemon (in the foreground)

```
$ taler-wallet-cli advanced serve
```

Connect to the daemon and execute an action

```
$ taler-wallet-cli --wallet-connection=$HOME/.wallet-core.sock ...
```

NGI TALER

Hacking on wallet-core

Important files

- `packages/taler-util/src/` (common Taler code)
  - `taler-types.ts` (core Taler protocol type definitions)
  - `transactions-types.ts` (transaction type definitions)
  - `wallet-types.ts` (core wallet API type definitions)
- `packages/taler-wallet-core/src/` (main wallet-core code)
  - `exchanges.ts` (exchange management and operations)
  - `pay-merchant.ts` (payments to merchants)
  - `pay-peer-*.ts` (p2p send/receive operations)
  - `shepherd.ts` (task scheduler)
  - `testing.ts` (test functions)
  - `transactions.ts` (transaction management)
  - `wallet-api-types.ts` (wallet-core API request/response types)
- `packages/taler-harness/src/` (integration tests)

Other components

- Merchant
- Auditor
- Challenger
- Sync
- GNU Anastasis
- Twister
- libeufin

Tutorials

- Wallet tutorials: `https://docs.taler.net/taler-wallet.html`
- Video tutorials: `https://tutorials.taler.net/`
- Support forum: `https://ich.taler.net/`

Funding

`https://nlnet.nl/propose`

Candidates that passed 1st round from April 1st submission proposed:

- Some more integrations (frameworks, ERP)
- Merchant implementation (?)
- Improvements to wallet usability

Acknowledgements

Funded by the European Union (Project 101135475).

Co-funded by
the European Union

Funded by SERI (HEU-Projekt 101135475-TALER).

Project funded by

Schweizerische Eidgenossenschaft
Confédération suisse
Confederazione Svizzera
Confederaziun svizra

Swiss Confederation

Federal Department of Economic Affairs,
Education and Research EAER
State Secretariat for Education,
Research and Innovation SERI

Views and opinions expressed are however those of the author(s) only and do not necessarily reflect those of the European Union.

Neither the European Union nor the granting authority can be held responsible for them.