



Flutter 1.22.6

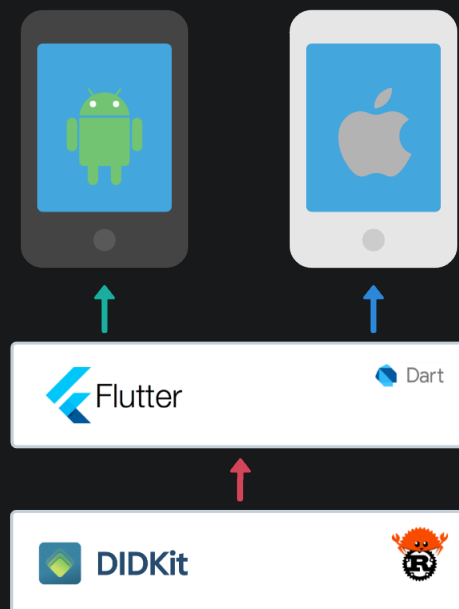
ssi v0.1

DIDKit v0.1

License Apache-2.0

22

Credible is a reference implementation of a native mobile wallet that supports W3C Verifiable Credentials and Decentralized Identifiers built on DIDKit and Flutter. We packaged the DIDKit library written in Rust into a Flutter application that can be rendered equally to both Android and iOS, using C bindings and Dart's FFI capabilities respectively. This is the wallet counterpart to the rich, growing issuance/verification toolkit supplied by DIDKit, the two pillars of a reference architecture for creating trusted interactions at scale using verifiable [credentials](#).



## Core Features

- QR support to initiate and execute issuance and presentation of verifiable credentials
- Official decentralized-identity wallet of [DID Method Tezos](#), i.e. "did-tz"
- Changing only a few lines of code, Credible can be rebuilt from source to natively handle any of [the DID methods supported by DIDKit](#) instead of did-tz: did-key, did-ethr, did-onion...
- Built in the Flutter native framework, for leaner, faster

builds and less dependencies, but also available through the top package manager of both ecosystems for more involved integrations (see the [Native Development](#) page)


- Demonstration-build installable from Apple Test Flight and [Google Play Store](#) (Early Access)

## Extensibility

We built Credible to be solid on the foundations and light on the context-specific details, meaning that it handles DIDs and VCs to an exceptional degree of conformance with the core specifications for each. We feel, however, that the protocols, semantics, and higher-order decisions taken by anyone integrating a wallet into one or more credential ecosystems should not be constrained, much less taken, by a reference implementation: those choices are up to Credible's white-labelers and forkers.

Credible is end-to-end open source, however, so do [open an issue](#) if you have thoughts on how to better

support the protocols you are implementing for authentication, authorization, VC exchange, etc. Or, if you want to contribute code, we are open to PRs on Credible and its Spruce-governed dependencies, with the appropriate [contributor agreements](#) and review!

 [Edit this page](#)

# Installation

## App stores

We are also in the process of listing Credible on the iOS TestFlight and Android Play Beta programs, and eventually their respective app marketplaces plus F-Droid.

## Common Dependencies

To manually build Credible for either Android or iOS, you will need to install the following dependencies (instructions follow):

- Rust
- Java 7 or higher
- Flutter ( `dev` channel)
- [DIDKit/SSI](#)
- `wasm-pack` (WEB)
- `binaryen` (WEB and targeting ASM.js)

## Rust

It is recommended to use [rustup](#) to manage your Rust installation.

## Java

On Ubuntu you could run:

```
$ sudo apt update  
$ sudo apt install openjdk-8-jdk
```

For more information, please refer to the documentation of your favorite flavour of Java and your operating system/package manager.

## Flutter

Please follow the official installation instructions available [here](#) to install Flutter, don't forget to also install the build dependencies for the platform you will be building (Android SDK/NDK, Xcode, etc).

We currently only support build this project using the `dev` channel of Flutter.

To change your installation to the `dev` channel, please execute the following command:

```
$ flutter channel dev  
$ flutter upgrade
```

To confirm that everything is setup correctly, please run the following command and resolve any issues that arise before proceeding to the next steps.

```
$ flutter doctor
```

## **wasm-pack** (Required for both WEB targets)

The `wasm-pack` cannot yet be compiled from `crates.io`. For now, to build the WASM target you will need `wasm-pack`, the fastest way is to fetch the script from github and run it in the credible root directory:

```
$ curl https://rustwasm.github.io/wasm-pack/installer/init.sh -sSf | sh
```

## **binaryen**

To build Credible for WEB using ASM.js you will need `binaryen`, which allows the conversion of DIDKit WASM to ASM.js. This is necessary in context where WASM support is not available and DIDKit needs to run in pure Javascript. More detailed instructions on how to build `binaryen` can be found [here](#).

If you are in a UNIX-like distribution you just have to clone the repo and build, we recommend cloning into your ``${HOME}``, to avoid having to specify the ``${BINARYEN_ROOT}`` variable:

```
$ git clone
https://github.com/WebAssembly/binaryen
~/binaryen
$ cd ~/binaryen
$ cmake . && make
```

Not that binaryen support for OS X and Windows is still limited, so it is highly recommended to build it in a linux shell on OS X and on WSL2 in Windows. For instructions on building it natively in Windows, see [the binaryen FAQ](#) on github.

## DIDKit and SSI

This project also depends on two other Spruce projects, `DIDKit` and `SSI`.

These projects are all configured to work with relative paths by default, so it is recommended to clone them all as subdirectories of the same root directory, for example `$HOME/spruceid/didkit`  
`$HOME/spruceid/ssi`  
`$HOME/spruceid/credible`  
`$HOME/spruceid/treehouse` etc

## Target-Specific Dependencies

### Android Dependencies



To build Credible for Android, you will require both the Android SDK and NDK.

These two dependencies can be easily obtained with [Android Studio](#), which install further dependencies upon first being opened after installation. Installing the appropriate Android NDK (often not the newest) in Android Studio can be accomplished by going to Settings > Appearance & Behavior > System Settings > Android SDK and selecting to install the "NDK (Side by Side)". An alternative method of installing SDK and NDK without Android Studio can be found in the optional [install\\_android\\_dependencies.sh](#) script included [here](#).

If your Android SDK doesn't live at

```
$HOME/Android/Sdk
```

 you will need to set `ANDROID_SDK_ROOT` like so:

```
$ export  
ANDROID_SDK_ROOT=/path/to/Android/Sdk
```

*Note: Some users have experienced difficulties with expected cross-compilation artefacts missing from the newest NDK, which is downloaded by default in the installation process. If you experience errors of this kind, you may have to manually downgrade or install multiple NDK versions as [shown here] ([img/ndk\\_downgrade.png](#)) in the Android Studio installer (screengrabbed from an Ubuntu*

installation). Alternately, running all or parts of the `install_android_dependencies.sh` script may be helpful.

If your `build-tools` and/or `NDK` live in different locations than the default ones inside `/SDK/`, or if you want to specify a specific NDK or build-tools version, you can manually configure the following two environment variables:

```
$ export ANDROID_TOOLS=/path/to/SDK/build
tools/XX.X.X/
$ export
ANDROID_NDK_HOME=/path/to/SDK/ndk/XX.X.X/
```

:::

## iOS Dependencies

To build Credible for iOS you will need to install CocoaPods, which can be done with Homebrew on MacOS, WSL2, or Linux-based systems. You will also need `XCode`, which is not available for WSL2 or Linux systems; it is currently possible but not recommended to install XCode in those build environments.

```
$ sudo apt install linuxbrew-wrapper
$ brew install cocoapods
```

## Web Dependencies

To build Credible for WASM, you will need the Node.JS and its package manager, `npm` to be installed:

```
$ sudo apt install nodejs
$ sudo apt install npm
```

*Note: in some environments, such as Ubuntu 18.04, `npm` may not automatically install a new enough version for our makefile to execute successfully; overriding it with `npm`'s *internal commands* may be necessary.*

## Building DIDKit for different targets

### Android

To build `DIDKit` for the Android targets, you will go to the root of `DIDKit` and run:

```
$ make -C lib install-rustup-android
$ make -C lib
./target/test/java.stamp
$ make -C lib
./target/test/aar.stamp
$ make -C lib
./target/test/flutter.stamp
$ cargo build
```

*This may take some time as it compiles the entire*

*project for multiple targets*

## Android APK

```
$ flutter build apk --no-sound-null-safety
```

## Android App Bundle

```
$ flutter build appbundle --no-sound-null-safety
```

## iOS

To build DIDKit for the iOS targets, you wGo to the root of `DIDKit` and run :

```
$ make -C lib install-rustup-ios  
$ make -C lib  
../target/test/ios.stamp  
$ cargo build
```

## Web using WASM

```
$ make -C lib  
../target/test/wasm.stamp
```

## Web using ASM.js

If you have installed `bynarien` somewhere other than `$HOME`, you will have to set `BYNARIEN_ROOT` as shown below, otherwise, just run the `make` command.

```
$ export
BINARIEN_ROOT=/path/to/binaryen
$ make -C lib
../target/test/asmjs.stamp
```

## Building Credible

You are now ready to build or run Credible.

### Run on emulator

If you want to run the project on your connected device, you can use:

```
$ flutter run --no-sound-null-safety
```

### Run on browser

If you want to run the project on your browser, you can use:

```
$ flutter run --no-sound-null-safety
-d chrome --csp --release
```

Otherwise, Flutter allows us to build many artifacts

for Android, iOS and WEB, below you can find the most common and useful commands, all of which you should run from the root of Credible.

## iOS .app for Simulator

```
$ flutter build ios --no-sound-null-safety --no-codesign --simulator
```

## iOS .app for Devices

```
$ flutter build ios --no-sound-null-safety --no-codesign
```

## iOS IPA

```
$ flutter build ipa --no-sound-null-safety
```

## Web

```
$ flutter build web \  
  --no-sound-null-safety \  
  --csp \  
  --release
```

If you don't have support for WASM, you'll probably need to provide your own `canvaskit` dependency without WASM as well as `DIDKit`, to do

that you need to specify the

`FLUTTER_WEB_CANVASKIT_URL` in the build command like below.

```
$ flutter build web \  
  --no-sound-null-safety \  
  --csp \  
  --dart-  
  define=FLUTTER_WEB_CANVASKIT_URL=vendor,  
  \  
  --release
```

For more details about any of these commands you can run

```
$ flutter build $SUBCOMMAND --help
```

## Note about `nullsafety`

While we are ready to migrate to Dart with `nullsafety`, a couple of the dependencies of the project are still lagging behind, so we need to add `--no-sound-null-safety` to both run and build commands for the time being.

## Note about `canvaskit`

Since by default `canvaskit` comes in a `WASM` build, in order to the `ASM.js` be fully supported `canvaskit` was manually built for this target.

A prebuilt `canvaskit` is already included in the

Credible web folder. If you want to build it by yourself, however, follow these steps:

- Install `emscripten`
- Clone `Skia` repository and pull its dependencies

```
git clone
https://skia.googlesource.com/skia.git
--depth 1 --branch canvaskit/0.22.0
cd skia
python2 tools/git-sync-deps
```

- Modify build script

```
modules/canvaskit/compile.sh
```

```
diff --git
a/modules/canvaskit/compile.sh
b/modules/canvaskit/compile.sh
index 6ba58bfae9..51f0297eb6 100755
--- a/modules/canvaskit/compile.sh
+++ b/modules/canvaskit/compile.sh
@@ -397,6 +397,7 @@ EMCC_DEBUG=1
 ${EMCXX} \
     -s MODULARIZE=1 \
     -s NO_EXIT_RUNTIME=1 \
     -s INITIAL_MEMORY=128MB \
-    -s WASM=1 \
+    -s WASM=0 \
+    -s NO_DYNAMIC_EXECUTION=1 \
     $STRICTNESS \
     -o $BUILD_DIR/canvaskit.js
```

- Build `canvaskit`



```
$ cd modules/canvaskit  
$ make debug
```

- Replace this line on

```
$SKIA/modules/canvaskit/canvaskit/bin  
/canvaskit.js
```

```
618c618  
< var isNode = !(new Function('try  
{return this===window;}catch(e){  
return false;}')());  
---  
> var isNode = false;
```

- Copy

```
$SKIA/modules/canvaskit/canvaskit/bin  
/canvaskit.js to  
$CREDIBLE/web/vendor/
```

- Build Credible as described above.

## Troubleshooting

### Build Environment Problems

If you encounter any errors in the build process described here, please first try clean builds of the projects listed.


For instance, on Flutter, you can delete build files to start over by running:

```
$ flutter clean
```

Also, reviewing the [install\\_android\\_dependencies.sh](#) script line by line or even rerunning it line by line may be helpful.

## Opaque Makefile error messages

- NPM may through an opaque error when compiling to WASM or ASM.js if `apt install` has installed too old of a version of `npm` based on your operating system kernel. This can be manually overridden from `npm`'s [internal commands](#)

 [Edit this page](#)