ElectrumSV SDK

The ElectrumSV SDK developers

Nov 28, 2022

GETTING STARTED

1 Overview 3

2 What Does It Do? 5

Licence
The Open BSV License

Maintainers
Roger Taylor, AustEcon

Project Lead
Roger Taylor

Language
Python (>=3.9)

Homepage
https://github.com/electrumsv/electrumsv-sdk

CHAPTER

ONE

OVERVIEW

This project provides a consolidated set of resources that together can allow a developer, whether working on ElectrumSV or on any other bitcoin application, to develop, run and test while offline (and is especially aimed at facilitating rigourous CI/CD functional testing).

CHAPTER

TWO

WHAT DOES IT DO?

A commandline tool that makes it very easy to spin up localhost instances of:

- Bitcoin Node
- ElectrumX
- ElectrumSV (as a daemon with a REST API or as a GUI desktop wallet)
- Merchant API
- Whatsonchain Block explorer.

To get started, please checkout:

- *Getting Started* documentation

There is full support for:

- Windows
- Linux
- MacOS X

Networks supported:

- Regtest
- Testnet

2.1 Why would you want this?

2.1.1 Accelerated development iteration cycle

- If you are building an application that needs to perform SPV verification you want to know that when a block is mined, the logic is correct.
- You cannot afford to wait around 10 minutes for a block to be mined on one of the public test networks.
- Ensure correct handling of rare events like reorgs or "unmatured" coins (freshly mined coins that require 100 block confirmations before they can be moved).

2.1.2 CI/CD Pipeline testing

- Enough said

2.1.3 Benchmarking / high-throughput testing

- Because, we are not supposed to do that on the standard public testnet…
- You could use the scaling testnet, but in some cases it's easier to start out with RegTest in early development and graduate to the scaling-testnet later when you're ready for it.

2.1.4 Solution:

- With a local RegTest node, you can mine blocks on demand.
- You can run two nodes locally and simulate a reorg (we haven't added reorg simulations yet but that's coming).
- You will never have to ask for more testnet coins - just mine more blocks and top up.
- You will not be at the mercy of 3rd party service providers staying operational to continue to build and test your application.

2.1.5 Release Notes:

See *Release Notes*.

Installing the SDK

Pre-requisites

All components:

- **Python3.7+** (64 bit) from: https://www.python.org/downloads/ for your platform.

For Whatsonchain:

- **Node.js (12+)**

For Merchant API:

- **Postgres** (with a priviledged admin user: `user=mapimaster` and `password=mapimasterpass`)

On linux, please install these system dependencies before proceeding:

```
sudo apt-get update
sudo apt-get install libusb-1.0-0-dev libudev-dev
sudo apt-get install git net-tools xterm
sudo apt-get install libssl1.0-dev   # OR libssl-dev for newer linux distros
python3 -m pip install pysqlite3-binary
```

On MacOS X please install these system dependencies before proceeding:

```
brew upgrade sqlite3
```

Install the SDK package

To install from pypi run:

```
pip install --upgrade electrumsv-sdk
```

As long as your <python dir>/Scripts directory is in the system PATH environment variable, you now have global access to a script called `electrumsv-sdk.exe` from any console window.

Troubleshooting (command not found - not on PATH):

If the electrumsv-sdk command is not accessible yet, it will be because the script is not on your system PATH.

On windows:

Add `C:\Users\<username>\AppData\Local\Programs\Python\Python38\Scripts` to your PATH in 'system environment variables' (from the windows search bar) and reload a new terminal window to try again. (substituting <username> and Python38 as appropriate).

On MacOSX:

```
sudo nano /etc/paths

# Add these two lines (replacing 3.9 with your version of python)
/Library/Frameworks/Python.framework/Versions/Current/bin
/Library/Frameworks/Python.framework/Versions/3.9/bin

# If you don't know your python version do:
python3 --version
```

On Linux (ubuntu 18.04 and 20.04):

The `electrumsv-sdk` script should be found in `$HOME/.local/bin/` dir and is usually on system PATH environment variable by default.

No action usually required.

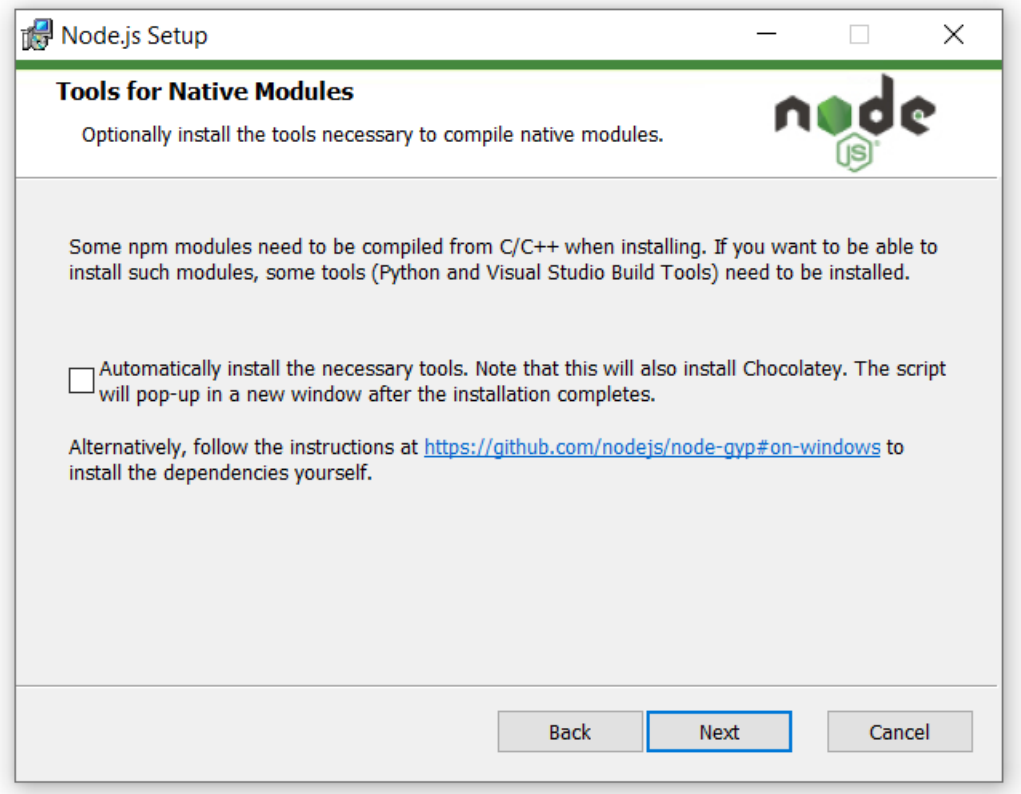
If all goes well you should see this:

Install node.js (only for whatsonchain)

Windows

1. Go to: https://nodejs.org/en/ and install node.js version 12 LTS or later

BUT please leave this box unchecked!



2. configure npm-gyp for python)

open a terminal window and type:

```
> npm config set python C:\Users\<username>\AppData\Local\Programs\Python\Python38\
↪python.exe
```

Linux

```
sudo apt install npm nodejs node-gyp nodejs-dev
```

MacOS

```
brew install node.js
```

Install Postgres

I suggest a system installation of postgres for Windows and MacOS rather than using something like docker (this is because docker installations on windows can wreak havoc with network adaptors and lead to wasted hours for the uninitiated). But docker is always an option if you prefer.

On linux the balance shifts in favour of just using docker in my personal opinion.

Note: It is planned that in a later release we will bundle an embedded postgres and automate the initialisation with:

```
user=mapimaster
password=mapimasterpass
```

Windows or MacOS

Go here: https://www.enterprisedb.com/downloads/postgres-postgresql-downloads Follow the standard instruction steps. Open PgAdmin4 in the browser (on windows) and add the superuser account (enable all user admin privileges):

```
user=mapimaster
password=mapimasterpass
```

Linux

Either follow these instructions here to do a system installation of postgres: https://www.digitalocean.com/community/tutorials/how-to-install-and-use-postgresql-on-ubuntu-18-04

Setup a postgres user:

```
user=mapimaster
password=mapimasterpass
```

Or learn to use docker to pull an official postgres image from: https://hub.docker.com/_/postgres

Don't forget to run it with evironment variables set for:

```
POSTGRES_USER=mapimaster
POSTGRES_PASSWORD=mapimasterpass
```

- The user experience of docker is much better on linux than it is on other platforms

The SDK creates the other needed database entities for you via this user account.

Install components (excluding merchant API)

```
electrumsv-sdk install node
electrumsv-sdk install simple_indexer
electrumsv-sdk install reference_server
electrumsv-sdk install electrumsv
electrumsv-sdk install whatsonchain
electrumsv-sdk install merchant_api
```

Install Merchant API

To install the Merchant API (version 1.3.0)

```
electrumsv-sdk install merchant_api
```

Remember to add Your Node to the Merchant API after starting the service like this:

```
curl --location --request POST 'https://127.0.0.1:5051/api/v1/Node' \
--header 'Content-Type: application/json' \
--header 'Api-Key: apikey' \
--data-raw '{
    "id": "localhost:18332",
    "username": "rpcuser",
    "password": "rpcpassword",
    "remarks": "remarks"
}' --insecure
```

Now you are ready to launch any component! I suggest you now checkout:

- *start command* documentation
- *stop command* documentation
- *reset command* documentation
- *node command* documentation
- *status command* documentation

Release Notes

0.0.X

- Add the simple indexer component.
- Add the reference server component.
- Remove the electrumx component (it is replaced by the simple indexer and the reference server).
- Remove the peer channels component (it is replaced by the reference server).

0.0.37

- Add full and strict type annotations and static analysis to the SDK code-base with mypy
- Fix a bug where boolean values passed to the node rpc via command line were not handled correctly.
- Upgrade Merchant API to version 1.3.0 and remove all SSL configuration requirements.
- Fixed SIGINT (Ctrl + C) on windows such that the ElectrumSV wallet is given a chance to shut down gracefully.
- Fixed an issue where rapid killing and restarting of ElectrumX server resulted in errors due to the port still being on cooldown (TCP TIME_WAIT). Actually fixed by patching ElectrumX to set `SO_REUSEADDR`
- Minor maintenance to docker-images
- Added a new `config` subcommand with –sdk-home-dir option for changing SDK_HOME_DIR location
- Python libraries for python-based components are now installed to SDK_HOME_DIR/python_libs/<component_name>.
- Produced bundled cross-platform python wheels for the leveldb C Extension library 'plyvel': https://pypi.org/project/plyvel-wheels/#files to avoid the need for mac users to manually install leveldb
- The SDK adds the SDK_HOME_DIR/python_libs/<component_name> dir and SDK_HOME_DIR/remote_repos/component_name to PYTHONPATH at runtime (and passes this into the subsequently spawned process - this is fundamentally aimed at avoiding dependency version conflicts and paving the way forward to a one-click-installation with an embedded python.
- Use latest aiorpcx

0.0.36

- Unpin ElectrumX
- Fixed a typo in help menu

0.0.35

- Stop dynamically downgrading requests dependency to cater to electrumsv.

0.0.34

- Updated to only officially support python 3.9
- Pinned ElectrumX version until we can handle the latest aiorpcX library

0.0.33

- Substantial updates to documentation which is now hosted at https://electrumsv-sdk.readthedocs.io/
- Multiple changes to make the SDK more useable as a library (aimed at generating reorg blocks)
- Add commands.py for exposing public methods for using the main SDK entrypoints as a library
- Update 'status' command to allow filtering by component_type or component_id
- Add reorg scripts to contrib/ to document reproducible methodology.
- Add the –deterministic-seed option for electrumsv

- Added the –regtest and –testnet cli option to node and electrumx components
- Simplify logic around mixing –id and <component_type> to always require <component_type> (simplifies the code and user interface). 'reset' and 'stop' commands still allow no args and will reset or stop all components.
- Fix pipeline by installing postgres via homebrew rather than using docker
- Make logging less verbose by default
- Removed electrumsv-indexer (deferred)
- Include electrumsv-server (BIP270 testing server) directory in pypi package and add it as a plugin
- Convert electrumsv-server (BIP270 testing server) from curio to aiohttp web framework
- Add broadcasting feature to the BIP270 testing server (to mAPI endpoints on all public networks & locally on RegTest)

Help

For top level help:

```
> electrumsv-sdk --help
```

For subcommand-specific help:

```
> electrumsv-sdk start --help   # starts a component
> electrumsv-sdk stop --help    # stops a component
> electrumsv-sdk reset --help   # resets a component (what a 'reset' does is defined in␣
↪the plugin)
> electrumsv-sdk node --help    # accesses a running node's RPC API
> electrumsv-sdk status --help  # cli access to an overview of component status
```

Install Command

To install a component (which usually means installing package dependencies for python or node.js applications or compiling/publishing an ASP.NET application):

```
> electrumsv-sdk install <component name>
```

Start Command

The start command is the most feature-rich and launches servers as background processes. dependencies are installed on-demand at run-time (see next):

General usage:

```
> electrumsv-sdk start --id=<unique_id> <component_name>
```

all `--` prefixed flags like `--id`, `--new`, `--repo`, `--inline`, `--background`, `--new-terminal` are optional but if used, must preceed the `component_name`.

Examples

run node + simple_indexer + reference_server + electrumsv:

```
> electrumsv-sdk start node
> electrumsv-sdk start simple_indexer
> electrumsv-sdk start reference_server
> electrumsv-sdk start electrumsv
```

By default, this will launch the servers with the –new-terminal flag (spawning a new console window showing std-out/logging output).

run new instances:

```
> electrumsv-sdk start --new node
```

run new instances with user-defined –id:

```
> electrumsv-sdk start --new --id=mynode2 node
```

specify `--repo` as a local path or remote git url for each component type:

```
> electrumsv-sdk start --repo=G:\electrumsv electrumsv
```

specify `--branch` as either "master" or "features/my-feature-branch"

NOTE1: The sdk tool only handles a single `component_type` at a time (i.e. for the `start`, `stop`, `reset` commands).
NOTE2: The "optional arguments" above come **before** specifying the `component_type` e.g:

```
> electrumsv-sdk start --new --id=myspecialnode node
```

This reserves the capability for arguments to the right hand side of the `component_type` to be fed to the component's underlying commandline interface (if one exists) - this is currently only supported for the electrumsv builtin component:

```
> electrumsv-sdk start --branch=master electrumsv
```

Run inline

To run the server in the current shell (and block until exit or Ctrl + C interrupt):

```
> electrumsv-sdk start --inline <component name>
```

To run the server in the background:

```
> electrumsv-sdk start --background <component name>
```

To run the server in a new terminal window (this is the default if no modifier flag is specified):

```
> electrumsv-sdk start --new-terminal <component name>
```

Stop Command

Stops a component (running server/spawned processes/application - however you prefer to think of it).

General Usage:

```
> electrumsv-sdk stop --id=<unique_id> <component_name>
```

Examples

```
> electrumsv-sdk stop                # no args -> stops all registered components
> electrumsv-sdk stop node           # stops all running ``node`` instances
> electrumsv-sdk stop --id=node1 node    # stops only the component with unique␣
↪identifier == ``node1``
```

Reset Command

General Usage:

```
> electrumsv-sdk reset --id=<unique_id> <component_name>
```

Examples

```
> electrumsv-sdk reset                    # no args -> resets all registered components
> electrumsv-sdk reset node               # resets all running ``node`` instances
> electrumsv-sdk reset --id=node1 node    # resets only the component with unique␣
↪identifier == ``node1``
```

Behaviour for each component

Component Type	Reset Result
node	deletes datadir contents
simple_indexer	deletes database/header files
reference_server	deletes database file
electrumsv	deletes wallet for the datadir and re-creates a new one (worker1.sqlite) with a standard BIP32 'account' (randomly generated seed)
merchant_api	not applicable
whatsonchain	not applicable
whatsonchain_api	not applicable
status_monitor	not applicable

NOTE: The SDK only creates and deletes **a single wallet database (worker1.sqlite) per datadir and there is only one datadir per instance of electrumsv.** Please see "Component ID & Datadirs" for context.

Node Command

Direct access to the standard bitcoin JSON-RPC interface e.g.:

```
> electrumsv-sdk node help
> electrumsv-sdk node generate 10
```

You can access the RPC API on different running node instances like so:

```
> electrumsv-sdk node --id=node2 getinfo
```

Status Command

Returns a json dump of information about components of the SDK:

```
> electrumsv-sdk status
```

Not yet implemented:

```
> electrumsv-sdk status <component_name>
OR
> electrumsv-sdk status --id=<unique id>
```

Which will give filtered results

Overview

Plugin Design Model

As of version 0.0.19 the SDK follows a plugin model whereby there are three layers:

- `'builtin_components/'` (located in site-packages/electrumsv_sdk/builtin_components
- `'user_plugins/'` (located in AppData/local/ElectrumSV-SDK/user_components
- `'electrumsv_sdk_plugins/` (local working directory)

Each layer overrides the one above it if there are any namespace clashes for a given `component_type` The rationale for using a plugin model is aimed at maintainability and extensibility.

To get a feel for the patterns and how to create your own plugin you can look at the `'builtin_components/'` as a template.

Disclaimer: Creating plugins is more the domain of software developers who are expected to have a certain competency level and are willing to work through some technical challenges to get it working.

Most users of this SDK would be expected to merely make use of it for the ease of spinning up 1 or more RegTest instances of bitcoin node(s) +/- manipulating the state of the RegTest environment via the various tools provided out-of-the-box (which may or may not include using the electrumsv wallet GUI or daemon/REST API)

Component ID & Datadirs

The default unique identifier is <component_name> + 1 i.e. node1, simple_indexer1, electrumsv1, whatsonchain1 etc. and by convention we expect plugin creators to strictly tie the default port to this default identifier to keep the SDK approachable for new users. For example these are the default mappings for the builtin components.

The general pattern for datadirs is:

```
~AppData/Local/ElectrumSV-SDK/component_datadirs/<component_name>/<unique id>
```

Or on Linux:

```
~home/.electrumsv-sdk/component_datadirs/<component_name>/<unique id>
```

Default Component IDs and Datadirs

Component Type	Default ID	Port	Datadir Windows	Datadir Linux
node	node1	18332	~AppData/Local/ElectrumSV-SDK/component_datadirs/node/node1	~home/.electrumsv-sdk/component_datadirs/node/node1
simple_indexer	simple_indexer1	49241	~AppData/Local/ElectrumSV-SDK/component_datadirs/simple_indexer/simple_indexer1	~home/.electrumsv-sdk/component_datadirs/simple_indexer/simple_indexer1
reference_server	reference_server1	47124	~AppData/Local/ElectrumSV-SDK/component_datadirs/reference_server/reference_server1	~home/.electrumsv-sdk/component_datadirs/reference_server/reference_serv
electrumsv	electrumsv1	9999	~AppData/Local/ElectrumSV-SDK/component_datadirs/electrumsv/electrumsv1	~home/.electrumsv-sdk/component_datadirs/electrumsv/electrumsv1
whatsonchain	whatson-chain1	3002	Not applicable (no datadir needed for this application)	Not applicable (no datadir needed for this application)
whatson-chain_api	whatson-chain_api1	12121	Not applicable (no datadir needed for this application)	Not applicable (no datadir needed for this application)
merchant_api	merchant_api1	45111	Not applicable (no datadir needed for this application)	Not applicable (no datadir needed for this application)
status_monitor	status_monitor1	5000	~AppData/Local/ElectrumSV-SDK/component_state.json	~home/.electrumsv-sdk/component_state.json

MacOSX datadir location is /Users/runner/.electrumsv-sdk/component_datadirs/<component_name>

Docker

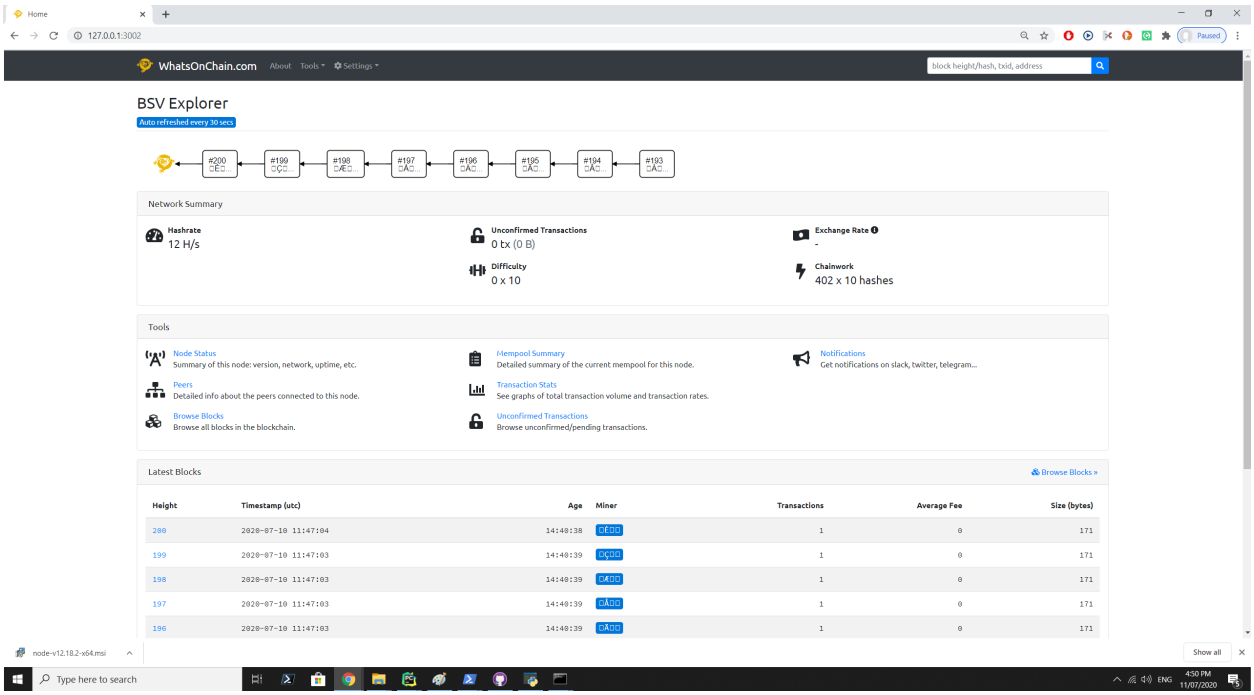
Docker images of each component are available from dockerhub: https://hub.docker.com/u/electrumsvsdk and can be configured via environment variables in the docker-compose (further documentation coming).

These images are created by merely running the SDK component types inside of docker. Perhaps you'd be better to use the docker images from the official sources if they are made available but nevertheless, they are there to use at your own discretion.

Whatsonchain

RegTest Whatsonchain block explorer - a minimally modified fork of https://github.com/waqas64/woc-explorer.

When you have success you should see this:



Note block height=200 because it is on RegTest network.

Run Whatsonchain

install package:

```
electrumsv-sdk start node
electrumsv-sdk start simple_indexer
electrumsv-sdk start reference_server

# Must mine at least one block for whatsonchain to display
electrumsv-sdk node generate 1

electrumsv-sdk start whatsonchain
```

Webpage is viewable at http://127.0.0.1:3002/

You need to be running the node and ideally the simple indexer and the reference server to have anything to look at. Hence why these components are started here.

If you forget a step, the error messages will guide you!

Node

A wrapper for cross-compiled (unofficial) binaries of the latest bitcoin node software for windows, macos x and linux - see: https://github.com/electrumsv/electrumsv-node.

The intended purpose is for local and CI/CD RegTest testing and an accelerated development iteration cycle.

Testnet is also supported (although not necessarily recommended in lieu of traditional methodologies - usually involving a supervisor such as systemd). But in some cases you may just want to run a testnet node on windows or mac in short bursts to test a service against it locally with minimal configuration hassles… In which case using the SDK may be perfectly reasonable for that (It's the easiest way I know of).

See overview section for datadir location.

These are the default settings:

```
regtest=1
server=1
maxstackmemoryusageconsensus=0
excessiveblocksize=10000000000

# TxIndex for indexer
txindex=1

# If --inline flag is set (printtoconsole=1)
printtoconsole=1

# JSON-RPC API
rpcuser=rpcuser
rpcpassword=rpcpassword
rpcport=18332
port=18444

# Rest API (with basic auth the same as the RPC settings)
rest

# ZMQ settings for mAPI
zmqpubrawtx=tcp://127.0.0.1:28332
zmqpubrawblock=tcp://127.0.0.1:28332
zmqpubhashtx=tcp://127.0.0.1:28332
zmqpubhashblock=tcp://127.0.0.1:28332
zmqpubinvalidtx=tcp://127.0.0.1:28332
zmqpubdiscardedfrommempool=tcp://127.0.0.1:28332
zmqpubremovedfrommempoolblock=tcp://127.0.0.1:28332
invalidtxsink=ZMQ
```

ElectrumSV

Automates the running of the ElectrumSV wallet in 'daemon mode' with the example REST API 'dapp' (daemon app) loaded and accessible on localhost (functions like a plugin).

REST API documentation is available here: https://electrumsv.readthedocs.io/en/sv-1.3.11/building-on-electrumsv/rest-api.html

Depends on these components:

- node
- simple_indexer
- reference_server

So start these components first.

To run in daemon mode do:

```
# In a new terminal window
electrumsv-sdk start electrumsv

OR

# Print to current shell
electrumsv-sdk start --inline electrumsv

OR

# As a background process (required for CI/CD or a headless server)
electrumsv-sdk start --background electrumsv
```

You can also run the ElectrumSV GUI in RegTest mode via:

```
# In a new terminal window
electrumsv-sdk start --gui electrumsv

OR

# Print to current shell
electrumsv-sdk start --gui --inline electrumsv

OR

# As a background process (required for CI/CD or a headless server)
electrumsv-sdk start --gui --background electrumsv
```

For testnet run any of the same commands above but including the `--testnet` flag:

```
electrumsv-sdk start --gui --testnet electrumsv
```

Merchant API

This is a wrapper for the Merchant API (a.k.a mAPI) - https://github.com/bitcoin-sv/merchantapi-reference and is an integral part in the shifting paradigm toward "true SPV" applications that can operate efficiently as bitcoin continues to scale.

Unofficial self-contained binaries have been produced for each platform here: https://github.com/electrumsv/electrumsv-mAPI/releases/tag/0.0.1

So ordinarily a system installation of the dotnet-sdk would NOT be required as all dependencies are included in the above .zip files.

However, the path of least resistance for setting up and trusting the SSL certificate is, in my opinion, to use the built-in dotnet-sdk tooling rather than work out how to replicate it via custom shell scripts.

If you have followed the instructions at:

- *start command* documentation

Then you should be ready to go.

Here are the default settings:

```
HTTPSPORT=5051
CERTIFICATEPASSWORD=YourSecurePassword
CERTIFICATEFILENAME=localhost.pfx

QUOTE_EXPIRY_MINUTES=10
ZMQ_CONNECTION_TEST_INTERVAL_SEC=60
RESTADMIN_APIKEY=apikey
DELTA_BLOCKHEIGHT_FOR_DOUBLESPENDCHECK=144
CLEAN_UP_TX_AFTER_DAYS=3
CLEAN_UP_TX_PERIOD_SEC=3600
WIF_PRIVATEKEY=Kz4oGtUm2jbJGmDVHgUxgMppaXNUbcfR3myHxvjVWm7zhrCK3LdW
MINERID_SERVER_URL=
MINERID_SERVER_ALIAS=
MINERID_SERVER_AUTHENTICATION=
NOTIFICATION_NOTIFICATION_INTERVAL_SEC=60
NOTIFICATION_INSTANT_NOTIFICATION_TASKS=2
NOTIFICATION_INSTANT_NOTIFICATIONS_QUEUE_SIZE=1000
NOTIFICATION_MAX_NOTIFICATIONS_IN_BATCH=100
NOTIFICATION_SLOW_HOST_THRESHOLD_MS=1000
NOTIFICATION_INSTANT_NOTIFICATIONS_SLOW_TASK_PERCENTAGE=20
NOTIFICATION_NO_OF_SAVED_EXECUTION_TIMES=10
NOTIFICATION_NOTIFICATIONS_RETRY_COUNT=10
NOTIFICATION_SLOW_HOST_RESPONSE_TIMEOUT_MS=1000
NOTIFICATION_FAST_HOST_RESPONSE_TIMEOUT_MS=2000
```

These are hard-coded at this time but I am open to requests to make them modifiable.

NOTE: The ssl certificate password literally IS the string: "YourSecurePassword". This is hardcoded and your .pfx certificate file must be created with this as the password. This is not for production use and so is a decision made to simplify matters as much as possible.

Usage:

- Make sure you don't forget to add the node to the Merchant API:

Status Monitor (Experimental)

This component is still immature but is designed to be standalone process that exposes an API (and websocket) for status notifications about changes in the state of any other running component.

NOTE: Do not use the REST API - it is not fit for public consumption yet.

But the cli interface can be used for getting basic information about each component (such as the log file location)

Possible states:

- Running
- Stopped
- Failed

There is also a simple backing store (actually just a json file with a file lock protecting it)... which can be accessed any time via the cli:

```
electrumsv-sdk status
```

User-defined Plugins

Documentation not yet available. We recommend you stick to using only the built-in component types at this time unless you are a proficient python developer and don't mind finding your own way.

The built-in plugins are housed here if you're curious.

Logging

Logging output is captured for all components regardless of being run with the –inline, –background or –new-terminal modifier flags. File logging does not need to be configured for this to work as it works by redirecting stdout to file. The location of all file logging is:

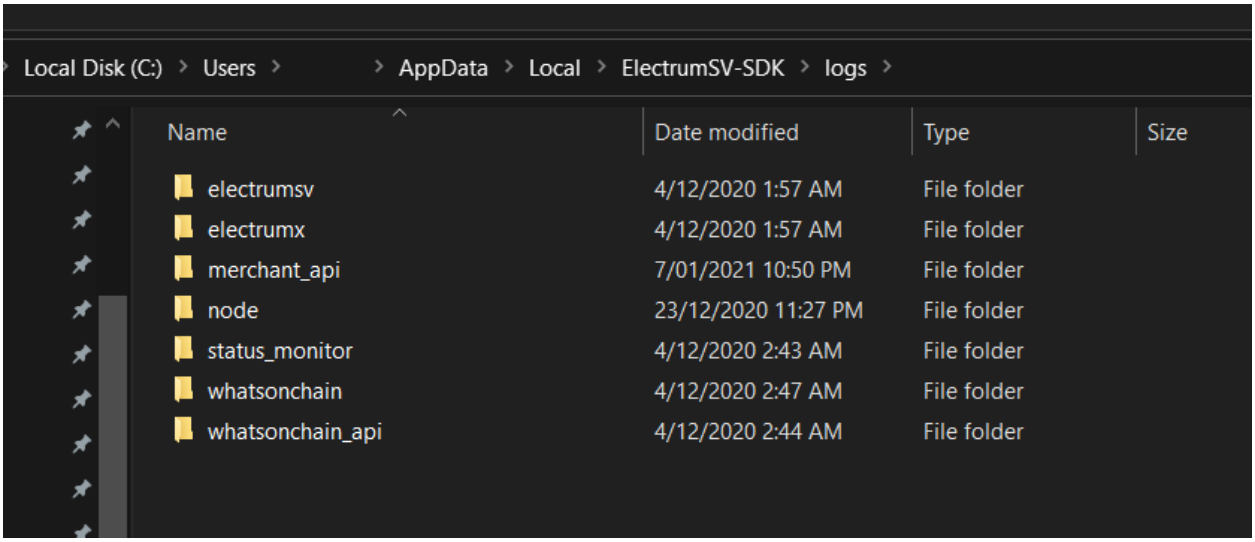
On Windows `C:Users<username>AppDataLocalElectrumSV-SDKlogs`

On Linux `$HOME/.electrumsv-sdk/logs`

On Mac `/Users/runner/.electrumsv-sdk/logs/`

The structure of the logging directory is as follows:

```
~/logs
   /<component_name>
     /component_id
       /<timestamp>.log
       /<timestamp>.log
```



Plugin Design Model

As of version 0.0.19 the SDK follows a plugin model whereby there are three layers:

- `'builtin_components/'` (located in site-packages/electrumsv_sdk/builtin_components
- `'user_plugins/'` (located in AppData/local/ElectrumSV-SDK/user_components
- `'electrumsv_sdk_plugins/` (local working directory)

Each layer overrides the one above it if there are any namespace clashes for a given `component_type` The rationale for using a plugin model is aimed at maintainability and extensibility.

To get a feel for the patterns and how to create your own plugin you can look at the `'builtin_components/'` as a template.

Disclaimer: Creating plugins is more the domain of software developers who are expected to have a certain competency level and are willing to work through some technical challenges to get it working.

Most users of this SDK would be expected to merely make use of it for the ease of spinning up 1 or more RegTest instances of bitcoin node(s) +/- manipulating the state of the RegTest environment via the various tools provided out-of-the-box (which may or may not include using the electrumsv wallet GUI or daemon/REST API)

Docker & Other design rationale

Docker images of each component are available from dockerhub: https://hub.docker.com/u/electrumsvsdk and can be configured via environment variables in the docker-compose (further documentation coming).

These images are created by merely running the SDK component types inside of docker.

Basically you have all options available to you once the plugin is configured.

Testing

For now this is a simple list of things that should be tested, and how they should work. It can be formalised later.

- Ensuring that stopped processes exit gracefully.
  - On Windows, when running *electrumsv-sdk start* with *–background*, *–inline* and *–new-terminal* (the default when none of these are provided) verify that running *electrumsv-sdk stop* allows ElectrumSV to exit cleanly. This can be checked by looking at the logs for ElectrumSV and ensuring it closes down.

    Note that this behaves differently in CI than it does when run locally, so it is not guaranteed that CI-based testing for Windows at least, can ever replace manual testing.