# GDK Documentation

***Release 0.0.63***

**Blockstream Corporation**

**Jun 01, 2023**

# Contents:

# CHAPTER 1

# GDK Functions

int **GA_init**(const GA_json* *config*)

Perform one-time initialization of the library. This call must be made once only before calling any other GDK functions, including any functions called from other threads.

**Parameters**

- **config** – The *Initialization config JSON*.

**Returns** GA_OK or an error code.

**Return type** int

int **GA_get_thread_error_details**(GA_json** *output*)

Get any error details associated with the last error on the current thread.

**Parameters**

- **output** – Destination for the output *Error details JSON* JSON. Returned GA_json should be freed using *GA_destroy_json*.

**Returns** GA_OK or an error code.

**Return type** int

int **GA_create_session**(struct GA_session** *session*)

Create a new session.

**Parameters**

- **session** – Destination for the resulting session. The returned session should be freed using *GA_destroy_session*.

Once created, the caller should set a handler for notifications using *GA_set_notification_handler*, before calling *GA_connect* to connect the session to the network for use. :return: GA_OK or an error code. :rtype: int

int **GA_set_notification_handler**(struct GA_session* *session*, GA_notification_handler *handler*, void* *context*)

Set a handler to be called when notifications arrive for a session.

**Parameters**

- **session** – The session to receive notifications for.
- **handler** – The handler to receive notifications.
- **context** – A context pointer to be passed to the handler.

This call must be initially made on a session before *GA_connect*. *GDK Notifications* may arrive on different threads, so the caller must ensure that shared data is correctly locked within the handler. The GA_json object passed to the caller must be destroyed by the caller using *GA_destroy_json*. Failing to do so will result in memory leaks.

Once a session has been connected, this call can be made only with null values for handler and context. Once this returns, no further notifications will be delivered for the lifetime of the session.

The caller should not call session functions from within the callback handler as this may block the application. :return: GA_OK or an error code. :rtype: int

int **GA_destroy_session**(struct GA_session* *session*)
Free a session allocated by *GA_create_session*.

**Parameters**

- **session** – The session to free.

If the session was connected using *GA_connect* then this call will disconnect it it before destroying it. :return: GA_OK or an error code. :rtype: int

int **GA_connect**(struct GA_session* *session*, const GA_json* *net_params*)
Connect the session to the specified network.

**Parameters**

- **session** – The session to connect.
- **net_params** – The *Connection parameters JSON* of the network to connect to.

This call connects to the remote network services that the session requires, for example the Green servers or Electrum servers. *GA_connect* must be called only once per session lifetime, after *GA_create_session* and before *GA_destroy_session* respectively. Once connected, the underlying network connection of the session can be controlled using *GA_reconnect_hint*.

Once the session is connected, use *GA_register_user* to create a new wallet for the session, or *GA_login_user* to open an existing wallet. :return: GA_OK or an error code. :rtype: int

int **GA_reconnect_hint**(struct GA_session* *session*, const GA_json* *hint*)
Connect or disconnect a sessions underlying network connection.

**Parameters**

- **session** – The session to use.
- **hint** – the *Reconnect JSON* describing the desired reconnection behaviour.

**Returns** GA_OK or an error code.

**Return type** int

int **GA_get_proxy_settings**(struct GA_session* *session*, GA_json** *output*)
Get the current proxy settings for the given session.

**Parameters**

- **session** – The session to use.

- **output** – Destination for the output *Proxy Settings JSON*. Returned GA_json should be freed using `GA_destroy_json`.

**Returns** GA_OK or an error code.

**Return type** int

int **GA_get_wallet_identifier**(const GA_json* *net_params*, const GA_json* *params*, GA_json** *output*)

Compute a hashed wallet identifier from a BIP32 xpub or mnemonic.

The identifier returned is computed from the network combined with the master chain code and public key of the xpub/mnemonic. It can be used as a unique wallet identifier to mitigate privacy risks associated with storing the wallet's xpub.

**Parameters**

- **net_params** – The *Connection parameters JSON* of the network to compute an identifier for.
- **params** – The *Wallet identifier request JSON* to compute an identifier for.
- **output** – Destination for the output JSON. Returned GA_json should be freed using `GA_destroy_json`.

**Returns** GA_OK or an error code.

**Return type** int

int **GA_http_request**(struct GA_session* *session*, const GA_json* *params*, GA_json** *output*)

Make a request to an http server.

**Parameters**

- **session** – The session to use.
- **params** – the *HTTP parameters JSON* of the server to connect to.
- **output** – Destination for the output JSON. Returned GA_json should be freed using `GA_destroy_json`.

**Returns** GA_OK or an error code.

**Return type** int

int **GA_refresh_assets**(struct GA_session* *session*, const GA_json* *params*)

Refresh the sessions internal cache of Liquid asset information.

Each release of GDK comes with a partial list of Liquid assets built-in. This call is used to update it to include all the registered Liquid assets or any new assets that have been registered since the last update.

**Parameters**

- **session** – The session to use.
- **params** – the *Asset parameters JSON* of the server to connect to.

**Returns** GA_OK or an error code.

**Return type** int

int **GA_get_assets**(struct GA_session* *session*, const GA_json* *params*, GA_json** *output*)

Query the Liquid asset registry.

This call is used to retrieve informations about a set of Liquid assets specified by their asset id.

**Parameters**

- **session** – The session to use.
- **params** – the *Get assets parameters JSON* specifying the assets to query.
- **output** – Destination for the output *Asset details JSON*. Returned GA_json should be freed using `GA_destroy_json`.

**Returns** GA_OK or an error code.

**Return type** int

int **GA_validate_asset_domain_name**(struct GA_session* *session*, const GA_json* *params*, GA_json** *output*)

Validate asset domain name. (This is a interface stub)

**Returns** GA_OK or an error code.

**Return type** int

int **GA_validate**(struct GA_session* *session*, GA_json* *details*, struct GA_auth_handler** *call*)

Validate a gdk format JSON document.

**Parameters**

- **session** – The session to use.
- **details** – The *Validate JSON* to validate.
- **call** – Destination for the resulting GA_auth_handler to complete the action. The call handlers result is *Validate Result JSON*. Returned GA_auth_handler should be freed using `GA_destroy_auth_handler`.

---

**Note:** When calling from C/C++, the parameter `details` will be emptied when the call

---

completes. :return: GA_OK or an error code. :rtype: int

int **GA_register_user**(struct GA_session* *session*, GA_json* *hw_device*, GA_json* *details*, struct GA_auth_handler** *call*)

Create a new user wallet.

**Parameters**

- **session** – The session to use.
- **hw_device** – *HW device JSON* or empty JSON for software wallet registration.
- **details** – The *Login credentials JSON* for software wallet registration.
- **call** – Destination for the resulting GA_auth_handler to perform the registration. Returned GA_auth_handler should be freed using `GA_destroy_auth_handler`.

---

**Note:** When calling from C/C++, the parameters `hw_device` and `details` will be emptied when the call

---

completes. :return: GA_OK or an error code. :rtype: int

int **GA_login_user**(struct GA_session* *session*, GA_json* *hw_device*, GA_json* *details*, struct GA_auth_handler** *call*)

Authenticate to a user's wallet.

**Parameters**

- **session** – The session to use.
- **hw_device** – *HW device JSON* or empty JSON for software wallet login.

- **details** – The *Login credentials JSON* for authenticating the user.
- **call** – Destination for the resulting GA_auth_handler to perform the login. Returned GA_auth_handler should be freed using *GA_destroy_auth_handler*.

If a sessions underlying network connection has disconnected and reconnected, the user will need to login again using this function. In this case, the caller can pass empty JSON for both `hw_device` and `details` to login using the previously passed credentials and device.

---

**Note:** When calling from C/C++, the parameters `hw_device` and `details` will be emptied when the call

---

completes. :return: GA_OK or an error code. :rtype: int

int **GA_set_watch_only**(struct GA_session* *session*, const char* *username*, const char* *password*)
Set or disable a watch-only login for a logged-in user wallet.

**Parameters**

- **session** – The session to use.
- **username** – The watch-only username to login with, or a blank string to disable.
- **password** – The watch-only password to login with, or a blank string to disable.

**Returns** GA_OK or an error code.

**Return type** int

int **GA_get_watch_only_username**(struct GA_session* *session*, char** *username*)
Get the current watch-only login for a logged-in user wallet, if any.

**Parameters**

- **session** – The session to use.
- **username** – Destination for the watch-only username. Empty string if not set. Returned string should be freed using *GA_destroy_string*.

**Returns** GA_OK or an error code.

**Return type** int

int **GA_remove_account**(struct GA_session* *session*, struct GA_auth_handler** *call*)
Remove and delete the server history of a wallet.

**Parameters**

- **session** – The session to use.
- **call** – Destination for the resulting GA_auth_handler to perform the removal. Returned GA_auth_handler should be freed using *GA_destroy_auth_handler*.

For multisig Green sessions, removing a wallet removes all history and data associated with the wallet on the server. This operation cannot be undone, and re-registering the wallet will not bring back the wallet's history. For this reason, only empty wallets can be deleted.

For singlesig sessions, removing a wallet removes the locally persisted cache. The actual removal will happen after *GA_destroy_session* is called. :return: GA_OK or an error code. :rtype: int

int **GA_create_subaccount**(struct GA_session* *session*, GA_json* *details*, struct GA_auth_handler** *call*)
Create a subaccount.

**Parameters**

- **session** – The session to use.
- **details** – The subaccount `"name"` (which must not be already used in the wallet) and `"type"` (either `"2of2"`, `"2of2_no_recovery"` or `"2of3"`) must be populated. Type `"2of2_no_recovery"` is available only for Liquid networks and always requires both keys for spending. For type `"2of3"` the caller may provide either `"recovery_mnemonic"` or `"recovery_xpub"` if they do not wish to have a mnemonic passphrase generated automatically. All other fields are ignored.
- **call** – Destination for the resulting GA_auth_handler to perform the creation. Returned GA_auth_handler should be freed using *GA_destroy_auth_handler*. Details of the created subaccount are returned in the `"result"` element of the GA_auth_handler. For 2of3 subaccounts the field `"recovery_xpub"` will be populated, and `"recovery_mnemonic"` will contain the recovery mnemonic passphrase if one was generated. These values must be stored safely by the caller as they will not be returned again by any call such as *GA_get_subaccounts*.

**Note:** When calling from C/C++, the parameter `details` will be emptied when the call completes.

**Returns** GA_OK or an error code.

**Return type** int

int **GA_get_subaccounts**(struct GA_session* *session*, const GA_json* *details*, struct GA_auth_handler** *call*)

Get the user's subaccount details.

**Parameters**

- **session** – The session to use.
- **details** – the *Get Subaccounts parameters JSON* controlling the request.
- **call** – Destination for the resulting GA_auth_handler to perform the creation. The call handlers result is *Subaccounts list JSON*. Returned GA_auth_handler should be freed using *GA_destroy_auth_handler*.

**Returns** GA_OK or an error code.

**Return type** int

int **GA_get_subaccount**(struct GA_session* *session*, uint32_t *subaccount*, struct GA_auth_handler** *call*)

Get subaccount details.

**Parameters**

- **session** – The session to use.
- **subaccount** – The value of `"pointer"` from *Subaccounts list JSON* for the subaccount.
- **call** – Destination for the resulting GA_auth_handler to perform the creation. The call handlers result is *Subaccount JSON*. Returned GA_auth_handler should be freed using *GA_destroy_auth_handler*.

**Returns** GA_OK or an error code.

**Return type** int

int **GA_rename_subaccount**(struct GA_session* *session*, uint32_t *subaccount*, const char* *new_name*)

Rename a subaccount.

**Parameters**

- **session** – The session to use.
- **subaccount** – The value of `"pointer"` from *Subaccounts list JSON* or *Subaccount JSON* for the subaccount to rename.
- **new_name** – New name for the subaccount.

---

**Note:** This call is deprecated and will be removed in a future release. Use `GA_update_subaccount` to rename subaccounts.

---

**Returns** GA_OK or an error code.

**Return type** int

int **GA_update_subaccount**(struct GA_session* *session*, GA_json* *details*, struct GA_auth_handler** *call*)

Update subaccount information.

**Parameters**

- **session** – The session to use.
- **details** – *Subaccount update JSON* giving the details to update.
- **call** – Destination for the resulting GA_auth_handler to complete the action. Returned GA_auth_handler should be freed using `GA_destroy_auth_handler`.

---

**Note:** When calling from C/C++, the parameter `details` will be emptied when the call completes

---

**Returns** GA_OK or an error code.

**Return type** int

int **GA_get_transactions**(struct GA_session* *session*, GA_json* *details*, struct GA_auth_handler** *call*)

Get a page of the user's transaction history.

**Parameters**

- **session** – The session to use.
- **details** – *Transactions details JSON* giving the details to get the transactions for.
- **call** – Destination for the resulting GA_auth_handler to complete the action. Returned GA_auth_handler should be freed using `GA_destroy_auth_handler`.

---

**Note:** When calling from C/C++, the parameter `details` will be emptied when the call completes.

---

---

**Note:** Transactions are returned as *Transaction list JSON* from newest to oldest with up to 30 transactions per page.

---

**Returns** GA_OK or an error code.

**Return type** int

int **GA_get_receive_address**(struct GA_session* *session*, GA_json* *details*, struct GA_auth_handler** *call*)

Get a new address to receive coins to.

**Parameters**

- **session** – The session to use.
- **details** – *Receive address details JSON*.
- **call** – Destination for the resulting GA_auth_handler to complete the action. Returned GA_auth_handler should be freed using `GA_destroy_auth_handler`.

**Note:** When calling from C/C++, the parameter `details` will be emptied when the call completes.

**Returns** GA_OK or an error code.

**Return type** int

int **GA_get_previous_addresses**(struct GA_session* *session*, GA_json* *details*, struct GA_auth_handler** *call*)

Get a page of addresses previously generated for a subaccount.

**Parameters**

- **session** – The session to use.
- **details** – *Previous addresses request JSON* detailing the previous addresses to fetch.
- **call** – Destination for the resulting GA_auth_handler to complete the action. The call handlers result is *Previous addresses JSON*. Returned GA_auth_handler should be freed using `GA_destroy_auth_handler`.

**Note:** When calling from C/C++, the parameter `details` will be emptied when the call completes.

**Note:** Iteration of all addresses is complete when 'last_pointer' is not present in the results.

**Returns** GA_OK or an error code.

**Return type** int

int **GA_get_unspent_outputs**(struct GA_session* *session*, GA_json* *details*, struct GA_auth_handler** *call*)

Get the user's unspent transaction outputs.

**Parameters**

- **session** – The session to use.
- **details** – *Unspent outputs request JSON* detailing the unspent transaction outputs to fetch.
- **call** – Destination for the resulting GA_auth_handler to complete the action. The call handlers result is *Unspent outputs JSON*. Returned GA_auth_handler should be freed using `GA_destroy_auth_handler`.

**Note:** When calling from C/C++, the parameter `details` will be emptied when the call completes.

**Returns** GA_OK or an error code.

**Return type** int

int **GA_get_unspent_outputs_for_private_key**(struct GA_session* *session*, const char* *private_key*, const char* *password*, uint32_t *unused*, GA_json** *utxos*)

Get the unspent transaction outputs associated with a non-wallet private key.

**Parameters**

- **session** – The session to use.
- **key** – The private key in WIF or BIP 38 format.
- **password** – The password the key is encrypted with, if any.
- **unused** – unused, must be 0
- **utxos** – Destination for the returned utxos (same format as *Transaction list JSON*). Returned GA_json should be freed using `GA_destroy_json`.

**Note:** Neither the private key or its derived public key are transmitted.

**Returns** GA_OK or an error code.

**Return type** int

int **GA_set_unspent_outputs_status**(struct GA_session* *session*, GA_json* *details*, struct GA_auth_handler** *call*)

Change the status of a user's unspent transaction outputs.

**Parameters**

- **session** – The session to use.
- **details** – *Unspent ouputs set status JSON* detailing the unspent transaction outputs status to set.
- **call** – Destination for the resulting GA_auth_handler to complete the action. Returned GA_auth_handler should be freed using `GA_destroy_auth_handler`.

**Note:** When calling from C/C++, the parameter `details` will be emptied when the call completes.

**Returns** GA_OK or an error code.

**Return type** int

int **GA_get_transaction_details**(struct GA_session* *session*, const char* *txhash_hex*, GA_json** *transaction*)

Get a transaction's details.

**Parameters**

- **session** – The session to use.

- **txhash_hex** – The transaction hash of the transaction to fetch.
- **transaction** – Destination for the *Transaction details JSON*. Returned GA_json should be freed using `GA_destroy_json`.

**Returns** GA_OK or an error code.

**Return type** int

int **GA_get_balance**(struct GA_session* *session*, GA_json* *details*, struct GA_auth_handler** *call*)
Get the sum of unspent outputs paying to a subaccount.

**Parameters**

- **session** – The session to use.
- **details** – *Unspent outputs request JSON* detailing the unspent transaction outputs to compute the balance from.
- **call** – Destination for the resulting GA_auth_handler to complete the action. Returned GA_auth_handler should be freed using `GA_destroy_auth_handler`.

---

**Note:** When calling from C/C++, the parameter `details` will be emptied when the call completes.

---

**Returns** GA_OK or an error code.

**Return type** int

int **GA_get_available_currencies**(struct GA_session* *session*, GA_json** *currencies*)
Get the list of allowed currencies for all available pricing sources.

**Parameters**

- **session** – The session to use.
- **currencies** – The returned list of *Available currencies JSON*. Returned GA_json should be freed using `GA_destroy_json`.

**Returns** GA_OK or an error code.

**Return type** int

int **GA_convert_amount**(struct GA_session* *session*, const GA_json* *value_details*, GA_json** *output*)
Convert Fiat to BTC and vice-versa.

**Parameters**

- **session** – The session to use.
- **value_details** – *Convert amount JSON* giving the value to convert.
- **output** – Destination for the converted values *Amount JSON*. Returned GA_json should be freed using `GA_destroy_json`.

**Returns** GA_OK or an error code.

**Return type** int

int **GA_encrypt_with_pin**(struct GA_session* *session*, GA_json* *details*, struct GA_auth_handler** *call*)
Encrypt JSON with a server provided key protected by a PIN.

**Parameters**

- **session** – The session to use.

- **details** – The *Encrypt with PIN JSON* to encrypt.
- **call** – Destination for the resulting GA_auth_handler to complete the action. The call handlers result is *Encrypt with PIN Result JSON* which the caller should persist. Returned GA_auth_handler should be freed using *GA_destroy_auth_handler*.

**Note:** When calling from C/C++, the parameter `details` will be emptied when the call completes.

**Returns** GA_OK or an error code.

**Return type** int

int **GA_decrypt_with_pin**(struct GA_session* *session*, GA_json* *details*, struct GA_auth_handler** *call*)

Decrypt JSON with a server provided key protected by a PIN.

**Parameters**

- **session** – The session to use.
- **details** – The *Decrypt with PIN JSON* to decrypt.
- **call** – Destination for the resulting GA_auth_handler to complete the action. The call handlers result is the decrypted JSON. Returned GA_auth_handler should be freed using *GA_destroy_auth_handler*.

**Note:** When calling from C/C++, the parameter `details` will be emptied when the call completes.

**Returns** GA_OK or an error code.

**Return type** int

int **GA_disable_all_pin_logins**(struct GA_session* *session*)

Disable all PIN logins previously set.

After calling this method, the user will not be able to login with PIN from any device that was previously enabled using *GA_encrypt_with_pin*.

**Parameters**

- **session** – The session to use.

**Returns** GA_OK or an error code.

**Return type** int

int **GA_create_transaction**(struct GA_session* *session*, GA_json* *transaction_details*, struct GA_auth_handler** *call*)

Construct a transaction.

**Parameters**

- **session** – The session to use.
- **transaction_details** – The *GDK Create Transaction JSON* for constructing.
- **call** – Destination for the resulting GA_auth_handler to complete the action. Returned GA_auth_handler should be freed using *GA_destroy_auth_handler*.

**Note:** When calling from C/C++, the parameter `transaction_details` will be emptied when the call completes.

**Returns** GA_OK or an error code.

**Return type** int

int **GA_blind_transaction**(struct GA_session* *session*, GA_json* *transaction_details*, struct GA_auth_handler** *call*)

Blind a transaction.

**Parameters**

- **session** – The session to use.
- **transaction_details** – The *GDK Create Transaction JSON* for blinding.
- **call** – Destination for the resulting GA_auth_handler to complete the action. Returned GA_auth_handler should be freed using *GA_destroy_auth_handler*.

**Note:** When calling from C/C++, the parameter `transaction_details` will be emptied when the call completes.

**Returns** GA_OK or an error code.

**Return type** int

int **GA_sign_transaction**(struct GA_session* *session*, GA_json* *transaction_details*, struct GA_auth_handler** *call*)

Sign the user's inputs to a transaction.

**Parameters**

- **session** – The session to use.
- **transaction_details** – The *Sign transaction JSON* for signing, previously returned from GA_create_transaction.
- **call** – Destination for the resulting GA_auth_handler to perform the signing. Returned GA_auth_handler should be freed using *GA_destroy_auth_handler*.

**Note:** When calling from C/C++, the parameter `transaction_details` will be emptied when the call completes.

**Returns** GA_OK or an error code.

**Return type** int

int **GA_create_swap_transaction**(struct GA_session* *session*, const GA_json* *swap_details*, struct GA_auth_handler** *call*)

Construct the initiators side of a swap transaction.

**Parameters**

- **session** – The session to use.
- **swap_details** – The *Create Swap Transaction JSON* for constructing.

- **call** – Destination for the resulting GA_auth_handler to complete the action. The call handlers result is *Create Swap Transaction Result JSON*. Returned GA_auth_handler should be freed using `GA_destroy_auth_handler`.

**Returns** GA_OK or an error code.

**Return type** int

int **GA_complete_swap_transaction**(struct GA_session* *session*, const GA_json* *swap_details*, struct GA_auth_handler** *call*)

Complete construction of the callers side of a swap transaction.

**Parameters**

- **session** – The session to use.
- **swap_details** – The *Complete Swap Transaction JSON* for completing.
- **call** – Destination for the resulting GA_auth_handler to complete the action. The call handlers result is *Complete Swap Transaction Result JSON*. Returned GA_auth_handler should be freed using `GA_destroy_auth_handler`.

**Returns** GA_OK or an error code.

**Return type** int

int **GA_psbt_sign**(struct GA_session* *session*, GA_json* *details*, struct GA_auth_handler** *call*)

Sign one or more of a user's inputs in a PSBT or PSET.

**Parameters**

- **session** – The session to use.
- **details** – The *Sign PSBT JSON* for signing.
- **call** – Destination for the resulting GA_auth_handler to perform the signing. The call handlers result is *Sign PSBT Result JSON*. Returned GA_auth_handler should be freed using `GA_destroy_auth_handler`.

---

**Note:** When calling from C/C++, the parameter `details` will be emptied when the call completes.

---

---

**Note:** EXPERIMENTAL warning: this call may be changed in future releases.

---

**Returns** GA_OK or an error code.

**Return type** int

int **GA_psbt_get_details**(struct GA_session* *session*, GA_json* *details*, struct GA_auth_handler** *call*)

Get wallet details of a PSBT or PSET.

**Parameters**

- **session** – The session to use.
- **details** – The *PSBT Get Details JSON* for getting the wallet details.
- **call** – Destination for the resulting GA_auth_handler to get the wallet details. The call handlers result is *PSBT Get Details Result JSON*. Returned GA_auth_handler should be freed using `GA_destroy_auth_handler`.

**Note:** When calling from C/C++, the parameter `details` will be emptied when the call completes.

**Note:** EXPERIMENTAL warning: this call may be changed in future releases.

**Returns** GA_OK or an error code.

**Return type** int

int **GA_broadcast_transaction**(struct GA_session* *session*, const char* *transaction_hex*, char** *tx_hash*)

Broadcast a fully signed transaction to the P2P network.

**Parameters**

- **session** – The session to use.
- **transaction_hex** – The signed transaction in hex to broadcast.
- **tx_hash** – Destination for the resulting transactions hash. Returned string should be freed using *GA_destroy_string*.

**Returns** GA_OK or an error code.

**Return type** int

int **GA_send_transaction**(struct GA_session* *session*, GA_json* *transaction_details*, struct GA_auth_handler** *call*)

Send a transaction created by GA_create_transaction and signed by GA_sign_transaction.

**Parameters**

- **session** – The session to use.
- **transaction_details** – The *Send transaction JSON* for sending.
- **call** – Destination for the resulting GA_auth_handler to perform the send. Returned GA_auth_handler should be freed using *GA_destroy_auth_handler*.

**Note:** When calling from C/C++, the parameter `transaction_details` will be emptied when the call completes.

**Returns** GA_OK or an error code.

**Return type** int

int **GA_sign_message**(struct GA_session* *session*, GA_json* *details*, struct GA_auth_handler** *call*)

Sign a message with the private key of an address.

**Parameters**

- **session** – The session to use.
- **details** – The *Sign Message JSON* detailing the message to sign and how to sign it.
- **call** – Destination for the resulting GA_auth_handler to perform the signing. The call handlers result is *Sign Message Result JSON*. Returned GA_auth_handler should be freed using *GA_destroy_auth_handler*.

**Note:** When calling from C/C++, the parameter `details` will be emptied when the call completes.

**Returns** GA_OK or an error code.

**Return type** int

int **GA_send_nlocktimes**(struct GA_session* *session*)

Request an email containing the user's nLockTime transactions.

**Parameters**

- **session** – The session to use.

**Returns** GA_OK or an error code.

**Return type** int

int **GA_set_csvtime**(struct GA_session* *session*, GA_json* *locktime_details*, struct GA_auth_handler** *call*)

Set the number of blocks after which CSV transactions become spendable without two factor authentication.

**Parameters**

- **session** – The session to use.
- **locktime_details** – The *Locktime details JSON* for setting the block value.
- **call** – Destination for the resulting GA_auth_handler to change the locktime. Returned GA_auth_handler should be freed using `GA_destroy_auth_handler`.

**Note:** When calling from C/C++, the parameter `locktime_details` will be emptied when the call completes.

**Returns** GA_OK or an error code.

**Return type** int

int **GA_set_nlocktime**(struct GA_session* *session*, const GA_json* *locktime_details*, struct GA_auth_handler** *call*)

Set the number of blocks after which nLockTime transactions become spendable without two factor authentication. When this call succeeds, if the user has an email address associated with the wallet, an updated nlocktimes.zip file will be sent via email.

**Parameters**

- **session** – The session to use.
- **locktime_details** – The *Locktime details JSON* for setting the block value.
- **call** – Destination for the resulting GA_auth_handler to change the locktime. Returned GA_auth_handler should be freed using `GA_destroy_auth_handler`.

**Returns** GA_OK or an error code.

**Return type** int

int **GA_set_transaction_memo**(struct GA_session* *session*, const char* *txhash_hex*, const char* *memo*, uint32_t *memo_type*)

Add a transaction memo to a user's GreenAddress transaction.

**Parameters**

- **session** – The session to use.
- **txhash_hex** – The transaction hash to associate the memo with.
- **memo** – The memo to set.
- **memo_type** – Unused, pass 0.

**Returns** GA_OK or an error code.

**Return type** int

int **GA_get_fee_estimates**(struct GA_session* *session*, GA_json** *estimates*)
Get the current network's fee estimates.

**Parameters**

- **session** – The session to use.
- **estimates** – Destination for the returned *Fee estimates JSON*. Returned GA_json should be freed using *GA_destroy_json*.

The estimates are returned as an array of 25 elements. Each element is an integer representing the fee estimate expressed as satoshi per 1000 bytes. The first element is the minimum relay fee as returned by the network, while the remaining elements are the current estimates to use for a transaction to confirm from 1 to 24 blocks.

**Returns** GA_OK or an error code.

**Return type** int

int **GA_get_credentials**(struct GA_session* *session*, GA_json* *details*, struct GA_auth_handler** *call*)
Get the user's credentials.

**Parameters**

- **session** – The session to use.
- **details** – The *Get credentials JSON* to get the credentials.
- **call** – Destination for the resulting GA_auth_handler to get the user's credentials. The call handlers result is *Login credentials JSON*. Returned GA_auth_handler should be freed using *GA_destroy_auth_handler*.

---

**Note:** When calling from C/C++, the parameter `details` will be emptied when the call completes.

---

**Returns** GA_OK or an error code.

**Return type** int

int **GA_get_system_message**(struct GA_session* *session*, char** *message_text*)
Get the latest un-acknowledged system message.

**Parameters**

- **session** – The session to use.
- **message_text** – The returned UTF-8 encoded message text. Returned string should be freed using *GA_destroy_string*.

---

**Note:** If all current messages are acknowledged, an empty string is returned.

---

**Returns** GA_OK or an error code.

**Return type** int

int **GA_ack_system_message**(struct GA_session* *session*, const char* *message_text*, struct GA_auth_handler** *call*)

Sign and acknowledge a system message.

The message text will be signed with a key derived from the wallet master key and the signature sent to the server.

**Parameters**

- **session** – The session to use.
- **message_text** – UTF-8 encoded message text being acknowledged.
- **call** – Destination for the resulting GA_auth_handler to acknowledge the message. Returned GA_auth_handler should be freed using *GA_destroy_auth_handler*.

**Returns** GA_OK or an error code.

**Return type** int

int **GA_get_twofactor_config**(struct GA_session* *session*, GA_json** *config*)

Get the two factor configuration for the current user.

**Parameters**

- **session** – The session to use.
- **config** – Destination for the returned *Two-Factor config JSON*. Returned GA_json should be freed using *GA_destroy_json*.

**Returns** GA_OK or an error code.

**Return type** int

int **GA_change_settings**(struct GA_session* *session*, GA_json* *settings*, struct GA_auth_handler** *call*)

Change wallet settings.

**Parameters**

- **session** – The session to use.
- **settings** – The new *Settings JSON* values.
- **call** – Destination for the resulting GA_auth_handler. Returned GA_auth_handler should be freed using *GA_destroy_auth_handler*.

---

**Note:** When calling from C/C++, the parameter `settings` will be emptied when the call completes.

---

**Returns** GA_OK or an error code.

**Return type** int

int **GA_get_settings**(struct GA_session* *session*, GA_json** *settings*)

Get current wallet settings.

**Parameters**

- **session** – The session to use.

- **settings** – Destination for the current *Settings JSON*. Returned GA_json should be freed using *GA_destroy_json*.

**Returns** GA_OK or an error code.

**Return type** int

int **GA_destroy_json**(GA_json* *json*)
Free a GA_json object.

**Parameters**

- **json** – GA_json object to free.

**Returns** GA_OK or an error code.

**Return type** int

int **GA_auth_handler_get_status**(struct GA_auth_handler* *call*, GA_json** *output*)
Get the status/result of an action requiring authorization.

**Parameters**

- **call** – The auth_handler whose status is to be queried.
- **output** – Destination for the resulting *Auth handler status JSON*. Returned GA_json should be freed using *GA_destroy_json*.

Methods in the api that may require two factor or hardware authentication to complete return a GA_auth_handler object. This object encapsulates the process of determining whether authentication is required and handling conditions such as re-prompting and re-trying after an incorrect two factor code is entered.

The object acts as a state machine which is stepped through by the caller until the desired action is completed. At each step, the current state can be determined and used to perform the next action required.

Some actions require a sequence of codes and decisions; these are hidden behind the state machine interface so that callers do not need to handle special cases or program their own logic to handle any lower level API differences.

The state machine has the following states, which are returned in the `"status"` element from *GA_auth_handler_get_status*:

- `"done"`: The action has been completed successfully. Any data returned from the action is present in the `"result"` element of the status JSON. The auth_handler object should be destroyed using *GA_destroy_auth_handler* after receiving this status.
- `"error"`: A non-recoverable error occurred performing the action. The associated error message is given in the status element `"error"`. The auth_handler object should be destroyed using *GA_destroy_auth_handler* and the action restarted from scratch if this state is returned.
- `"request_code"`: Two factor authorization is required. The caller should prompt the user to choose a two factor method from the `"methods"` element and call *GA_auth_handler_request_code* with the selected method.
- `"resolve_code"`: A twofactor code from the `"request_code"` step, or data from a hardware device is required. If the status JSON contains *Required Data JSON*, then see *GDK Hardware Wallet Interface* for details. Otherwise, to resolve a twofactor code, the caller should prompt the user to enter the code from the twofactor method chosen in the `"request_code"` step, and pass this code to *GA_auth_handler_resolve_code*.
- `"call"`: Twofactor or hardware authorization is complete and the caller should call *GA_auth_handler_call* to perform the action.

**Returns** GA_OK or an error code.

**Return type** int

int **GA_auth_handler_request_code**(struct GA_auth_handler* *call*, const char* *method*)
Request a two factor authentication code to authorize an action.

**Parameters**

- **call** – The auth_handler representing the action to perform.
- **method** – The selected two factor method to use

**Returns** GA_OK or an error code.

**Return type** int

int **GA_auth_handler_resolve_code**(struct GA_auth_handler* *call*, const char* *code*)
Authorize an action by providing its previously requested two factor authentication code.

**Parameters**

- **call** – The auth_handler representing the action to perform.
- **code** – The two factor authentication code received by the user, or the serialised JSON response for hardware interaction (see *GDK Hardware Wallet Interface*).

**Returns** GA_OK or an error code.

**Return type** int

int **GA_auth_handler_call**(struct GA_auth_handler* *call*)
Perform an action following the completion of authorization.

**Parameters**

- **call** – The auth_handler representing the action to perform.

**Returns** GA_OK or an error code.

**Return type** int

int **GA_destroy_auth_handler**(struct GA_auth_handler* *call*)
Free an auth_handler after use.

**Parameters**

- **call** – The auth_handler to free.

**Returns** GA_OK or an error code.

**Return type** int

int **GA_change_settings_twofactor**(struct GA_session* *session*, const char* *method*, GA_json* *twofactor_details*, struct GA_auth_handler** *call*)
Enable or disable a two factor authentication method.

**Parameters**

- **session** – The session to use
- **method** – The two factor method to enable/disable, i.e. `"email"`, `"sms"`, `"phone"`, `"gauth"`
- **twofactor_details** – The two factor method and associated data such as an email address. *Two-factor detail JSON*
- **call** – Destination for the resulting GA_auth_handler to perform the action Returned GA_auth_handler should be freed using *GA_destroy_auth_handler*.

**Note:** When calling from C/C++, the parameter `twofactor_details` will be emptied when the call completes.

**Returns** GA_OK or an error code.

**Return type** int

int **GA_twofactor_reset**(struct GA_session* *session*, const char* *email*, uint32_t *is_dispute*, struct GA_auth_handler** *call*)

Request to begin the two factor authentication reset process.

Returns the `"twofactor_reset"` portion of *Two-Factor config JSON* in the GA_auth_handler result.

**Parameters**

- **session** – The session to use.
- **email** – The new email address to enable once the reset waiting period expires.
- **is_dispute** – GA_TRUE if the reset request is disputed, GA_FALSE otherwise.
- **call** – Destination for the resulting GA_auth_handler to request the reset. Returned GA_auth_handler should be freed using *GA_destroy_auth_handler*.

**Returns** GA_OK or an error code.

**Return type** int

int **GA_twofactor_undo_reset**(struct GA_session* *session*, const char* *email*, struct GA_auth_handler** *call*)

Undo a request to begin the two factor authentication reset process.

Returns the `"twofactor_reset"` portion of *Two-Factor config JSON* in the GA_auth_handler result.

**Parameters**

- **session** – The session to use.
- **email** – The email address to cancel the reset request for. Must be the email previously passed to *GA_twofactor_reset*.
- **call** – Destination for the resulting GA_auth_handler to request the reset. Returned GA_auth_handler should be freed using *GA_destroy_auth_handler*.

**Note:** Unlike *GA_twofactor_cancel_reset*, this call only removes the reset request associated with the given email. If other emails have requested a reset, the wallet will still remain locked following this call.

**Returns** GA_OK or an error code.

**Return type** int

int **GA_twofactor_cancel_reset**(struct GA_session* *session*, struct GA_auth_handler** *call*)

Cancel all two factor reset requests and unlock the wallet for normal operation.

This call requires authentication using an existing wallet twofactor method.

Returns the `"twofactor_reset"` portion of *Two-Factor config JSON* in the GA_auth_handler result.

**Parameters**

- **session** – The session to use.

- **call** – Destination for the resulting GA_auth_handler to cancel the reset. Returned GA_auth_handler should be freed using *GA_destroy_auth_handler*.

**Returns** GA_OK or an error code.

**Return type** int

int **GA_twofactor_change_limits**(struct GA_session* *session*, GA_json* *limit_details*, struct GA_auth_handler** *call*)

Change twofactor limits settings.

**Parameters**

- **session** – The session to use.
- **limit_details** – Details of the new *Transaction limits JSON*
- **call** – Destination for the resulting GA_auth_handler to perform the change. Returned GA_auth_handler should be freed using *GA_destroy_auth_handler*.

---

**Note:** When calling from C/C++, the parameter `limit_details` will be emptied when the call completes.

---

**Returns** GA_OK or an error code.

**Return type** int

int **GA_bcur_encode**(struct GA_session* *session*, GA_json* *details*, struct GA_auth_handler** *call*)

Encode CBOR into (potentially multi-part) UR-encoding.

**Parameters**

- **session** – The session to use.
- **details** – *BCUR Encode JSON* containing the CBOR data to encode.
- **call** – Destination for the resulting GA_auth_handler to complete the action. The call handlers result is *BCUR Encoded fragments JSON*. Returned GA_auth_handler should be freed using *GA_destroy_auth_handler*.

---

**Note:** When calling from C/C++, the parameter `details` will be emptied when the call completes.

---

**Returns** GA_OK or an error code.

**Return type** int

int **GA_bcur_decode**(struct GA_session* *session*, GA_json* *details*, struct GA_auth_handler** *call*)

Decode (potentially multi-part) UR-encoded data to CBOR.

**Parameters**

- **session** – The session to use.
- **details** – *BCUR Decode JSON* containing the the first URI to decode.
- **call** – Destination for the resulting GA_auth_handler to complete the action. The call handlers result is *BCUR Decoded data JSON*. Returned GA_auth_handler should be freed using *GA_destroy_auth_handler*.

For multi-part data, the call hander will request further parts using `"request_code"` with a method of `"data"`. see: *Auth handler status JSON*.

**Note:** When calling from C/C++, the parameter `details` will be emptied when the call completes.

**Returns** GA_OK or an error code.

**Return type** int

void **GA_destroy_string**(char* *str*)
Free a string returned by the api.

**Parameters**

- **str** – The string to free.

**Returns** GA_OK or an error code.

**Return type** int

int **GA_get_random_bytes**(size_t *num_bytes*, unsigned char* *output_bytes*, size_t *len*)
Get up to 32 random bytes.

Generate up to 32 random bytes using the same strategy as Bitcoin Core code.

**Parameters**

- **output_bytes** – bytes output buffer
- **siz** – Number of bytes to return (max. 32)

**Returns** GA_OK or an error code.

**Return type** int

int **GA_generate_mnemonic**(char** *output*)
Generate a new random BIP 39 mnemonic.

**Parameters**

- **output** – The generated mnemonic phrase. Returned string should be freed using *GA_destroy_string*.

**Returns** GA_OK or an error code.

**Return type** int

int **GA_generate_mnemonic_12**(char** *output*)
Generate a new random 12 word BIP 39 mnemonic.

**Parameters**

- **output** – The generated mnemonic phrase. Returned string should be freed using *GA_destroy_string*.

**Returns** GA_OK or an error code.

**Return type** int

int **GA_validate_mnemonic**(const char* *mnemonic*, uint32_t* *valid*)
Validate a BIP 39 mnemonic.

**Parameters**

- **mnemonic** – The mnemonic phrase

- **valid** – Destination for the result: GA_TRUE if the mnemonic is valid else GA_FALSE

**Returns** GA_OK or an error code.

**Return type** int

int **GA_register_network**(const char* *name*, const GA_json* *network_details*)
Register a network configuration

**Parameters**

- **name** – The name of the network to register
- **network_details** – The *Network JSON* configuration to register

Any existing configuration with the same name is overwritten. If the provided JSON is empty, any existing configuration for the network is removed.

**Returns** GA_OK or an error code.

**Return type** int

int **GA_get_networks**(GA_json** *output*)
Get the available network configurations

**Parameters**

- **output** – Destination for the *Network list JSON* Returned GA_json should be freed using *GA_destroy_json*.

**Returns** GA_OK or an error code.

**Return type** int

int **GA_get_uniform_uint32_t**(uint32_t *upper_bound*, uint32_t* *output*)
Get a uint32_t in the range 0 to (upper_bound - 1) without bias

**Parameters**

- **output** – Destination for the generated uint32_t.

**Returns** GA_OK or an error code.

**Return type** int

CHAPTER 2

# GDK JSON

This section describes the various JSON formats used by the library.

## 2.1 Initialization config JSON

Passed to *GA_init* when initializing the library.

```
{
    "datadir": "/path/to/store/data"
    "tordir": "/path/to/store/tor/data"
    "registrydir": "/path/to/store/registry/data"
    "log_level": "info",
}
```

**datadir** Mandatory. A directory which gdk will use to store encrypted data relating to sessions.

**tordir** An optional directory for tor state data, used when the internal tor implementation is enabled in *Connection parameters JSON*. Note that each process using the library at the same time requires its own distinct directory. If not given, a subdirectory `"tor"` inside `"datadir"` is used.

**registrydir** An optional directory for the registry data, used when the network is liquid based. Note that each process using the library at the same time requires its own distinct directory. If not given, a subdirectory `"registry"` inside `"datadir"` is used.

**log_level** Library logging level, one of `"debug"`, `"info"`, `"warn"`, `"error"`, or `"none"`.

## 2.2 Connection parameters JSON

```
{
   "name": "testnet",
   "proxy": "localhost:9150",
```

(continues on next page)

(continued from previous page)

```
    "use_tor": true,
    "user_agent": "green_android v2.33",
    "spv_enabled": false,
    "cert_expiry_threshold": 1
}
```

**name** The name of the network to connect to. Must match a key from *Network list JSON*.

**proxy** The proxy connection to pass network traffic through, if any.

**use_tor** `true` to enable Tor connections, `false` otherwise. If enabled and a proxy is not given, a Tor connection will be started internally. If a proxy is given and Tor is enabled, the proxy must support resolving `".onion"` domains.

**user_agent** The user agent string to pass to the server for multisig connections.

**spv_enabled** `true` to enable SPV verification for the session, `false` otherwise.

**cert_expiry_threshold** Ignore certificates expiring within this many days from today. Used to pre-empt problems with expiring embedded certificates.

## 2.3 Proxy Settings JSON

Contains the proxy settings in use by a session.

```
{
    "proxy": "localhost:9150",
    "use_tor": true
}
```

**proxy** The proxy connection being used to pass network traffic through, or an empty string.

**use_tor** `true` if Tor is enabled, `false` otherwise.

## 2.4 Login credentials JSON

To authenticate with a hardware wallet, pass empty JSON and provide *HW device JSON*.

To authenticate with a mnemonic and optional password:

```
{
    "mnemonic": "moral lonely ability sail balance simple kid girl inhale master␣
↪dismiss round about aerobic purpose shiver silly happy kitten track kind pattern␣
↪nose noise",
    "password": ""
}
```

Or, with a mnemonic and optional BIP39 passphrase:

```
{
    "mnemonic": "moral lonely ability sail balance simple kid girl inhale master␣
↪dismiss round about aerobic purpose shiver silly happy kitten track kind pattern␣
↪nose noise",
    "bip39_passphrase": ""
}
```

To authenticate with a PIN:

```
{
   "pin": "123456",
   "pin_data": {
       "encrypted_data":
↪"0b39c1e90ca6adce9ff35d1780de74b91d46261a7cbf2b8d2fdc21528c068c8e2b26e3bf3f6a2a992e0e1ecfad0220343
↪",
       "pin_identifier": "38e2f188-b3a8-4d98-a7f9-6c348cb54cfe",
       "salt": "a99/9Qy6P7ON4Umk2FafVQ=="
    }
}
```

**pin** The PIN entered by the user to unlock the wallet.

**pin_data** See *PIN data JSON*.

To authenticate a watch-only user (multisig only):

```
{
   "username": "my_watch_only_username",
   "password": "my_watch_only_password"
}
```

To authenticate a watch-only wallet (singlesig and Bitcoin only):

```
{
   "core_descriptors": ["pkh([00000000/44'/1'/0
↪']tpubDC2Q4xK4XH72J7Lkp6kAvY2Q5x4cxrKgrevkZKC2FwWZ9A9qA5eY6kvv6QDHb6iJtByzoC5J8KZZ29T45CxFz2Gh6m6P
↪0/*"],
}
```

Or alternatively:

```
{
   "slip132_extended_pubkeys": [
↪"tpubDC2Q4xK4XH72J7Lkp6kAvY2Q5x4cxrKgrevkZKC2FwWZ9A9qA5eY6kvv6QDHb6iJtByzoC5J8KZZ29T45CxFz2Gh6m6PQ
↪"],
}
```

The values to use for `"core_descriptors"` and `"slip132_extended_pubkeys"` can be obtained from *GA_get_subaccount*.

## 2.5 HW device JSON

Describes the capabilities of an external signing device.

```
{
   "device": {
      "name": "Ledger",
      "supports_ae_protocol": 0,
      "supports_arbitrary_scripts": true,
      "supports_host_unblinding": false,
      "supports_external_blinding": false,
      "supports_liquid": 1,
      "supports_low_r": false,
```

(continues on next page)

(continued from previous page)

```
    }
}
```

**name** The unique name of the hardware device.

**supports_arbitrary_scripts** True if the device can sign non-standard scripts such as CSV.

**supports_low_r** True if the device can produce low-R ECDSA signatures.

**supports_liquid** 0 if the device does not support Liquid, 1 otherwise.

**supports_host_unblinding** True if the device supports returning the Liquid master blinding key.

**supports_external_blinding** True if the device supports blinding and signing Liquid transactions with outputs that are already blinded from another wallet (e.g. 2-step swaps).

**supports_ae_protocol** See "ae_protocol_support_level" enum in the gdk source for details.

The default for any value not provided is false or 0.

## 2.6 PIN data JSON

Contains the data returned by *GA_encrypt_with_pin*. The caller must persist this data and pass it to *GA_login_user* along with the users PIN in order to allow a PIN login.

```
{
   "encrypted_data":
↪"0b39c1e90ca6adce9ff35d1780de74b91d46261a7cbf2b8d2fdc21528c068c8e2b26e3bf3f6a2a992e0e1ecfad0220343
↪",
   "pin_identifier": "38e2f188-b3a8-4d98-a7f9-6c348cb54cfe",
   "salt": "a99/9Qy6P7ON4Umk2FafVQ=="
}
```

## 2.7 Encrypt with PIN JSON

```
{
   "pin": "...",
   "plaintext": {}
}
```

**pin** The PIN to protect the server provided key.

**plaintext** The json to encrypt. For instance it can be the *Login credentials JSON* with the mnemonic.

## 2.8 Encrypt with PIN Result JSON

```
{
   "pin_data": "...",
}
```

**pin_data** See *PIN data JSON*.

## 2.9 Decrypt with PIN JSON

```
{
   "pin": "...",
   "pin_data": "...",
}
```

**pin** The PIN that protects the server provided key.

**pin_data** See *PIN data JSON*.

## 2.10 Wallet identifier request JSON

Describes the wallet to compute an identifier for using *GA_get_wallet_identifier*. You may pass *Login credentials JSON* to compute an identifier from a mnemonic and optional password, note that PIN or watch-only credentials cannot be used. otherwise, pass the wallets master xpub as follows:

```
{
   "master_xpub":
↪"tpubD8G8MPH9RK9uk4EV97RxhzaY8SJPUWXnViHUwji92i8B7vYdht797PPDrJveeathnKxonJe8SbaScAC1YJ8xAzZbH9Uvy
↪",
}
```

**master_xpub** The base58-encoded BIP32 extended master public key of the wallet.

## 2.11 Get credentials JSON

Accepts an optional password to encrypt the mnemonic.

```
{
   "password": ""
}
```

## 2.12 Subaccount JSON

Describes a subaccount within the users wallet. Returned by *GA_get_subaccount* and as the array elements of *GA_get_subaccounts*.

```
{
  "hidden": false,
  "name": "Subaccount Name",
  "pointer": 0,
  "receiving_id": "GA7ZnuhsieSMNp2XAB3oEyLy75peM",
  "recovery_chain_code": "",
  "recovery_pub_key": "",
  "recovery_xpub": "",
  "required_ca": 0,
  "type": "2of2"
  "bip44_discovered": false
}
```

**hidden** Whether the subaccount is hidden.

**name** The name of the subaccount.

**pointer** The subaccount number.

**receiving_id** The Green receiving ID for the subaccount.

**recovery_chain_code** For `"2of3"` subaccounts, the BIP32 chaincode of the users recovery key.

**recovery_pub_key** For `"2of3"` subaccounts, the BIP32 public key of the users recovery key.

**recovery_xpub** For `"2of3"` subaccounts, the BIP32 xpub of the users recovery key.

**required_ca** For `"2of2_no_recovery"` subaccounts, the number of confidential addresses that the user must upload to the server before transacting.

**type** For multisig subaccounts, one of `"2of2"`, `"2of3"` or `"2of2_no_recovery"`. For singlesig subaccounts, one of `"p2pkh"`, `"p2wpkh"` or `"p2sh-p2wpkh"`.

**bip44_discovered** Singlesig only. Whether or not this subaccount contains at least one transaction.

**user_path** The BIP32 path for this subaccount. This field is only returned by *GA_get_subaccount*.

**core_descriptors** Singlesig only. The Bitcoin Core compatible output descriptors. One for the external chain and one for internal chain (change), for instance `"sh(wpkh(tpubDC2Q4xK4XH72H18SiEV2A6HUwUPLhXiTEQXU35r4a41ZVrUv2cgKUMm2fsKTapi8DH4Y8z0/*))"` `"sh(wpkh(tpubDC2Q4xK4XH72H18SiEV2A6HUwUPLhXiTEQXU35r4a41ZVrUv2cgKUMm2fsKTapi1/*))"` for a `p2sh-p2wpkh` subaccount. This field is only returned by *GA_get_subaccount*.

**slip132_extended_pubkey** Singlesig and Bitcoin only. The extended public key with modified version as specified in SLIP-0132 (xpub, ypub, zpub, tpub, upub, vpub). Use of this value is discouraged and this field might be removed in the future. Callers should use descriptors instead. This field is only returned by *GA_get_subaccount*.

## 2.13 Subaccount update JSON

Describes updates to be made to a subaccount via *GA_update_subaccount*.

```
{
  "hidden": true,
  "name": "New name",
  "subaccount": 1
}
```

**hidden** If present, updates whether the subaccount will be marked hidden.

**name** If present, updates the name of the subaccount.

**subaccount** The subaccount to update.

## 2.14 Subaccounts list JSON

```
{
  "subaccounts": [
    { },
    { }
```

(continues on next page)

(continued from previous page)

```
  ]
}
```

**subaccounts** An array of *Subaccount JSON* elements for each of the users subaccounts.

## 2.15 Transaction list JSON

Describes a users transaction history returned by *GA_get_transactions*.


```
{
  "transactions": [
    {
      "block_height": 2098691,
      "can_cpfp": false,
      "can_rbf": false,
      "created_at_ts": 1633987189032056,
      "fee": 207,
      "fee_rate": 1004,
      "inputs": [
        {
          "address": "",
          "address_type": "csv",
          "is_internal": false,
          "is_output": false,
          "is_relevant": true,
          "is_spent": true,
          "pointer": 287,
          "pt_idx": 0,
          "satoshi": 27071081568,
          "script_type": 15,
          "subaccount": 0,
          "subtype": 0
        }
      ],
      "memo": "",
      "outputs": [
        {
          "address": "2MztTCrvpq73a8homScCo659VADSLEfR2FW",
          "address_type": "csv",
          "is_internal": false,
          "is_output": true,
          "is_relevant": true,
          "is_spent": false,
          "pointer": 288,
          "pt_idx": 0,
          "satoshi": 26970081361,
          "script_type": 15,
          "subaccount": 0,
          "subtype": 51840
        },
        {
          "address": "tb1qt0lenzqp8ay0ryehj7m3wwuds240mzhgdhqp4c",
          "address_type": "",
          "is_internal": false,
```


(continues on next page)

(continued from previous page)


```
          "is_output": true,
          "is_relevant": false,
          "is_spent": false,
          "pointer": 0,
          "pt_idx": 1,
          "satoshi": 101000000,
          "script_type": 11,
          "subaccount": 0,
          "subtype": 0
        }
      ],
      "rbf_optin": false,
      "satoshi": {
        "btc": -101000207
      },
      "spv_verified": "disabled",
      "transaction_vsize": 206,
      "transaction_weight": 824,
      "txhash": "0a934eaa5c8a7c961c1c3aef51a49d11d7d9a04a839620ec6e796156b429c7b4",
      "type": "outgoing"
    }
  ]
}
```


**transactions** Top level container for the users transaction list.

**block_height** The network block height that the transaction was confirmed in, or `0` if the transaction is in the mempool.

**can_cpfp** A boolean indicating whether the user can CPFP the transaction.

**can_rbf** A boolean indicating whether the use can RBF (bump) the transaction fee.

**created_at_ts** The timestamp in microseconds from the Unix epoc when the transaction was seen by gdk or Green servers, or included in a block.

**fee** The BTC or L-BTC network fee paid by the transaction in satoshi.

**fee_rate** The fee rate in satoshi per thousand bytes.

**inputs** See *Transaction list input element*.

**memo** The users memo, if previously set by *GA_set_transaction_memo*.

**outputs** See *Transaction list output element*.

**rbf_optin** A boolean indicating whether the transaction is RBF-enabled.

**satoshi** A map of asset names to the signed satoshi total for that asset in the transaction. Negative numbers represent outgoing amounts, positive incoming.

**spv_verified** The SPV status of the transaction, one of `"in_progress"`, `"verified"`, `"not_verified"`, `"disabled"`, `"not_longest"` or `"unconfirmed"`.

**transaction_vsize** The size of the transaction in vbytes.

**transaction_weight** The weight of the transaction.

**txhash** The txid of the transaction.

**type** One of `"incoming"`, `"outgoing"`, `"mixed"` or `"not unblindable"`.

## 2.16 Transaction list input element

Describes a transaction input in *Transaction list JSON*.


```
{
  "address": "2MxVC4kQTpovRHiEmzd3q7vGtofM8CAijYY",
  "address_type": "csv",
  "is_internal": false,
  "is_output": false,
  "is_relevant": true,
  "is_spent": true,
  "pointer": 287,
  "pt_idx": 0,
  "satoshi": 27071081568,
  "script_type": 15,
  "subaccount": 0,
  "subtype": 0
}
```


**address** For user wallet addresses, the wallet address in base58, bech32 or blech32 encoding.

**addressee** Optional, multisig only. For historical social payments, the account name sent from.

**address_type** For user wallet addresses, One of `"csv"`, `"p2sh"`, `"p2wsh"` (multisig), or `"p2pkh"`, `"p2sh-p2wpkh"`, `"p2wpkh"` (singlesig), indicating the type of address.

**is_internal** Whether or not the user key belongs to the internal chain. Always false for multisig.

**is_output** Always false. Deprecated, will be removed in a future release.

**is_relevant** A boolean indicating whether the input relates to the subaccount the caller passed to *GA_get_transactions*.

**is_spent** Always true. Deprecated, will be removed in a future release.

**pointer** For user wallet addresses, the address number/final number in the address derivation path.

**pt_idx** Deprecated, will be removed in a future release.

**satoshi** The amount of the input in satoshi.

**script_type** Deprecated, will be removed in a future release.

**subaccount** For user wallet addresses, the subaccount this output belongs to, or `0`.

**subtype** For `"address_type" "csv"`, the number of CSV blocks used in the receiving scriptpubkey.

Liquid inputs have additional fields:


```
{
  "amountblinder": "3ad591ed6289ab0a7fa1777197f84a05cd12f651cca831932eaa8a09ac7cc7d2",
  "asset_id": "144c654344aa716d6f3abcc1ca90e5641e4e2a7f633bc09fe3baf64585819a49",
  "asset_tag": "0b5ff0a91c05353089cd40250a2b6c81f09507637d90c37c7e372a8465a4dc0458",
  "assetblinder": "0cd232883f93a3376b88e19a17192495663315a94bd54a24f20299b9af7a696c",
  "commitment": "09f9ac1dfa5042e25a9791fde4aa8292e21c25479eec7783ec5400805a227be256",
  "is_blinded": true,
  "nonce_commitment":
↪"03dcec00304fe2debe04a57f84962966b92db9390b96e9931fef47b002fb265278",
  "previdx": 1,
  "prevpointer": 40,
  "prevsubaccount": null,
  "prevtxhash": "be5ad6db9598873b1443796aa0b34445aa85145586b3355324130c0fd869948f",
```


(continues on next page)

(continued from previous page)


```
  "script": "a914759262b6664d3be92ff41f3a06ade42fa429843087",
}
```


**amountblinder** The hex-encoded amount blinder (value blinding factor, vbf).

**asset_id** The hex-encoded asset id in display format.

**asset_tag** The hex-encoded asset commitment.

**assetblinder** The hex-encoded asset blinder (asset blinding factor, abf).

**commitment** The hex-encoded value commitment.

**is_blinded** A boolean indicating whether or not the input is blinded.

**nonce_commitment** The hex-encoded nonce commitment.

**previdx** The output index of the transaction containing the output representing this input.

**prevpointer** Deprecated, will be removed in a future release.

**prevsubaccount** Deprecated, will be removed in a future release.

**prevtxhash** The txid of the transaction containing the output representing this input.

**script** The scriptpubkey of the output representing this input.

## 2.17 Transaction list output element

Describes a transaction output in *Transaction list JSON*.


```
{
  "address": "2MwdBCwyJnEtp2Bq8CBxyeSi5JWJQ9nXkjj",
  "address_type": "p2wsh",
  "is_internal": false,
  "is_output": true,
  "is_relevant": true,
  "is_spent": true,
  "pointer": 275,
  "pt_idx": 0,
  "satoshi": 1000,
  "script_type": 14,
  "subaccount": 0,
  "subtype": 0
}
```


**address** For user wallet addresses, the wallet address in base58, bech32 or blech32 encoding.

**address_type** For user wallet output addresses, One of `"csv"`, `"p2sh"`, `"p2wsh"` (multisig), or `"p2pkh"`, `"p2sh-p2wpkh"`, `"p2wpkh"` (singlesig), indicating the type of address.

**is_internal** Whether or not the user key belongs to the internal chain. Always false for multisig.

**is_output** Always true. Deprecated, will be removed in a future release.

**is_relevant** A boolean indicating whether the output relates to the subaccount the caller passed to *GA_get_transactions*.

**is_spent** A boolean indicating if this output has been spent.

**pointer** For user wallet addresses, the address number/final number in the address derivation path.

**pt_idx** Deprecated, will be removed in a future release.

**satoshi** The amount of the output in satoshi.

**script_type** Deprecated, will be removed in a future release.

**subaccount** For user wallet addresses, the subaccount this output belongs to, or `0`.

**subtype** For `"address_type" "csv"`, the number of CSV blocks used in the receiving scriptpubkey.

Liquid outputs have additional fields:

```
{
  "amountblinder": "752defd24e9163917aea608a2ff8b77773311a4728551f49761781af9eb4905a",
  "asset_id": "38fca2d939696061a8f76d4e6b5eecd54e3b4221c846f24a6b279e79952850a5",
  "asset_tag": "0ad82ac7489779a5303af3c30b1ec8abd47007f3d5ee01cb1f3b0aac2277a1df23",
  "assetblinder": "d29b09a3f18c7b404ba99338f6427370d0a3b0f6b9591ecf54bce4623a93eb06",
  "blinding_key": "039f2fd9daf37ae24e6a5311ffc18f60aaf3d8adac755c4ee93bf23bbde62071f7
↪",
  "commitment": "0920c8c8ffe7a3529d48947ee1102e3ffbaa62ffa941bc00544d4dd90767426f2d",
  "is_blinded": true,
  "is_confidential": true,
  "nonce_commitment":
↪"0389e67d84f9d04fd163ca540efa599fb51433e7891156c96321f9e85a2687b270",
  "script": "a9144371b94845ee9b316fad126238ccefc05ae74ae587",
  "unconfidential_address": "8ka5DahqHU82oALm372w9rPLZskn4jwpSu"
}
```

**amountblinder** The hex-encoded amount blinder (value blinding factor, vbf).

**asset_id** The hex-encoded asset id in display format.

**asset_tag** The hex-encoded asset commitment.

**assetblinder** The hex-encoded asset blinder (asset blinding factor, abf).

**blinding_key** The blinding public key for the output.

**commitment** The hex-encoded value commitment.

**is_blinded** For user wallet outputs, a boolean indicating whether or not the output is blinded.

**is_confidential** For user wallet outputs, always true when `is_blinded` is true.

**nonce_commitment** The hex-encoded nonce commitment.

**script** For user wallet outputs, the scriptpubkey of this output.

**unconfidential_address** For user wallet outputs, the non-confidential address corresponding to `address`. This is provided for informational purposes only and should not be used to receive.

## 2.18 Transaction details JSON

Contains information about a transaction that may not be associated with the users wallet. Returned by *GA_get_transaction_details*.

```
{
  "transaction":
↪"02000000000101ab0dec345ed48b0761411306eae50f90dd34f3c8598e48f1c3ad324a862bc72b0000000000feffffff02
↪",
  "transaction_locktime": 432,
```

(continues on next page)

(continued from previous page)


```
  "transaction_version": 2,
  "transaction_vsize": 142,
  "transaction_weight": 565,
  "txhash": "dc5c908a6c979211e6482766adb69cbcbe760c92923671f6304d12a3f462a2b0"
}
```


## 2.19 Sign transaction JSON


```
{
"addressees": [
  {
    "address": "2N5xpcfb1TCjncrKABhw2LWPKTSdzVYSy3A",
    "satoshi": 5000
  }
],
"addressees_read_only": false,
"amount_read_only": false,
"available_total": 50000,
"calculated_fee_rate": 1000,
"change_address": {
  "btc": {
    "address": "2N7M3gisUPGmZBeU4WnV9UNkJ9zW2n8bEW7",
    "address_type": "csv",
    "branch": 1,
    "pointer": 3,
    "script":
↪"2103bff5afb55b115068c2f5d906fc97a41ec3b81446f616a31d2304d2cf18c87db9ad2103eaf7e8cf60e89cfb9fe8cabi
↪",
    "script_type": 15,
    "service_xpub":
↪"tpubEAUTpVqYYmSyPnSwSTWrdahLK22WRUkFK66kH348bRawwcBDegdUaucPGU28qS1z9ZiMjH7N2Qqc6HPJiQvekLS8GCpHH0
↪",
    "subaccount": 0,
    "subtype": 51840,
    "user_path": [
      1,
      3
    ]
  }
},
"change_amount": {
  "btc": 44792
},
"change_index": {
  "btc": 0
},
"change_subaccount": 0,
"error": "",
"fee": 208,
"fee_rate": 1000,
"is_redeposit": false,
"is_sweep": false,
"network_fee": 0,
"satoshi": {
```


(continues on next page)

(continued from previous page)


```
  "btc": 5000
},
"send_all": false,
"transaction":
↪"0200000000010135d2bb82963e54a9060567b101760530797590d2b4a636606c4f1e6ac62bed430000000023000000000
↪",
"transaction_locktime": 201,
"transaction_outputs": [
  {
    "address": "2N7M3gisUPGmZBeU4WnV9UNkJ9zW2n8bEW7",
    "address_type": "csv",
    "asset_id": "btc",
    "branch": 1,
    "is_change": true,
    "is_fee": false,
    "pointer": 3,
    "satoshi": 44792,
    "scriptpubkey": "a9149aaba80ae1e733f8fb4034abcb6bd835608a5c9e87",
    "script_type": 15,
    "service_xpub":
↪"tpubEAUTpVqYYmSyPnSwSTWrdahLK22WRUkFK66kH348bRawwcBDegdUaucPGU28qS1z9ZiMjH7N2Qqc6HPJiQvekLS8GCpHH
↪",
    "subaccount": 0,
    "subtype": 51840,
    "user_path": [
      1,
      3
    ]
  },
  {
    "address": "2N5xpcfb1TCjncrKABhw2LWPKTSdzVYSy3A",
    "asset_id": "btc",
    "is_change": false,
    "is_fee": false,
    "satoshi": 5000,
    "script": "a9148b7f781fc9425ffaeafcd4973d3ae1dc9a09d02b87"
  }
],
"transaction_version": 2,
"transaction_vsize": 208,
"transaction_weight": 829,
"used_utxos": [
  {
    "address_type": "csv",
    "block_height": 201,
    "expiry_height": 52041,
    "is_internal": false,
    "pointer": 1,
    "prevout_script":
↪"210375d1b5be6c3f60759fd594b27a05459095ce0f371372d2f0297691c39357a60aad2102129801c6d879b59f27472ba1
↪",
    "pt_idx": 0,
    "satoshi": 50000,
    "script_type": 15,
    "sequence": 4294967293,
    "service_xpub":
↪"tpubEAUTpVqYYmSyPnSwSTWrdahLK22WRUkFK66kH348bRawwcBDegdUaucPGU28qS1z9ZiMjH7N2Qqc6HPJiQvekLS8GCpHH
↪",
```


(continues on next page)

(continued from previous page)


```
    "subaccount": 0,
    "subtype": 51840,
    "txhash": "43ed2bc66a1e4f6c6036a6b4d290757930057601b1670506a9543e9682bbd235",
    "user_path": [
      1,
      1
    ],
    "user_sighash": 1,
    "skip_signing": false,
    "user_status": 0
  }
],
"utxo_strategy": "default",
}
```


To sign with a specific sighash, set `"user_sighash"` for the elements of `"used_utxos"` you wish to sign with a certain sighash, otherwise `SIGHASH_ALL` (`1`) will be used.

Set `"skip_signing"` to `true` for any input in `"used_utxos"` you do not wish to have signed.

## 2.20 Send transaction JSON


```
{
"addressees": [
  {
    "address": "2N5xpcfb1TCjncrKABhw2LWPKTSdzVYSy3A",
    "satoshi": 5000
  }
],
"addressees_read_only": false,
"amount_read_only": false,
"available_total": 50000,
"is_blinded": true,
"calculated_fee_rate": 1230,
"change_address": {
  "btc": {
    "address": "2N7M3gisUPGmZBeU4WnV9UNkJ9zW2n8bEW7",
    "address_type": "csv",
    "branch": 1,
    "pointer": 3,
    "script":
↪"2103bff5afb55b115068c2f5d906fc97a41ec3b81446f616a31d2304d2cf18c87db9ad2103eaf7e8cf60e89cfb9fe8cabi
↪",
    "script_type": 15,
    "service_xpub":
↪"tpubEAUTpVqYYmSyPnSwSTWrdahLK22WRUkFK66kH348bRawwcBDegdUaucPGU28qS1z9ZiMjH7N2Qqc6HPJiQvekLS8GCpHH
↪",
    "subaccount": 0,
    "subtype": 51840,
    "user_path": [
      1,
      3
    ]
  }
```


(continues on next page)

(continued from previous page)


```
},
"change_amount": {
  "btc": 44792
},
"change_index": {
  "btc": 0
},
"change_subaccount": 0,
"error": "",
"fee": 208,
"fee_rate": 1000,
"is_redeposit": false,
"is_sweep": false,
"network_fee": 0,
"satoshi": {
  "btc": 5000
},
"send_all": false,
"transaction":
↪"020000000010135d2bb82963e54a9060567b101760530797590d2b4a636606c4f1e6ac62bed43000000023220020baba
↪",
"transaction_locktime": 201,
"transaction_outputs": [
  {
    "address": "2N7M3gisUPGmZBeU4WnV9UNkJ9zW2n8bEW7",
    "address_type": "csv",
    "asset_id": "btc",
    "branch": 1,
    "is_change": true,
    "is_fee": false,
    "pointer": 3,
    "satoshi": 44792,
    "scriptpubkey": "a9149aaba80ae1e733f8fb4034abcb6bd835608a5c9e87",
    "script_type": 15,
    "service_xpub":
↪"tpubEAUTpVqYYmSyPnSwSTWrdahLK22WRUkFK66kH348bRawwcBDegdUaucPGU28qS1z9ZiMjH7N2Qqc6HPJiQvekLS8GCpHH
↪",
    "subaccount": 0,
    "subtype": 51840,
    "user_path": [
      1,
      3
    ]
  },
  {
    "address": "2N5xpcfb1TCjncrKABhw2LWPKTSdzVYSy3A",
    "asset_id": "btc",
    "is_change": false,
    "is_fee": false,
    "satoshi": 5000,
    "script": "a9148b7f781fc9425ffaeafcd4973d3ae1dc9a09d02b87"
  }
],
"transaction_version": 2,
"transaction_vsize": 169,
"transaction_weight": 675,
"used_utxos": [
```


(continues on next page)

(continued from previous page)


```
  {
    "address_type": "csv",
    "block_height": 201,
    "expiry_height": 52041,
    "is_internal": false,
    "pointer": 1,
    "prevout_script":
↪"210375d1b5be6c3f60759fd594b27a05459095ce0f371372d2f0297691c39357a60aad2102129801c6d879b59f27472ba1
↪",
    "pt_idx": 0,
    "satoshi": 50000,
    "script_type": 15,
    "sequence": 4294967293,
    "service_xpub":
↪"tpubEAUTpVqYYmSyPnSwSTWrdahLK22WRUkFK66kH348bRawwcBDegdUaucPGU28qS1z9ZiMjH7N2Qqc6HPJiQvekLS8GCpHH0
↪",
    "subaccount": 0,
    "subtype": 51840,
    "txhash": "43ed2bc66a1e4f6c6036a6b4d290757930057601b1670506a9543e9682bbd235",
    "user_path": [
      1,
      1
    ],
    "user_status": 0
  }
],
"utxo_strategy": "default",
}
```


## 2.21 Create Swap Transaction JSON

Describes the swap to be created when calling *GA_create_swap_transaction*.


```
{
  "swap_type": "liquidex",
  "input_type": "liquidex_v1",
  "liquidex_v1": {},
  "output_type": "liquidex_v1"
}
```


**swap_type** Pass `"liquidex"` to create the maker's side of a LiquiDEX 2-step swap.

**input_type** Pass `"liquidex_v1"` to pass LiquiDEX version 1 details.

**liquidex_v1** The LiquiDEX v1 specific parameters, see *LiquiDEX Create Swap transaction JSON*. This field must included only if `"input_type"` is `"liquidex_v1"`.

**output_type** Pass `"liquidex_v1"` to return LiquiDEX proposal JSON version 1.

## 2.22 Create Swap Transaction Result JSON

If the `"output_type"` was `"liquidex_v1"` this field is *LiquiDEX Create Swap Transaction Result JSON*.

## 2.23 Complete Swap Transaction JSON

Describes the swap to be completed when calling *GA_complete_swap_transaction*.

```
{
  "swap_type": "liquidex",
  "input_type": "liquidex_v1",
  "liquidex_v1": {},
  "output_type": "transaction",
  "utxos": {},
}
```

**swap_type** Pass `"liquidex"` to complete the taker's side of a LiquiDEX 2-step swap.

**input_type** Pass `"liquidex_v1"` to pass a LiquiDEX proposal JSON version 1.

**liquidex_v1** The LiquiDEX v1 specific parameters, see *LiquiDEX Complete Swap transaction JSON*. This field must included only if `"input_type"` is `"liquidex_v1"`.

**output_type** Pass `"transaction"` to return a transaction JSON that can be passed to *GA_sign_transaction*.

**utxos** Mandatory. The UTXOs to fund the transaction with, *Unspent outputs JSON* as returned by *GA_get_unspent_outputs*. Note that coin selection is not performed on the passed UTXOs. All passed UTXOs of the same asset as the receiving asset id will be included in the transaction.

## 2.24 Complete Swap Transaction Result JSON

If the `"output_type"` was `"transaction"` this field is *Sign transaction JSON*.

## 2.25 Sign PSBT JSON

```
{
  "psbt": "...",
  "utxos": [],
  "blinding_nonces": [],
}
```

**psbt** The PSBT or PSET encoded in base64 format.

**utxos** Mandatory. The UTXOs that should be signed, *Unspent outputs JSON* as returned by *GA_get_unspent_outputs*. UTXOs that are not inputs of the PSBT/PSET can be included. Caller can avoid signing an input by not passing in its UTXO.

**blinding_nonces** For `"2of2_no_recovery"` subaccounts only, the blinding nonces in hex format for all outputs.

## 2.26 Sign PSBT Result JSON

```
{
  "psbt": "...",
  "utxos": [],
}
```

**psbt** The input PSBT or PSET in base64 format, with signatures added for all inputs signed.

**utxos** The UTXOs corresponding to each signed input, in the order they appear in the PSBT transaction.

## 2.27 PSBT Get Details JSON

```
{
  "psbt": "...",
  "utxos": [],
}
```

**psbt** The PSBT or PSET encoded in base64 format.

**utxos** Mandatory. The UTXOs owned by the wallet, *Unspent outputs JSON* as returned by `GA_get_unspent_outputs`. UTXOs that are not inputs of the PSBT/PSET can be included.

## 2.28 PSBT Get Details Result JSON

```
{
  "inputs": [
    {
      "asset_id": "...",
      "satoshi": 0,
      "subaccount": 0,
    },
  ],
  "outputs": [
    {
      "asset_id": "...",
      "satoshi": 0,
      "subaccount": 0,
    },
  ],
}
```

**Note:** Inputs and outputs might have additional fields that might be removed or changed in following releases.

## 2.29 Sign Message JSON

Describes a request for the wallet to sign a given message via `GA_sign_message`.

```
{
  "address": "...",
```

(continues on next page)

(continued from previous page)

```
  "message": "..."
}
```

**address** The address to use for the private key. Must be a P2PKH address, and the address must belong to the wallet.

**message** The message to sign.

## 2.30 Sign Message Result JSON

Returned by *GA_sign_message*.

```
{
  "signature": "..."
}
```

**message** The recoverable signature of the message encoded in base 64.

## 2.31 Fee estimates JSON

```
{"fees":[1000,10070,10070,10070,3014,3014,3014,2543,2543,2543,2543,2543,2543,1499,
↪1499,1499,1499,1499,1499,1499,1499,1499,1499,1499,1499]}
```

## 2.32 Two-Factor config JSON

Describes the wallets enabled two factor methods, current spending limits, and two factor reset status.

```
{
 "all_methods": [
   "email",
   "sms",
   "phone",
   "gauth"
 ],
 "any_enabled": true,
 "email": {
   "confirmed": true,
   "data": "***@@g***",
   "enabled": true
 },
 "enabled_methods": [
   "email"
 ],
 "gauth": {
   "confirmed": false,
   "data": "otpauth://totp/Green%20Bitcoin?secret=IZ3SMET5RDWVUSHB4CPTKUWBJM4HSYHO",
   "enabled": false
 },
 "limits": {
   "bits": "5000.00",
```

(continues on next page)

(continued from previous page)


```
    "btc": "0.00500000",
    "fiat": "0.01",
    "fiat_currency": "EUR",
    "fiat_rate": "1.10000000",
    "is_fiat": false,
    "mbtc": "5.00000",
    "satoshi": 500000,
    "sats": "500000",
    "ubtc": "5000.00"
  },
  "phone": {
    "confirmed": false,
    "data": "",
    "enabled": false
  },
  "sms": {
    "confirmed": false,
    "data": "",
    "enabled": false
  },
  "twofactor_reset": {
    "days_remaining": -1,
    "is_active": false,
    "is_disputed": false
  }
}
```


When the user has a fiat spending limit set instead of BTC, limits are returned as e.g:


```
"limits": {
  "fiat": "0.01",
  "fiat_currency": "EUR",
  "is_fiat": true
}
```


**twofactor_reset/days_remaining** The number of days remaining before the wallets two factor authentication is reset, or -1 if no reset procedure is underway.

**twofactor_reset/is_active** Whether or not the wallet is currently undergoing the two factor reset procedure.

**twofactor_reset/is_disputed** Whether or not the wallet two factor reset procedure is disputed.

## 2.33 BCUR Encode JSON

Contains CBOR data to encode into UR format using *GA_bcur_encode*.


```
{
    "ur_type": "crypto-seed",
    "data": "A20150C7098580125E2AB0981253468B2DBC5202D8641947DA",
    "max_fragment_len": 100
}
```


**ur_type** The type of the CBOR-encoded data.

**data** CBOR-encoded data in hexadecimal format.

**max_fragment_len** The maximum size of each UR-encoded fragment to return.

Where `data` is longer than `max_fragment_len`, the result is a multi-part encoding using approximately 3 times the minimum number of fragments needed to decode the data, split into parts of size `max_fragment_len` or less.

In this case, the caller must provide all retrned parts to any decoder, e.g. by generating an animated QR code from them.

## 2.34 BCUR Encoded fragments JSON

Contains UR format data encoded using *GA_bcur_encode*.

```
{
   "parts": ["ur:crypto-seed/
↪oeadgdstaslplabghydrpfmkbggufgludprfgmaotpiecffltnlpqdenos"]
}
```

**parts** The resulting array of UR-encoded fragments representing the input CBOR.

## 2.35 BCUR Decode JSON

Contains UR encoded data to decode into CBOR using *GA_bcur_decode*.

```
{
   "part": "ur:crypto-seed/oeadgdstaslplabghydrpfmkbggufgludprfgmaotpiecffltnlpqdenos"
}
```

**part** The UR-encoded string for an individual part. For multi-part decoding, the parts can be provided in any order.

## 2.36 BCUR Decoded data JSON

Contains CBOR data decoded from UR format using *GA_bcur_decode*.

```
{
   "ur_type": "crypto-seed",
   "data": "A20150C7098580125E2AB0981253468B2DBC5202D8641947DA"
}
```

**ur-type** The type of the decoded data as specified when it was encoded.

**data** The resulting CBOR-encoded data in hexadecimal format.

## 2.37 Settings JSON

```
{
  "altimeout": 10,
  "csvtime": 51840,
  "nlocktime": 12960,
  "notifications": {
```

(continues on next page)

(continued from previous page)


```
    "email_incoming": true,
    "email_outgoing": true,
    "email_login": true
  },
  "pgp": "",
  "pricing": {
    "currency": "EUR",
    "exchange": "KRAKEN"
  },
  "required_num_blocks": 12,
  "sound": true,
  "unit": "BTC"
}
```


## 2.38 Receive address details JSON


```
{
  "address": "2N2x4EgizS2w3DUiWYWW9pEf4sGYRfo6PAX",
  "address_type": "p2wsh",
  "branch": 1,
  "pointer": 13,
  "script":
↪"52210338832debc5e15ce143d5cf9241147ac0019e7516d3d9569e04b0e18f3278718921025dfaa85d64963252604e1b13
↪",
  "script_type": 14,
  "subaccount": 0,
  "subtype": 0
  "user_path": [1, 13]
}
```


**address** The wallet address in base58, bech32 or blech32 encoding.

**address_type** One of `"csv"`, `"p2sh"`, `"p2wsh"` (multisig), or `"p2pkh"`, `"p2sh-p2wpkh"`, `"p2wpkh"` (singlesig), indicating the type of address.

**branch** Always `1`, used in the address derivation path for subaccounts.

**pointer** The address number/final number in the address derivation path.

**script** The scriptpubkey of the address.

**script_type** Integer representing the type of script.

**subaccount** The subaccount this address belongs to. Matches `"pointer"` from *Subaccounts list JSON* or *Subaccount JSON*.

**subtype** For `"address_type" "csv"`, the number of CSV blocks referenced in `"script"`, otherwise, 0.

**user_path** The BIP32 path for the user key.

For Liquid addresses, the following additional fields are returned:


```
{
  "blinding_key": "02a519491b130082a1abbe17395213b46dae43c3e1c05b7a3dbd2157bd83e88a6e
↪",
  "blinding_script": "a914c2427b28b2796243e1e8ee65be7598d465264b0187",
```


(continues on next page)

(continued from previous page)

```
  "is_blinded": true,
  "unconfidential_address": "XV4PaYgbaJdPnYaJDzE41TpbBF6yBieeyd"
}
```

**blinding_key** The blinding key used to blind this address.

**blinding_script** The script used to generate the blinding key via https://github.com/satoshilabs/slips/blob/master/slip-0077.md.

**is_blinded** Always `true`.

**unconfidential_address** The non-confidential address corresponding to `address`. This is provided for informational purposes only and should not be used to receive.

## 2.39 Previous addresses request JSON

Contains the query parameters for requesting previously generated addresses using *GA_get_previous_addresses*.

```
{
  "subaccount": 0,
  "last_pointer": 0,
}
```

**subaccount** The value of "pointer" from *Subaccounts list JSON* or *Subaccount JSON* for the subaccount to fetch addresses for. Default 0.

**last_pointer** The address pointer from which results should be returned. If this key is not present, the newest generated addresses are returned. If present, the "last_pointer" value from the resulting *Previous addresses JSON* should then be given, until sufficient pages have been fetched or the "last_pointer" key is not present indicating all addresses have been fetched.

**is_internal** Singlesig only. Whether or not the user key belongs to the internal chain.

## 2.40 Previous addresses JSON

Contains a page of previously generated addresses, from newest to oldest.

```
{
  "last_pointer": 2,
  "list": [
    {
      "address": "2N52RVsChsCi439PpJ1Hn8fHCiTrRjcAEiL",
      "address_type": "csv",
      "branch": 1,
      "is_internal": false,
      "pointer": 2,
      "script":
↪"2102df992d7fa8f012d61048349e366f710aa0168a1c08606d7bebb65f980ccf2616ad2102a503dfc70ad1f1a510f7e3c
↪",
      "script_type": 15,
      "subaccount": 0,
      "subtype": 51840,
```

(continues on next page)

(continued from previous page)


```
      "tx_count": 0,
      "user_path": [
        1,
        2
      ],
    },
    {
      "address": "2MzyxeSfodsJkj4YYAyyNpGwqpvdze7qLSf",
      "address_type": "csv",
      "branch": 1,
      "is_internal": false,
      "pointer": 1,
      "script":
↪"2102815c7ba597b1e0f08357ddb346dab3952b2a76e189efc9ebde51ec005df0b41cad210328154df2714de6b15e74033(
↪",
      "script_type": 15,
      "subaccount": 0,
      "subtype": 51840,
      "tx_count": 0,
      "user_path": [
        1,
        1
      ],
    }
  ],
}
```


**last_pointer** If present indicates that there are more addresses to be fetched, and the caller to get the next page should call again *GA_get_previous_addresses* passing this value in *Previous addresses request JSON*. If not present there are no more addresses to fetch.

**list** Contains the current page of addresses in *Receive address details JSON* format.

## 2.41 Unspent outputs request JSON

Describes which unspent outputs to return from *GA_get_unspent_outputs*, or which unspent outputs to include in the balance returned by *GA_get_balance*.


```
{
  "subaccount": 3,
  "num_confs": 0,
  "all_coins": false,
  "expired_at": 99999,
  "confidential": false,
  "dust_limit": 546,
  "sort_by": "newest"
}
```


**subaccount** The subaccount to fetch unspent outputs for.

**num_confs** Pass `0` for unconfirmed UTXOs or `1` for confirmed.

**all_coins** Pass `true` to include UTXOs with status `frozen`. Defaults to `false`.

**expired_at** If given, only UTXOs where two-factor authentication expires by the given block are returned.

**confidential** Pass `true` to include only confidential UTXOs. Defaults to `false`.

**dust_limit** If given, only UTXOs with a value greater than the limit value are returned.

**sort_by** One of `"oldest"`, `"newest"`, `"largest"`, `"smallest"`. Returns the unspent outputs sorted by block height or value respectively. If not given, defaults to `"oldest"` for 2of2 subaccounts and `"largest"` for other subaccount types.

## 2.42 Unspent outputs JSON

Contains the filtered unspent outputs.


```
{
  "unspent_outputs": {
    "btc": [
      {
        "txhash": "09933a297fde31e6477d5aab75f164e0d3864e4f23c3afd795d9121a296513c0",
        "pt_idx": 0,
        "satoshi": 10000,
        "block_height": 1448369,
        "address_type": "p2wsh",
        "is_internal": false,
        "pointer": 474,
        "subaccount": 0,
        "prevout_script":
↪"522102ff54a17dc6efe168673dbf679fe97e06b5cdcaf7dea8ab83dc6732350cd1b4e4210279979574e0743b4659093c00
↪",
        "user_path": [
          1,
          474
        ],
        "public_key":
↪"0279979574e0743b4659093c005256c812f68f512c50d7d1622650b891de2cd61e",
        "expiry_height": 1458369,
        "script_type": 14,
        "user_status": 0,
        "subtype": 0,
      },
    ],
  }
}
```


**txhash** The txid of the transaction.

**pt_idx** The index of the output, the vout.

**satoshi** The amount of the output.

**block_height** The height of the block where the transaction is included. Is 0 if the transaction is unconfirmed.

**address_type** One of `"csv"`, `"p2sh"`, `"p2wsh"` (multisig), or `"p2pkh"`, `"p2sh-p2wpkh"`, `"p2wpkh"` (singlesig), indicating the type of address.

**is_internal** Whether or not the user key belongs to the internal chain. Always false for multisig.

**pointer** The user key number/final number in the derivation path.

**subaccount** The subaccount this output belongs to. Matches `"pointer"` from *Subaccounts list JSON* or *Subaccount JSON*.

**prevout_script** The script being signed, the script code.

**user_path** The BIP32 path for the user key.

**public_key** Singlesig only. The user public key.

**expiry_height** Multisig only. The block height when two-factor authentication expires.

**script_type** Multisig only. Integer representing the type of script.

**user_status** Multisig only. 0 for `"default"` and 1 for `"frozen"`.

**subtype** Multisig only. For `"address_type" "csv"`, the number of CSV blocks referenced in `"script"`, otherwise, 0.

For Liquid instead of having the `"btc"` field, there are (possibly) multiple fields, one for each asset owned, and the keys are the hex-encoded policy ids.

For Liquid the inner maps have additional fields:

```
{
  "amountblinder": "3be117b88ba8284b05b89998bdee1ded8cd5b561ae3d05bcd91d4e8abab2cd47",
  "asset_id": "e4b76d990f27bf6063cb66ff5bbc783d03258a0406ba8ac09abab7610d547e72",
  "asset_tag": "0b103a2d34cf469987dd06937919f9dae8c9856be17c554fd408fdc226b1769e59",
  "assetblinder": "aedb6c37d0ea0bc64fbc7036b52d0a0784da0b1ca90ac918c19ee1025b0c944c",
  "commitment": "094c3f83d5bac22b527ccac141fe04883d79bf04aef10a1dd42f501c5b51318907",
  "is_blinded": true,
  "nonce_commitment":
↪"0211b39afe463473e428cfafd387f9c85b350f440131fad03aa5f4809b6c834f30"
}
```

**amountblinder** The hex-encoded amount blinder (value blinding factor, vbf).

**asset_id** The hex-encoded asset id in display format.

**asset_tag** The hex-encoded asset commitment.

**assetblinder** The hex-encoded asset blinder (asset blinding factor, abf).

**commitment** The hex-encoded value commitment.

**is_blinded** A boolean indicating whether or not the output is blinded.

**nonce_commitment** The hex-encoded nonce commitment.

## 2.43 Unspent ouputs set status JSON

Valid status values are `"default"` for normal behaviour or `"frozen"`. Frozen outputs are hidden from the caller's balance and unspent output requests, are not returned in nlocktime emails, and cannot be spent. An account containing frozen outputs can be deleted, whereas an account with unfrozen outputs can not.

Freezing an output requires two factor authentication. Outputs should only be frozen in response to e.g. a dust attack on the wallet. Once a wallet is deleted, any frozen outputs it contained will be unspendable forever.

---

**Note:** Only outputs of value less that two times the dust limit can be frozen.

---

```
{
  "list" : [
    {
      "txhash": "09933a297fde31e6477d5aab75f164e0d3864e4f23c3afd795d9121a296513c0",
      "pt_idx": 1,
      "user_status": "frozen"
    }
  ]
}
```


## 2.44 Transactions details JSON


```
{"subaccount":0,"first":0,"count":30}
```


## 2.45 Network JSON


```
{
  "address_explorer_url": "",
  "bech32_prefix": "bcrt",
  "bip21_prefix": "bitcoin",
  "csv_buckets": [
    144,
    4320,
    51840
  ],
  "development": true,
  "electrum_tls": false,
  "electrum_url": "localhost:19002",
  "liquid": false,
  "mainnet": false,
  "name": "Localtest",
  "network": "localtest",
  "p2pkh_version": 111,
  "p2sh_version": 196,
  "server_type": "green",
  "service_chain_code":
↪"b60befcc619bb1c212732770fe181f2f1aa824ab89f8aab49f2e13e3a56f0f04",
  "service_pubkey":
↪"036307e560072ed6ce0aa5465534fb5c258a2ccfbc257f369e8e7a181b16d897b3",
  "spv_multi": false,
  "spv_servers": [],
  "spv_enabled": false,
  "tx_explorer_url": "",
  "wamp_cert_pins": [],
  "wamp_cert_roots": [],
  "wamp_onion_url": "",
  "wamp_url": "ws://localhost:8080/v2/ws"
}
```

## 2.46 Network list JSON

Contains a list of all available networks the API can connect to.


```
{
  "all_networks": [
    "mainnet",
    "liquid",
    "testnet"
  ],
  "liquid": { },
  "mainnet": { },
  "testnet": { },
}
```


For each network listed, a *Network JSON* element is present containing the networks information.

## 2.47 Transaction limits JSON


```
{"is_fiat":false,"mbtc":"555"}
{"is_fiat":true,"fiat":"555"}
```


## 2.48 Two-factor detail JSON


```
{"confirmed":true,"data":"mail@example.com","enabled":true}
```


## 2.49 Auth handler status JSON

Describes the status of a GA_auth_handler. Returned by *GA_auth_handler_get_status*.

All status JSON contains a `"name"` element with the name of the handler being invoked.

The remaining data returned depends on the current state of the handler, as follows:

- `"done"`:


```
{
  "status": "done",
  "action": "disable_2fa",
  "result": {}
}
```


**action** The action being processed.

**result** The data returned from the call, if any.

- `"error"`:


```
{
  "status": "error",
  "action": "disable_2fa",
```


(continues on next page)

(continued from previous page)

```
  "error": "Incorrect code"
}
```

**action** The action being processed.

**error** A text description of the error that occured.

- `"call"`:

```
{
  "status": "call",
  "action": "disable_2fa"
}
```

**action** The action being processed.

- `"request_code"`:

```
{
  "status": "request_code",
  "action": "disable_2fa",
  "methods": [ "email", "sms", "phone", "gauth", "telegram" ]
}
```

**action** The action being processed.

**methods** A list of available two factor methods the user has enabled, or the single element `"data"` if the call requires more data to continue.

- `"resolve_code"` (two factor):

```
{
  "status": "resolve_code",
  "action": "disable_2fa",
  "method": "email",
  "auth_data": {},
  "attempts_remaining": "3"
}
```

**action** The action being processed.

**method** The two factor method the user should fetch the code to enter from.

**auth_data** Method-specific ancillary data for resolving the call.

**attempts_remaining** If present, the number of incorrect attempts that can be made before the call fails.

- `"resolve_code"` (hardware wallet/external device):

```
{
  "status": "resolve_code",
  "action": "disable_2fa",
  "required_data": {
      "action": "get_xpubs",
      "device": {}
  }
}
```

**action** The action being processed.

**required_data** Contains the data the HWW must provide, see *GDK Hardware Wallet Interface*.

- `"resolve_code"` (request for additional data):

```
{
  "status": "resolve_code",
  "action": "data",
  "method": "data",
  "auth_data": {}
}
```

**action** Always "data".

**method** Always "data".

**auth_data** Method-specific ancillary data for processing the additional data request.

## 2.50 Reconnect JSON

Controls session and internal Tor instance reconnection behaviour.

```
{
  "hint": "connect",
  "tor_hint": "connect"
}
```

**hint** Optional, must be either `"connect"` or `"disconnect"` if given.

**tor_hint** Optional, must be either `"connect"` or `"disconnect"` if given.

For both hint types, `"disconnect"` will disconnect the underlying network connection used by the session, while `"connect"` will reconnect it. if a hint is not given, no action will be taken for that connection type.

Each session will automatically attempt to reconnect in the background when they detect a disconnection, unless `"disconnect"` is passed to close the connection first. The session will be notified using a *Network notification* when the underlying network connection changes state.

For environments such as mobile devices where networking may become unavailable to the callers application, the network must be disconnected and reconnected using *GA_reconnect_hint* in order for connectivity to be resumed successfully. In particular, when using the built-in Tor implementation to connect, failure to do so may result in Tor failing to connect for the remaining lifetime of the application (this is a Tor limitation).

## 2.51 Convert amount JSON

Amounts to convert are passed with a single key containing the unit value to convert, returning all possible conversion values for that value. See *Amount JSON* for the list of unit values available.

For example, to convert satoshi into all available units:

```
{
  "satoshi": 1120
}
```

If `"fiat_currency"` and `"fiat_rate"` members are provided, the fiat conversion will fall back on these values if no fiat rates are available. Callers can check the `"is_current"` member in the result *Amount JSON* to determine if the fall back values were used.

For example, to convert bits into all available units, with a fiat conversion fallback:

```
{
  "bits": "20344.69",
  "fiat_currency": "USD",
  "fiat_rate": "42161.22"
}
```


## 2.52 Amount JSON


```
{
  "bits": "20344.69",
  "btc": "0.02034469",
  "fiat": "0.02",
  "fiat_currency": "EUR",
  "fiat_rate": "1.10000000",
  "mbtc": "20.34469",
  "satoshi": 2034469,
  "sats": "2034469",
  "subaccount": 0,
  "ubtc": "20344.69"
  "is_current": true
}
```


**fiat_currency** Set to the users fiat currency if available, otherwise an empty string.

**fiat_rate** Set to the users fiat exchange rate if available, otherwise `null`.

**is_current** `true` if the `"fiat_currency"` and `"fiat_rate"` members are current.

## 2.53 Available currencies JSON


```
{
  "all":["AUD","BRL","CAD","CHF","CNY","DKK","EUR","GBP","HKD","IDR","INR","JPY","MXN
↪","MYR","NGN","NOK","NZD","PLN","RUB","SEK","SGD","THB","TRY","USD","ZAR"],
  "per_exchange":{"BITFINEX":["USD"],"BITSTAMP":["USD"],"BTCAVG":[],"BTCCHINA":[],
↪"HUOBI":[],"KIWICOIN":["NZD"],"KRAKEN":["EUR","USD"],"LOCALBTC":["AUD","BRL","CAD",
↪"CHF","CNY","DKK","EUR","GBP","HKD","IDR","INR","JPY","MXN","MYR","NGN","NOK","NZD",
↪"PLN","RUB","SEK","SGD","THB","TRY","USD","ZAR"],"LUNO":["IDR","MYR","NGN","ZAR"],
↪"QUADRIGACX":["CAD","USD"],"TRT":["EUR"]}
}
```


## 2.54 HTTP parameters JSON


```
{
   "accept":"json"
   "method":"GET"
   "urls":[
       "https://assets.blockstream.info/index.json"
       "http://vi5flmr4z3h3luup.onion/index.json"
   ]
   "proxy":"localhost:9150"
```


(continues on next page)

(continued from previous page)

```
   "headers":{"If-Modified-Since":"Mon, 02 Sep 2019 22:39:39 GMT"}
   "timeout":10
}
```

## 2.55 Locktime details JSON

```
{
  "value":65535
}
```

## 2.56 Asset parameters JSON

```
{
   "assets": true,
   "icons": true
}
```

## 2.57 Get assets parameters JSON

Informations about Liquid assets can be obtained by either passing a list of asset ids to query:

```
{
   "assets_id": ["6f0279e9ed041c3d710a9f57d0c02928416460c4b722ae3457a11eec381c526d",
↪"ce091c998b83c78bb71a632313ba3760f1763d9cfcffae02258ffa9865a37bd2"]
}
```

or by specifying one or more of the following attributes:

**names** a list of strings representing asset names;

**tickers** a list of strings representing asset tickers:

**category** must be one of: - `"with_icons"`: only assets that have icons associated to them will be returned; - `"hard_coded"`: only assets bundled in the GDK release will be returned; - `"all"`: all the locally-stored assets and icons will be returned.

Specifying multiple attributes is interpreted as a logical AND. For example, `{"category": "with_icons", "tickers": ["LCAD"]}` will return all the assets with ticker `LCAD` that also have an icon.

## 2.58 Asset details JSON

```
{
   "assets": {
      "6f0279e9ed041c3d710a9f57d0c02928416460c4b722ae3457a11eec381c526d": {
         "asset_id": "6f0279e9ed041c3d710a9f57d0c02928416460c4b722ae3457a11eec381c526d
↪",
         "contract": null,
```

(continues on next page)

(continued from previous page)


```
            "entity": null,
            "issuance_prevout": {
               "txid": "0000000000000000000000000000000000000000000000000000000000000000
↪",
               "vout": 0
            },
            "issuance_txin":{
               "txid": "0000000000000000000000000000000000000000000000000000000000000000
↪",
               "vin": 0
            },
            "issuer_pubkey": "",
            "name": "btc",
            "precision": 8,
            "ticker": "L-BTC",
            "version": 0
         }
      },
      "icons": {
         "6f0279e9ed041c3d710a9f57d0c02928416460c4b722ae3457a11eec381c526d": "BASE64"
      }
}
```


## 2.59 Error details JSON


```
{
   "details":"assertion failure: ../src/ga_session.cpp:rename_subaccount:2166:Unknown␣
↪subaccount"
}
```


## 2.60 Get Subaccounts parameters JSON

Parameters controlling the *GA_get_subaccounts* call.


```
{
   "refresh": false
}
```


**refresh** If set to `true`, subaccounts are re-discovered if appropriate for the session type. Note that this will take significantly more time if set. Defaults to `false`.

## 2.61 Validate JSON

Passed to *GA_validate* to check the validity of gdk input data.

To validate addressees, for example prior to calling *GA_create_transaction*:


```
{
  "addressees": []
}
```

**addressees** An array of *Addressee JSON* elements.

Validation includes that the address is correct and supported by the network, and that the amount given is valid. The given amount in whatever denomination will be converted into `"satoshi"` in the returned addressee. For Liquid, a valid hex `"asset_id"` must be present.

To validate a LiquiDEX version 1 proposal:

```
{
  "liquidex_v1": {
    "proposal": {}
  }
}
```

**liquidex_v1/proposal** The LiquiDEX version 1 proposal to validate.

## 2.62 Validate Result JSON

Returned from *GA_validate* to indicate the validity of the given JSON document.

```
{
  "is_valid": true,
  "errors": [],
  "addressees": {}
}
```

**is_valid** `true` if the JSON is valid, `false` otherwise.

**errors** An array of strings describing each error found in the given document; Empty if `"is_valid"` is `true`.

**addressees** If validating addressees, the given *Addressee JSON* elements with data sanitized and converted if required. For example, BIP21 URLs are converted to addresses, plus amount/asset if applicable.

CHAPTER 3

# GDK Create Transaction JSON

This section details how to create various kinds of transaction using *GA_create_transaction*. Once created, the resulting JSON is generally passed to *GA_sign_transaction* to obtain signatures, then broadcast to the network via *GA_send_transaction* or *GA_broadcast_transaction*.

## 3.1 Overview

The caller passes details about the transaction they would like to construct. The returned JSON contains the resulting transaction and an `"error"` element which is either empty if the call suceeded in creating a valid transaction or contains an error code describing the problem.

Building transactions can be done iteratively, by passing the result of one call into the next after making changes to the returned JSON. This is useful for manual transaction creation as it allows users to see the effect of changes such as different fee rates interactively, and to fix errors on the fly.

When using gdk as a integration solution, *GA_create_transaction* is generally only called once, and if an error occurs the operation is aborted.

Note that the returned JSON will contain additional elements beyond those documented here. The caller should not attempt to change these elements; the documented inputs are the only user-level changes that should be made, and the internal elements may change name or meaning from release to release.

## 3.2 Mandatory and Optional Elements

Only two elements are always mandatory: `"addressees"` and `"utxos"`. A transaction sending some amount from the wallet can be created using e.g:

```
{
  "addressees": [ {} ],
  "utxos": { }
}
```

**addressees** Mandatory. An array of *Addressee JSON* elements, one for each recipient.

**utxos** Mandatory. The UTXOs to fund the transaction with, *Unspent outputs JSON* as returned by `GA_get_unspent_outputs`. Any UTXOs present are candidates for inclusion in the transaction.

Optional elements allow more precise control over the transaction:

**fee_rate** Defaults to the sessions default fee rate setting. The fee rate in satoshi per 1000 bytes to use for fee calculation.

**utxo_strategy** Defaults to `"default"`. Set to `"manual"` for manual UTXO selection.

**send_all** Defaults to `false`. If set to `true`, all given UTXOs will be sent and no change output will be created.

**randomize_inputs** Defaults to `true`. If set to `true`, the order of the used UTXOs in the created transaction is randomized.

**is_partial** Defaults to `false`. Used for creating partial/incomplete transactions such as half-swaps. If set to `true`, no change outputs will be created, fees will not be calculated or deducted from inputs, and the transaction inputs and outputs will not be expected to balance. Sets `"send_all"` and `"randomize_inputs"` to `false`. Consider using `GA_create_swap_transaction` instead of using this element.

**transaction_version** Defaults to `2`. The Bitcoin/Liquid transaction version to use.

**transaction_locktime** Defaults to The current block with occasional random variance for privacy. The transaction level locktime to use.

If you wish to customize a transaction further, consider creating a PSBT/PSET directly from the wallets inputs and using `GA_psbt_sign` to sign it.

## 3.3 Returned metadata

Some data returned when creating a transaction may be read by the user:

**error** If not empty, the error description that prevented the transaction being created.

**addressees_read_only** Whether or not the transaction addressees can be changed. Set to `"true"` for re-deposit, fee bump, CPFP and sweep transactions.

**amount_read_only** Whether or not the send amount can be changed. Set to `"true"` when `"addressees_read_only"` is true and also when `"send_all"` = `"true"` (the amount to send is automatically calculated in this case).

**transaction** The hex-encoded resulting transaction. This may be partially complete or contain dummy data, e.g. missing blinding data or signatures before it is fully completed and signed. The vsize and weight elements described below are adjusted with reasonable estimates for any missing data until the transaction is fully signed.

**transaction_vsize** The expected final vsize of the `"transaction"` in vbytes.

**transaction_weight** The expected final weight of `"transaction"` in segwit weight units.

**calculated_fee_rate** The expected fee rate for the final signed transaction. This may differ slightly from the requested `"fee_rate"` due to variance in the size of witness data such as signatures.

**used_utxos** An array of the `"utxos"` elements that are used by the transaction.

## 3.4 Addressee JSON

Describes an intended recipient for a transaction.

```
{
  "address": "2NFHMw7GbqnQ3kTYMrA7MnHiYDyLy4EQH6b",
  "satoshi": 100000,
  "asset_id": "6f0279e9ed041c3d710a9f57d0c02928416460c4b722ae3457a11eec381c526d"
}
```

**address** Mandatory. The address to send to. All address types for the network are supported. Additionally, BIP 21 URLs are supported along with the Liquid adaptation. Note that BIP 70 payment requests are not supported.

**satoshi** Normally mandatory. The amount to send to the recipient in satoshi. May be ommitted when `"send_all"` is true or when sweeping.

**asset_id** Mandatory for Liquid, must not be present for Bitcoin. The asset to be sent to the recipient, in display hex format.

## 3.5 Coin selection

Callers can control the UTXOs used when creating a transaction. When using `"utxo_strategy": "default"`, UXTOs are selected in order from the `"utxos"` element. The caller can reorder and filter these UTXOs using the query parameters to *GA_get_unspent_outputs* to control which UTXOs are used (and their ordering, if `"randomize_inputs"` is set to `false`).

For finer control, setting `"utxo_strategy"` to `"manual"` allows the UTXOs to be used to be placed in directly into the `"used_utxos"` element by the caller. In this case, `"utxos"` is unused.

The sum of input UTXOs for a given asset must be sufficient to cover the amounts sent to any addressees receiving it, or an error will occur unless `"is_partial"` is `true`. Excess amounts will be returned to the wallet as change, and for `"utxo_strategy": "default"` some UTXOs may be omitted from the created transaction if they are not needed.

Finally, creating a PSBT/PSET and using *GA_psbt_sign* to sign it allows exact specification of all transaction details including UTXOs.

## 3.6 Re-deposit

A re-deposit is just a simple send with the addressee being an address from the users wallet. Setting `"is_redeposit"` to `"true"` when redepositing will also set `send_all"` to `"true"`.

## 3.7 Fee bump

A fee bump or RBF transaction increases the fee rate of an outgoing transaction that the caller has already submitted to the mempool, but which is not yet confirmed.

To create a fee bump, the caller should include the transaction to bump in the `"previous_transaction"` element, and provide the updated fee rate in `"fee_rate"`.

```
{
  "previous_transaction": {},
  "fee_rate": 5000
}
```

**previous_transaction** The transaction to bump, as returned from *Transaction list JSON*.

**fee_rate** The new fee rate in satoshi per 1000 bytes to use for fee calculation. This must be higher than the exiting fee rate in `"previous_transaction"`.

## 3.8 Sweeping

A sweep transaction moves coins from an address with a known private key to another address. Unlike a simple send transaction, the coins to be moved are not associated with the users wallet in any way. Sweeping is typically used to move coins from a paper wallet into the users wallet.

To create a sweep transaction, pass the following json to *GA_create_transaction*:

```
{
  "addressees": [ {} ],
  "private_key": "mrWqGcXTrZpqQvvLwN63amstf8no1W8oo6"
}
```

**addressees** Mandatory. Pass a single *Addressee JSON* element for the coin destination.

**private_key** Mandatory. The private key for the coin to sweep, in either BIP 38 or Wallet Import Format.

Note that `"send_all"` will always be automatically set to `true` for sweep transactions.

It is also possible to send the swept coin to an address that does not belong to the callers wallet. Currently it is not possible to include sweep inputs along with wallet inputs to combine spending.

# CHAPTER 4

# GDK Notifications

This section describes the notifications emitted by the library.

All notifications contain an `"event"` element which describes the type of event being notified. The notification data is available under an element named with the content of the `"event"` element.

## 4.1 Network notification

Notified when the state of a session's underlying network connection changes.

```
{
  "event": "network",
  "network": {
    "wait_ms": 1000,
    "current_state": "disconnected",
    "next_state": "connected"
  }
}
```

**current_state** One of `"connected"` or `"disconnected"`. The current state of the network connection.

**next_state** One of `"connected"` or `"disconnected"`. The next state that the connection will move to. If this value is the same as `"current_state"` then no state change is currently in progress.

**wait_ms** The number of milliseconds before the current state will change to the next state. `0` if the change will happen immediately or no change is due to occur.

## 4.2 Tor notification

Notified when using the built-in Tor connection during connection establishment.

```
{
  "event": "tor",
  "tor": {
    "progress": 20,
    "summary": "Establishing an encrypted directory connection",
    "tag": "onehop_create"
  }
}
```

**tor/progress** An integer from 0-100 indicating the progress percentage.

**tor/summary** A human-readable summary of the current progress step.

**tor/onehop_create** A fixed identifier string for the current progress step.

## 4.3 Settings notification

Notified upon successful authentication. Describes the current wallet settings.

```
{
   "event": "settings",
   "settings": {}
}
```

**settings** Contains the *Settings JSON* of the user.

## 4.4 Two factor reset notification

Notified by multisig sessions upon successful authentication. Describes the current two factor reset status of the wallet.

```
{
   "event": "twofactor_reset",
   "twofactor_reset": {}
}
```

**twofactor_reset** Contains the `"twofactor_reset"` portion of *Two-Factor config JSON*.

## 4.5 Block notification

Notified when a new block is mined by the network.

```
{
   "event": "block",
   "block": {
     "block_hash": "00000000a09b62cc7c076cf8bb25840e67bb5f9f47492f8a82a09105a6aab72d",
     "block_height": 2138311,
     "initial_timestamp": 1489943482,
     "previous_hash":
↪"00000000000000bcf344da3c3d691f5581136bf78c52de4c712949541f0ccf3c"
   }
}
```

**block/block_hash** The hash of the block.

**block/block_height** The height of the block.

**block/initial_timestamp** Multisig only. The time that the users wallet was created, in seconds since the epoc.

**block/previous_hash** The hash of the block prior to this block.

## 4.6 Transaction notification

Notified when a new transaction is received by the wallet.

```
{
  "event":"transaction",
  "transaction":{
      "satoshi":50000,
      "subaccounts":[
          0
      ],
      "txhash":"2bee55e07ab6cc520487f57cb74e87c2960d5f01d291d34f6b395417a276a42c",
      "type":"incoming"
  }
}
```

**transaction/satoshi** Bitcoin only. The net amount of the transaction (always positive).

**transaction/subaccounts** The wallet subaccounts the transaction affects.

**transaction/txhash** The txid of the transaction.

**transaction/type** Bitcoin only. One of `"incoming"`, `"outgoing"` or `"redeposit"`.

## 4.7 Ticker notification

Notified when the user's exchange rate changes.

```
{
  "event": "ticker",
  "ticker": {
    "currency": "NZD",
    "exchange": "KIWICOIN",
    "rate": "44100.84"
  }
}
```

**ticker/currency** The user's chosen fiat currency.

**ticker/exhange** The user's chosen exchange source.

**ticker/rate** The price of 1 Bitcoin expressed in the user's fiat currency, expressed as a floating point string.

# CHAPTER 5

# GDK Hardware Wallet Interface

This section details the format of data requests from hardware wallet interaction during resolution of GA_auth_handler processing when *GA_auth_handler_get_status* returns the status "resolve_code" with a "required_data" element present.

## 5.1 Required Data JSON

Returned as an element "required_data" of *Auth handler status JSON* when data is required from a registered hardware device.

```
{
  "action": "",
  "device": {
  },
}
```

**action** Describes the hardware wallet data requested.

**device** Contains the *HW device JSON* originally registered with the session.

Additional fields will be present depending on the action requested, as follows:

## 5.2 Hardware Get XPubs Action

When "action" is "get_xpubs", this describes a request to compute one or more xpubs from the wallet's master key.

```
{
  "paths": [ [], [ 2147501889 ] ]
}
```

**paths** An array of unsigned 32-bit integer arrays representing each xpub to fetch. The integer values should be interpreted per BIP32, i.e. the topmost bit may be set to indicate a private derivation in the path. An empty array indicates that the top level xpub should be returned.

**Expected response**:

```
{
  "xpubs": [

↪"tpubD8G8MPH9RK9uk4EV97RxhzaY8SJPUWXnViHUwji92i8B7vYdht797PPDrJveeathnKxonJe8SbaScAC1YJ8xAzZbH9Uvyw
↪",

↪"tpubD6NzVbkrYhZ4X9jwmpJxg1kjEJTQgkrnHNEWww2e86X1eUfWu1f7hZpgezAyWUk5zRt4fMPHB33CXrvJSYHHAoVMFXrfxp
↪"
  ]
}
```

**xpubs** An array of base58-encoded BIP32 extended public keys, in the same order as the "paths" elements in the request.

## 5.3 Hardware Get Master Blinding Key Action

When "action" is "get_master_blinding_key", this describes a request to return the wallet's SLIP0077 master blinding key if the user allows this.

**Note:** This action is only returned when using the Liquid network.

No request data is currently associated with this request.

**Expected response**:

```
{
  "master_blinding_key":
↪"512cd6c0b73452a2414e9d86d37cdcc8283b44f0b6dd2b1eec23c59ff12b4f7e5949569b3430220dafce1e0e299a2a6f3
↪"
}
```

**master_blinding_key** The 512-bit or 256-bit master blinding key for the wallet, hex-encoded. If a 256-bit key is returned, it should be the lower 256-bits of the SLIP0021 derived node as specified in https://github.com/satoshilabs/slips/blob/master/slip-0077.md.

**Note:** If the user denies the request to share the key, an empty string should be returned.

## 5.4 Hardware Sign Message Action

When "action" is "sign_message", this describes a request to sign a message using the given path.

```
{
  "message": "A text message to sign",
  "path": [ 1195487518 ],
```

(continues on next page)

(continued from previous page)

```
  "use_ae_protocol": false,
  "create_recoverable_sig": false
}
```

**message** The message to be utf-8 encoded and signed.

**path** The path from the wallet's master key to the key that the message should be signed with.

**use_ae_protocol** `true` if the hardware device advertises Anti-Exfil support and it should be used for signing, `false` otherwise.

**create_recoverable_sig** `true` if the signature to produce should be recoverable. Default `false`.

**Expected response**:

```
{
  "signature":
↪"304402207c673ef4255873cf095016c98c4982cea9a5133060b66a380f1bf3880e54f6c8022056fd731cbd44cd9636621
↪"
}
```

**signature** The ECDSA signature corresponding to the given message. If `"create_recoverable_sig"` is `false` it must use DER encoding, otherwise it must be encoded in hex.

## 5.5 Hardware Get Blinding Public Keys Action

When `"action"` is `"get_blinding_public_keys"`, this describes a request to compute blinding public keys from wallet scripts.

**Note:** This action is only returned when using the Liquid network.

```
{
  "scripts": [ "a91403f650e2434916d5b7f124de8f673442b696282887" ]
}
```

**scripts** An array of hex-encoded scripts for which a blinding key should be generated.

**Expected response**:

```
{
  "public_keys": [ "02045e92b8f68bd066180c05a39969f862a67f4efc8f5d7aeca32c627a463b8f27
↪" ]
}
```

**public_keys** An array of hex-encoded compressed public keys for blinding the given scripts.

## 5.6 Hardware Get Blinding Nonces Action

When `"action"` is `"get_blinding_nonces"`, this describes a request to compute blinding nonces and possibly blinding public keys for the given scripts and shared public keys.

**Note:** This action is only returned when using the Liquid network.

```
{
  "blinding_keys_required": true
  "scripts": [ "a91403f650e2434916d5b7f124de8f673442b696282887" ],
  "public_keys": [ "035f242d49b88ca17948b156263e1f0c86d2cc9e9ff316b058dbbdb351e34bc9aa
↪" ]
}
```

**blinding_keys_required** `true` if the blinding public keys must be returned, `false` otherwise. Blinding public keys are not requested if the master blinding key has previously been given.

**public_keys** An array of hex-encoded compressed shared public keys for computing the nonces.

**scripts** An array of hex-encoded scripts for which a blinding key should be generated and then the nonce computed using the public key given.

**Expected response**:

```
{
  "public_keys": [ "02045e92b8f68bd066180c05a39969f862a67f4efc8f5d7aeca32c627a463b8f27
↪" ]
  "nonces": [ "8d940a5ec4ad122394cd2596ecfbf933a8d8fb0196015cc0a35399e3c326758c" ]
}
```

**public_keys** An array of hex-encoded compressed public keys for blinding the given scripts. Must be present if `"blinding_keys_required"` was `true` in the request, and absent otherwise.

**nonces** An array of hex-encoded 256 bit blinding nonces.

## 5.7 Hardware Get Blinding Factors Action

When `"action"` is `"get_blinding_factors"`, this describes a request to compute asset (ABF) and value (VBF) blinding factors for the given transaction outputs.

**Note:** This action is only returned when using the Liquid network.

```
{
  "is_partial": false,
  "transaction_outputs": [],
  "used_utxos": [
    {
      "txhash": "797c40d53c4a5372303f765281bb107c40ed9618646c46851514ff0483bee894"
      "pt_idx": 2,
    },
    {
      "txhash": "9c7cffca5711968a22b8a03cc6d17224d0d85d884a4d2f638371b6fd6d59afdb"
      "pt_idx": 1,
    }
  ]
}
```

**is_partial** `true` if the transaction to be blinded is incomplete, `false` otherwise.

**transaction_outputs** The transaction output details for the outputs to be blinded, in the format returned by *GA_create_transaction*. Any output with a "blinding_key" key present requires blinding factors to be returned. When "is_partial" is false, the final vbf need not be returned. An empty string should be returned for blinding factors that are not required. It is not an error to provide blinding factors that are not required; they will be ignored.

**used_utxos** An array of prevout txids and their indices, supplied so the request handler can compute hashPrevouts for deterministic blinding.

**Expected response**:

```
{
  "amountblinders": [
    "ce8259bd2e7fa7d6695ade7cf8481919612df28e164a9f89cd96aace69a78bb9",
    ""
  ],
  "assetblinders": [
    "5ca806862967cde0d51950dd4e9add68e7cae8cda928750037fca1fb9cfc9e58",
    "5748810a8d2c4d87ea8c3038fb71369d8d9c85f09cfa4f6412359910fce93616"
  ]
}
```

**amountblinders** An array of hex-encoded, display format value blinding factors (VBFs) to blind the transaction output values. Any non-required values should be returned as empty strings.

**assetblinders** An array of hex-encoded, display format asset blinding factors (ABFs) to blind the transaction output assets. Any non-required values should be returned as empty strings.

CHAPTER 6

# GDK Swap Interface

This section details the Liquid swap protocols supported by GDK and the functions available to run those protocols. Currently there is only one swap protocol supported, *LiquiDEX*.

# CHAPTER 7

# LiquiDEX

**Warning:** Note that in the current version (1) of the protocol if the swap involves inputs or outputs from a `"2of2_no_recovery"` account is not safe to send the proposal directly to untrusted parties (either directly to the Taker, or an untrusted third party). Unless you know what you are doing you should not use this version of the protocol.

LiquiDEX is a 2-step swap protocol, to perform a swap of this kind use `"swap_type" "liquidex"`.

The protocol is started by the Maker, who creates a proposal to swap a certain utxo for a certain amount of another asset. This action is performed using *GA_create_swap_transaction*.

The proposal is then shared with the Taker, who first validates it and verifies that it is a swap that it is willing to accept. This action is performed using *GA_validate*.

If the Taker wants to accept the proposal, they will add more inputs and outputs to fund and balance the transaction. This action is performed using *GA_complete_swap_transaction*.

Note that, unlike *GA_create_swap_transaction*, *GA_complete_swap_transaction* requires the caller to sign the transaction with *GA_sign_transaction*. Moreover for `"2of2_no_recovery"` (AMP) subaccounts, the caller should get a delayed signature for the Maker input.

## 7.1 LiquiDEX Create Swap transaction JSON

`"input_type"` and `"output_type"` must be `"liquidex_v1"`.

```
{
  "swap_type": "liquidex",
  "input_type": "liquidex_v1",
  "output_type": "liquidex_v1",
  "liquidex_v1": {
```

(continues on next page)

(continued from previous page)

```
    "receive": [{
      "asset_id": "ASSET_ID",
      "satoshi": 1
    }],
    "send": [{}],
  },
}
```

**receive/asset_id** The hex-encoded asset id to receive, in display format. This list must have 1 element.

**receive/satoshi** The satoshi amount of the specified asset to receive. This list must have 1 element.

**send** The Maker's UTXO to swap, as returned from *GA_get_unspent_outputs*. This list must have 1 element. The swapped asset will be received to the same subaccount as the utxo provided.

## 7.2 LiquiDEX Create Swap Transaction Result JSON

Returned when `"output_type"` is `"liquidex_v1"`.

```
{
  "liquidex_v1": {
    "proposal": {},
  }
}
```

**proposal** The LiquiDEX version 1 proposal to be shared.

## 7.3 LiquiDEX Complete Swap transaction JSON

`"input_type"` must be `"liquidex_v1"`, and `"output_type"` must be `"transaction"`.

```
{
  "swap_type": "liquidex",
  "input_type": "liquidex_v1",
  "output_type": "transaction",
  "utxos": {},
  "liquidex_v1": {
    "proposals": [{}],
  },
}
```

**proposals** The LiquiDEX version 1 proposals to take.

CHAPTER 8

# Indices and tables

- genindex
- search

# Index

## G