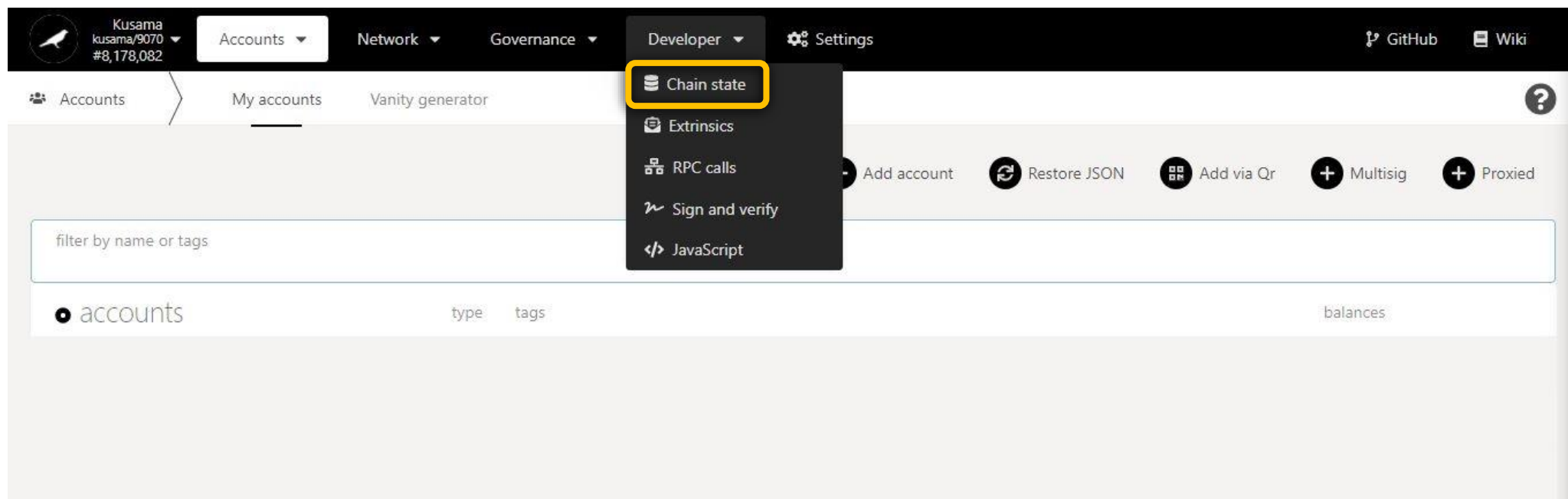


## PART VIII: Developer

### 1. Chain state: Make API calls to query on-chain data.



a) View storage items.

The screenshot shows the top navigation bar of the Polkadot.js Developer interface. The 'Storage' tab is selected, highlighted with a green arrow and a blue callout box that says '1. Click Storage.' Below the navigation bar, a yellow box contains the text 'Interface for accessing information about the state of the chain stored on-chain.' The main content area is enclosed in a green border and contains a 'selected state query' dropdown menu with 'timestamp' selected, a 'now(): u64' input field, and a 'blockhash to query at' input field with '0x...' entered. A '+' button is visible on the right side of the main content area.

The screenshot shows the same Polkadot.js Developer interface, but with the 'Storage' dropdown menu open. A blue callout box with the text '2. Click on the dropdown arrow to view a list of storage items to query.' points to the dropdown arrow. The dropdown menu lists several storage items: 'timestamp', 'auctions', 'authorship', 'babe', 'bagsList', 'balances', and 'bounties'. A green arrow points to the 'balances' item, and a blue callout box with the text '3. Select one storage item.' points to it. The 'balances' item is highlighted in the dropdown menu.

Kusama kusama/9122 #10,317,289 Accounts Network Governance Developer Settings GitHub Wiki

Chain state Storage Constants Raw storage

4. Click on the **dropdown arrow** to view a list of methods for building queries.

selected state query ?  
balances

AccountId32  
ANAELLE LTD@KSM

blockhash to query at  
0x...

5. Select one **method**.

account(AccountId32): PalletBalancesAccountData	The balance of an account.	+
<b>account(AccountId32): PalletBalancesAccountData</b>	<b>The balance of an account.</b>	
locks(AccountId32): Vec<PalletBalancesBalanceLock>	Any liquidity locks on some account balances.	
palletVersion(): u16	Returns the current pallet version from storage	
reserves(AccountId32): Vec<PalletBalancesReserveData>	Named reserves on some account balances.	
storageVersion(): PalletBalancesReleases	Storage version of the pallet.	
totalIssuance(): u128	The total units issued in the system.	

Kusama kusama/9122 #10,317,305 Accounts Network Governance Developer Settings GitHub Wiki

Chain state Storage Constants Raw storage

6. Click on the **+** button to submit your query.

selected state query ?  
balances

totalIssuance(): u128

blockhash to query at  
0x...

7. The information requested is now available on the interface!

balances.totalIssuance: u128  
11,729,681,057,947,515,284

## GUIDE TO POLKADOT-JS – PART VIII: Developer

Version 2.0

Kusama kusama/9122 #10,317,355

Accounts Network Governance Developer Settings GitHub Wiki

Chain state Storage Constants Raw storage

balances parasShared phragmenElection proxy recovery registrar scheduler

totalIssuance(): u128 The total units issued in the system.

8. You can continue to build different queries by selecting another **storage item**...

Kusama kusama/9122 #10,317,360

Accounts Network Governance Developer Settings GitHub Wiki

Chain state Storage Constants Raw storage

selected state query ? proxy

AccountId32 ANAELLE LTD@KSM

announcements(AccountId32): (Vec<PalletProxyAnnouncement>,u128) The announcements made by the proxy (key).

announcements(AccountId32): (Vec<PalletProxyAnnouncement>,u128) The announcements made by the proxy (key).

palletVersion(): u16 Returns the current pallet version from storage

proxies(AccountId32): (Vec<PalletProxyProxyDefinition>,u128) The set of account proxies. Maps the account which has delegated to ...

balances.totalIssuance: u128 11,729,681,057,947,515,284

9. ...and another **method**...

## GUIDE TO POLKADOT-JS – PART VIII: Developer

Version 2.0

The screenshot shows the Polkadot.js Developer interface. At the top, there's a navigation bar with 'Kusama' selected, and tabs for 'Accounts', 'Network', 'Governance', and 'Developer'. Below this, the 'Storage' tab is active, showing a query for 'proxies' with the type 'proxies(AccountId32): (Vec<PalletProxyProxyDefinition>,u128)'. A callout box with the text '10. ...then all the necessary options.' points to the 'include option' toggle, which is currently turned off. Below the query, there's a section for 'balances.totalIssuance: u128' with the value '11,729,681,057,947,515,284'.

selected state query ?  
proxy proxies(AccountId32): (Vec<PalletProxyProxyDefinition>,u128) The set of account proxies. Maps the account which has delegated t...

AccountId32  
LEDGER KSM include option ☐

blockhash to query at  
0x...

balances.totalIssuance: u128  
11,729,681,057,947,515,284

10. ...then all the necessary options.

**11. Click on the + button to submit your second query.**

selected state query ?  
proxy proxies(AccountId32): (Vec<PalletProxyProxyDefinition>,u128) The set of account proxies. Maps the account which has delegated t...  
AccountId32  
LEDGER KSM  
include option ☐  
blockhash to query at:  
0x...

**12. The new information requested is now stacked on the interface!**

```
proxy.proxies: (Vec<PalletProxyProxyDefinition>,u128)
[
  [
    {
      delegate: HjcErRijmpoiBiKEHT3edPXM3NFycJogVwDPuByNe7hv9Ae
      proxyType: NonTransfer
      delay: 0
    }
  ]
]
66,803,331,300
```

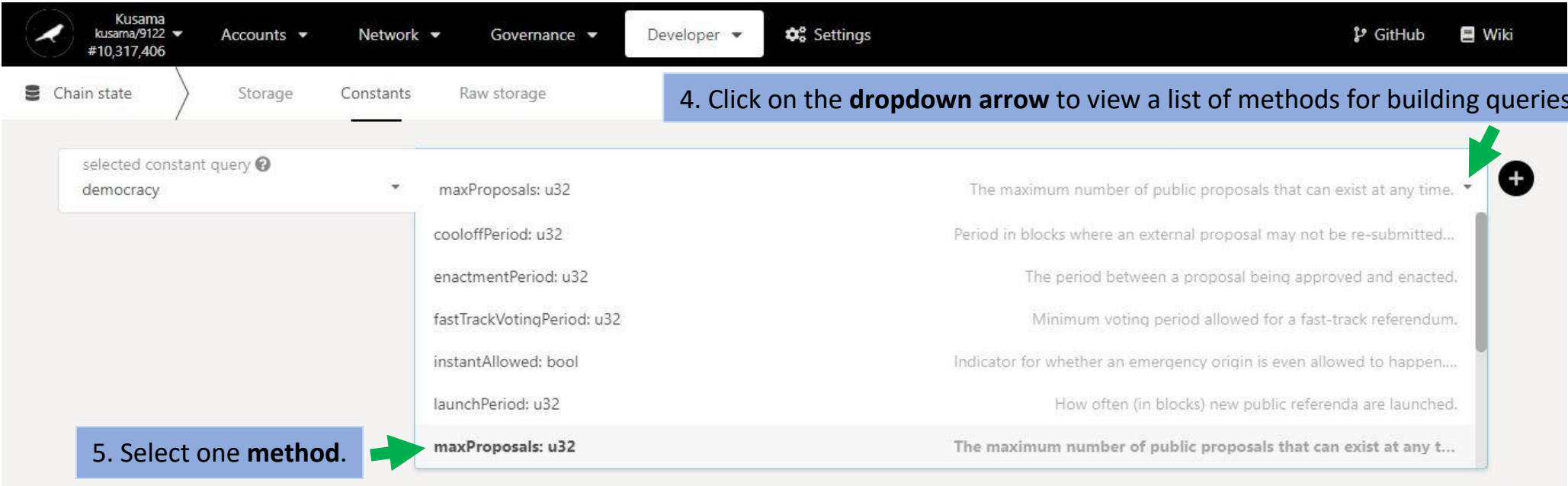
```
balances.totalIssuance: u128
11,729,681,057,937,515,284
```

b) View chain constants.

The image consists of two screenshots of the Polkadot.js Developer interface, illustrating the steps to view chain constants.

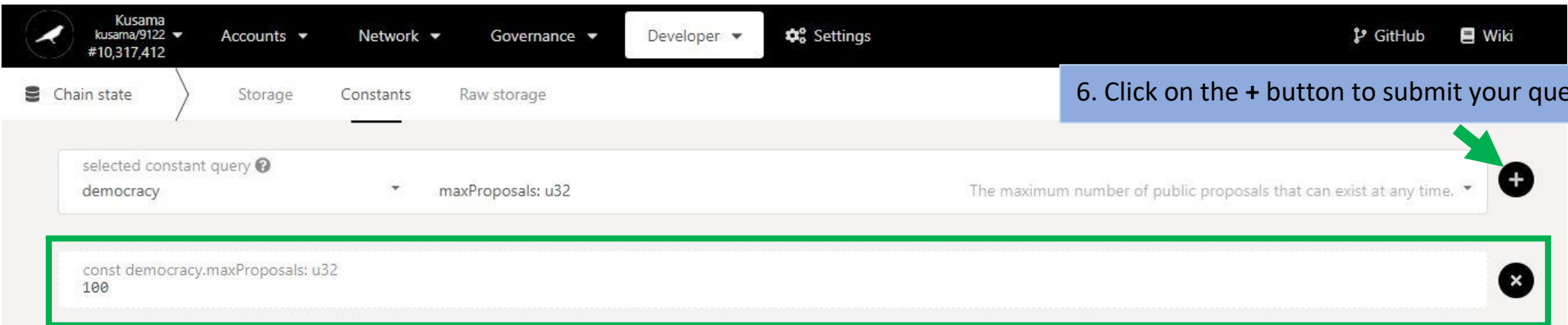
**Top Screenshot:** The interface shows the top navigation bar with the Kusama network selected. The 'Constants' tab is highlighted in the 'Storage' section. A blue callout box with the text '1. Click **Constants**.' has a green arrow pointing to the 'Constants' tab. A yellow callout box with the text 'Interface for accessing information about predefined chain parameters stored on-chain.' is positioned to the right of the 'Constants' tab. Below the tabs, a search bar is shown with 'selected constant query' and a dropdown menu displaying 'system'. The selected constant is 'blockWeights: FrameSystemLimitsBlockWeights'.

**Bottom Screenshot:** The interface shows the 'Constants' tab selected. A blue callout box with the text '2. Click on the **dropdown arrow** to view a list of chain constants to query.' has a green arrow pointing to the dropdown arrow next to the search bar. The dropdown menu is open, showing a list of constants: 'democracy', 'bounties', 'claims', 'crowdloan', 'democracy' (highlighted), 'electionProviderMultiPhase', and 'gilt'. A blue callout box with the text '3. Select one **chain constant**.' has a green arrow pointing to the highlighted 'democracy' option. The search bar now displays 'cooloffPeriod: u32'.



4. Click on the **dropdown arrow** to view a list of methods for building queries.

5. Select one **method**.



6. Click on the **+** button to submit your query.

```
const democracy.maxProposals: u32
100
```

7. The information requested is now available on the interface!



## GUIDE TO POLKADOT-JS – PART VIII: Developer

Version 2.0

Kusama kusama/9122 #10,317,435

Accounts Network Governance Developer Settings

Chain state Storage Constants Raw storage

staking bondingDuration: u32 Number of eras that staked funds must remain bonded for.

slots

society

staking

system

timestamp

tips

8. You can continue to build different queries by selecting another **chain constant**...

Kusama kusama/9122 #10,317,440

Accounts Network Governance Developer Settings

Chain state Storage Constants Raw storage

selected constant query ?

staking

const democracy.maxProposals: u32 100

bondingDuration: u32 Number of eras that staked funds must remain bonded for.

**bondingDuration: u32** Number of eras that staked funds must remain bonded for.

maxNominations: u32 MaxNominations

maxNominatorRewardedPerValidator: u32 The maximum number of nominators rewarded for each validator.

sessionsPerEra: u32 Number of sessions per era.

slashDeferDuration: u32 Number of eras that slashes are deferred by, after computation.

9. ...and another **method**...

The screenshot shows the Polkadot.js Developer interface. The top navigation bar includes the Kusama logo, account information (kusama/9122 #10,317,445), and tabs for Accounts, Network, Governance, and Developer. The Developer tab is active, and the Settings icon is visible. Below the navigation bar, the 'Chain state' tab is selected, and the 'Constants' sub-tab is active. A list of constants is displayed, including 'staging' and 'slashDeferDuration: u32'. A blue callout box with the text '10. Click on the + button to submit your second query.' points to a '+' button in the top right corner of the constants list. Another blue callout box with the text '11. The new information requested is now stacked on the interface!' points to a newly added query in the list: 'const staging.slashDeferDuration: u32 27'. The interface also shows a 'Number of eras that slashes are deferred by, after computation.' dropdown menu.

10. Click on the + button to submit your second query.

11. The new information requested is now stacked on the interface!

c) Retrieve raw storage information.

The image consists of three screenshots of the Polkadot.js Developer interface, illustrating the steps to retrieve raw storage information.

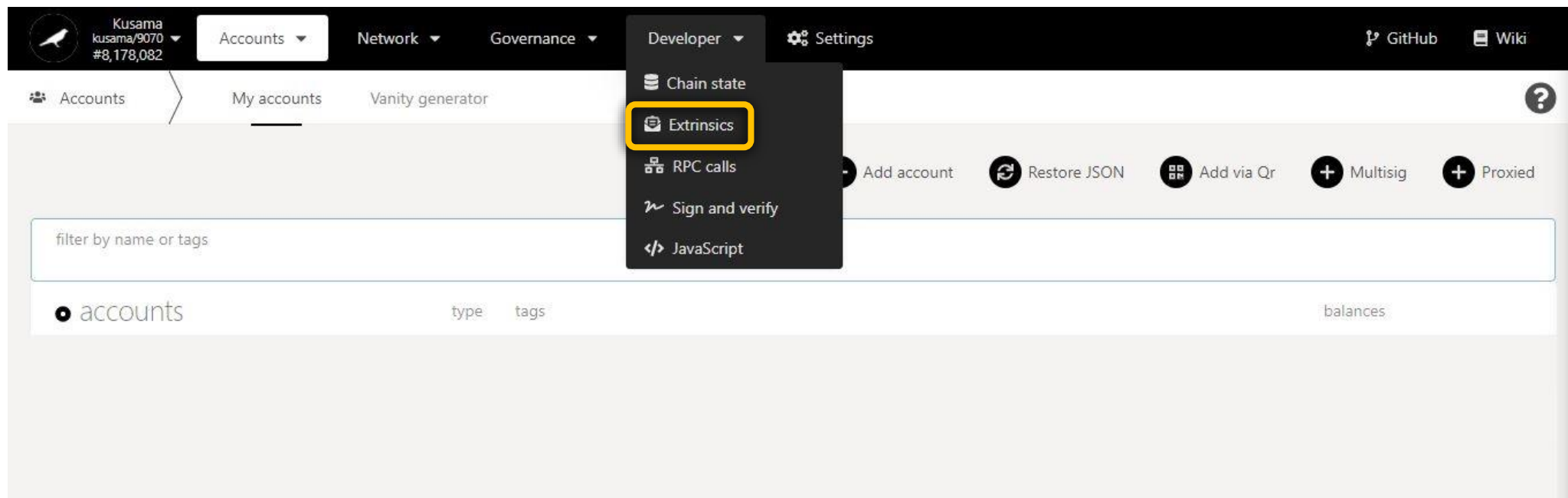
**Step 1:** The first screenshot shows the top navigation bar with the 'Raw storage' tab selected. A blue callout box with the text '1. Click Raw storage.' has a green arrow pointing to the 'Raw storage' tab. A yellow callout box on the right says 'Interface for accessing information about storage maps stored on-chain.'

**Step 2:** The second screenshot shows the 'hex-encoded storage key' input field filled with a long hexadecimal string. A blue callout box with the text '2. Provide the storage key constructed for the map to query and click the + button.' has two green arrows: one pointing to the input field and another pointing to the '+' button on the right.

**Step 3:** The third screenshot shows the same input field, but now the raw storage data is displayed below it. A blue callout box with the text '3. The information requested is now available on the interface, ready to be further decoded!' is positioned over the data. The data is a long hexadecimal string starting with '0x26aa394eea5630e07c48ae0c9558cef7b99d880ec681799c0cf30e8886371da9ace1a6b1b7323d8b9a275f85eaf1fc248cc1b91e8946862c2c79915a4bc004926510fcf71c422fde977c0b0e9d9be40e:' followed by a long sequence of zeros.

You can find more about Storage keys construction and Storage maps on this page:  
<https://www.shawntabrizi.com/substrate/transparent-keys-in-substrate/>

## 2. Extrinsics: Make API calls to submit data onto the chain directly.



a) View runtime modules and methods.

The screenshot displays the Polkadot.js Developer interface. At the top, a dark navigation bar contains the Kusama logo, account information (Kusama, kusama/9122, #10,317,610), and tabs for Governance, Developer (selected), and Settings. On the right of the navigation bar are links for GitHub and Wiki. Below the navigation bar, a secondary bar shows 'Extrinsics', 'Submission' (selected), and 'Decode'. A blue box with the text '1. Click Submission.' and a green arrow points to the 'Submission' tab. A yellow box with the text 'Interface for submitting external information onto the chain.' is positioned below the tabs. The main content area is enclosed in a green border and shows the 'Submission' interface. It includes a header with the account name 'ANAELLE LTD@KSM' and a 'free balance' of '4.0410 KSM'. Below this, a section titled 'submit the following extrinsic' shows a dropdown menu set to 'system' and a method 'setCode(code)'. A text input field for 'code: Bytes' is provided, with a hint '0x prefixed hex, e.g. 0x1234 or ascii data'. To the right of the input field is a 'file upload' toggle switch. Below the input field are two sections for 'encoded call data' and 'encoded call hash', each with a '0x' prefix and a copy icon. At the bottom right of the interface are two buttons: 'Submit Unsigned' and 'Submit Transaction'.

1. Click **Submission**.

Interface for **submitting external information** onto the chain.

using the selected account  
ANAELLE LTD@KSM  
free balance 4.0410 KSM  
HjcErRijmpoiBiKEHT3edPXM3NFycJogVwDPuByNe7hv9Ae

submit the following extrinsic  
system setCode(code) Set the new runtime code.

code: Bytes  
0x prefixed hex, e.g. 0x1234 or ascii data file upload

encoded call data  
0x

encoded call hash  
0x

Submit Unsigned Submit Transaction

## GUIDE TO POLKADOT-JS – PART VIII: Developer

Version 2.0

The screenshot shows the Polkadot.js Developer interface. At the top, there is a navigation bar with links for Accounts, Network, Governance, and Developer (selected). Below this, there are tabs for Extrinsics, Submission, and Decode. The main area displays the account 'ANAELLE LTD@KSM' with a free balance of 4.0410 KSM. A dropdown menu is open, showing a list of runtime modules: system, democracy, electionProviderMultiPhase, guilt, grandpa, hrmp, and identity. A green arrow points to the 'democracy' module, indicating it should be selected. Another green arrow points to the dropdown arrow in the 'system' module, indicating it should be clicked to view all available runtime modules. The interface also includes a 'setCode(code)' button and a 'file upload' toggle.

2. Click on the **dropdown arrow** to view all the available runtime modules.

3. Select one **module**.

[illegible]

## GUIDE TO POLKADOT-JS – PART VIII: Developer

Version 2.0

Kusama  
kusama/9122  
#10,317,772

Accounts

Network

Governance

Developer

Settings

GitHub

Wiki

Extrinsics

Submission

Decode

using the selected account

ANAELLE LTD@KSM

free balance 4,0410 KSM

HjcErRijmpoiBiKEHT3edPX13NFycJogVwDPuByNe7hv9Ae

submit the following extrinsic ?

democracy

target: AccountId32

LEDGER KSM

encoded call data

0x0d13673f7dcedb9c2c96f1678a77d5c5dee88e3

0x165317ff7673a91ddfc2fc8e960a441cd0cd8665

unlock(target)

removeVote(index)

second(proposal, secondsUpperBound)

undelegate()

unlock(target)

vetoExternal(proposalHash)

vote(refIndex, vote)

Unlock tokens that have an expired lock.

Remove a vote for a referendum.

Signals agreement with a particular proposal.

Undelegate the voting power of the sending account.

Unlock tokens that have an expired lock.

Veto and blacklist the external proposal hash.

Vote in a referendum. If `vote.is\_aye()`, the vote is to enact the proposal;


Submit Unsigned

Submit Transaction

6. Select one method.







Kusama  
kusama/9122  
#10,317,776

Accounts ▾

Network ▾

Governance ▾

Developer ▾

Settings

GitHub

Wiki

Extrinsics

Submission

Decode



using the selected account  
✓ ANAELLE LTD@KSM

7. The module and method requested for the call are now available on the interface!

submit the following extrinsic ?

democracy ▾ unlock(target) Unlock tokens that have an expired lock. ▾

target: AccountId32  
LEDGER KSM Euh ▾

encoded call data  
0x0d13673f7dcedb9c2c96f1678a77d5c5dee88e349d3043bb81c2e9f4dc9d371e1728

encoded call hash  
0x165317ff7673a91ddfc2fc8e960a441cd0cd8665e82424480ca3890642126400

Submit Unsigned Submit Transaction


## b) Submit calls.

The screenshot displays the Polkadot.js Developer interface. At the top, a dark navigation bar contains the Kusama logo, account information (Kusama, kusama/9122, #10,317,638), and links for Governance, Developer, Settings, GitHub, and Wiki. Below this, a light gray bar shows tabs for Extrinsics, Submission (selected), and Decode. The main content area shows the 'Submission' tab. On the left, a dropdown menu is open, listing runtime modules: democracy, society, staking (highlighted with a green arrow), system, technicalCommittee, technicalMembership, and timestamp. A blue callout box with the text '3. Select one module.' points to the 'staking' option. Above the dropdown, a blue callout box with the text '2. Click on the dropdown arrow to view all the available runtime modules.' points to the dropdown arrow. At the top of the main content area, another blue callout box with the text '1. Click Submission.' points to the 'Submission' tab. The right side of the interface shows the account balance (free balance 4.0410 KSM) and a list of transactions, including one with the hash 982b5e6be893d7f4510f2963f9. At the bottom right, there are two buttons: 'Submit Unsigned' and 'Submit Transaction'.

1. Click **Submission**.

2. Click on the **dropdown arrow** to view all the available runtime modules.

3. Select one **module**.


 Kusama  
kusama/9122  
#10,317,651


Accounts ▾


Network ▾

Governance ▾

Developer ▾

Settings 


GitHub 

Wiki 

Extrinsics

Submission

Decode

 using the selected account  
✓ ANAELLE LTD@KSM

submit the following extrinsic ?  
staking ▾

controller: MultiAddress (LookupSource)  
Id

value: Compact<u128> (BalanceOf)  
0

payee: PalletStakingRewardDestination  
Staked

**4. Click on the dropdown arrow to view all the available methods for this module.**

**5. Select one method.**

bond(controller, value, payee)  
Take the origin account as a stash and lock up `value` of its balance. `c...

payoutStakers(validatorStash, era)  
Pay out all the stakers behind a single validator for a single era.

reapStash(stash, numSlashingSpans)  
Remove all data structure concerning a staker/stash once its balance is ...

**rebond(value)**  
Rebond a portion of the stash scheduled to be unlocked.

scaleValidatorCount(factor)  
Scale up the ideal number of validators by a factor.

setController(controller)  
(Re-)set the controller of a stash.

setHistoryDepth(newHistoryDepth, erasItemsDeleted)  
Set `HistoryDepth` value. This function will delete any history information

encoded call data  
0x060000673f7dcedb9c2c96f1678a77d5c5dee88e349d3043bb81c2e9f4dc9d371e17280000

encoded call hash  
0x78e8d1f63da23426056266c32fcb7788f0691c36bdfa59dd96d74e1d1239b4db

## GUIDE TO POLKADOT-JS – PART VIII: Developer

Version 2.0

The screenshot shows the Polkadot.js Developer interface. At the top, there's a navigation bar with 'Kusama kusama/9122 #10,317,658', 'Accounts', 'Network', 'Governance', 'Developer', and 'Settings'. Below this, there are tabs for 'Extrinsics', 'Submission', and 'Decode'. The 'Submission' tab is active, showing a form for submitting a transaction. The form includes a section for 'submit the following staking' with a blue callout box stating '6. Enter the information required.' The 'value' field is set to 'Compact<u128> (BalanceOf)' and '2'. Below this, the 'encoded call data' is '0x061308' and the 'encoded call hash' is '0xd156c384d0451ac7ecc448c9252e8614be2f24a841a535203dcdd76d3063f1ed'. At the bottom right, there are two buttons: 'Submit Unsigned' and 'Submit Transaction', with the latter being circled in green.

using the selected account

ANAELLE LTD@KSM

free balance 4,0410 KSM

HjcErRijmpoiBiKEHT3edPXm3NFyc3ogVwDPuByNe7hv9Ae

submit the following staking

6. Enter the information required.

value: Compact<u128> (BalanceOf)

2

encoded call data

0x061308

encoded call hash

0xd156c384d0451ac7ecc448c9252e8614be2f24a841a535203dcdd76d3063f1ed

Submit Unsigned

Submit Transaction

7. Click on **Submit Transaction** to continue the procedure.

c) Decode hex-encoded calls.

The screenshot shows the Polkadot.js Developer interface. At the top, a dark navigation bar contains the Kusama logo, account information (kusama/9122, #10,317,732), a dropdown menu, 'Accounts', 'Developer', 'Settings', 'GitHub', and 'Wiki'. Below this is a light-colored bar with tabs: 'Extrinsics', 'Submission', and 'Decode'. The 'Decode' tab is selected and highlighted with a green underline. A blue box with the text '1. Click Decode.' and a green arrow points to the 'Decode' tab. To the right of the tabs, a yellow box contains the text 'Interface for decoding external information (to be) submitted onto the chain.' The main content area is enclosed in a green border and contains a large text input field labeled 'hex-encoded call' with placeholder text '0x...'. Below this is a section titled 'using the selected account' which shows a green checkmark, the account name 'ANAELLE LTD@KSM', the free balance '4.0410 KSM', and the account address 'HjcErRijmpoiBiKEHT3edPXH3NFycJogVwDPuByNe7hv9Ae'. At the bottom right of the main area are two buttons: 'Submit Unsigned' and 'Submit Transaction'.

1. Click Decode.

Interface for decoding external information (to be) submitted onto the chain.

hex-encoded call  
0x...


using the selected account

ANAELLE LTD@KSM

free balance 4.0410 KSM

HjcErRijmpoiBiKEHT3edPXH3NFycJogVwDPuByNe7hv9Ae

Submit Unsigned Submit Transaction



Kusama  
kusama/9122  
#10,317,776

2. Click **Submission.**

Governance ▾

Developer ▾

Settings ⚙


GitHub

Wiki

Extrinsics

Submission

Decode

 using the selected account  
ANAELLE LTD@KSM

3. Select the **module and method** required for your call.


free balance 4,0410 KSM  
HjcErRijmpoiBiKEHT3edPXm3NFycJogVwDPuByNe7hv9Ae ▾

submit the following extrinsic ?

democracy ▾

unlock(target)

Unlock tokens that have an expired lock. ▾

 target: AccountId32  
LEDGER KSM

4. Copy the call data and store it for future use.

encoded call data  
0x0d13673f7dcedb9c2c96f1678a77d5c5dee88e349d3043bb81c2e9f4dc9d371e1728

encoded call hash  
0x165317ff7673a91ddfc2fc8e960a441cd0cd8665e82424480ca3890642126400

Submit Unsigned

Submit Transaction

5. Click **Decode**.

6. Paste the call data previously stored.

7. The module and method of the call are now loaded on the interface!

hex-encoded call  
0x0d13673f7dcedb9c2c96f1678a77d5c5dee88e349d3043bb81c2e9f4dc9d371e1728

decoded call  
democracy unlock(target) Unlock tokens that have an expired lock.

target: AccountId32  
LEDGER KSM

encoded call hash  
0x165317ff7673a91ddfc2fc8e960a441cd0cd8665e82424480ca3890642126400

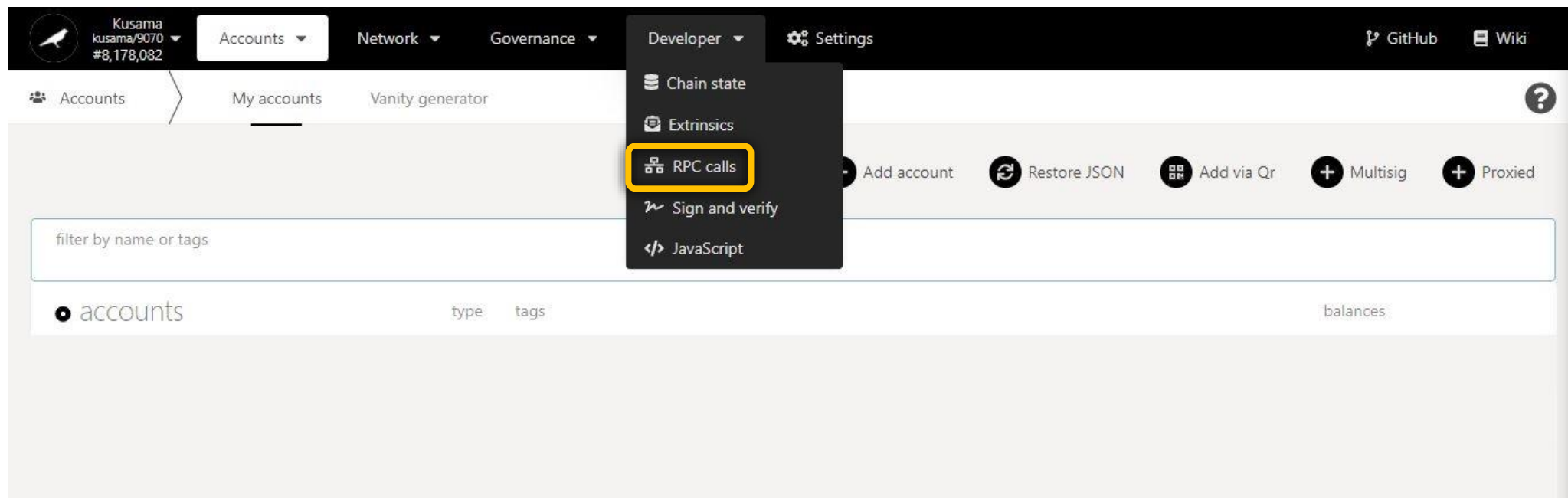
using the selected account  
ANAELLE LTD@KSM

free balance 4.0410 KSM  
HjcErRijmpoibIKEHT3edPXm3NFycJogVwDPuByNe7hv9Ae

Submit Unsigned Submit Transaction

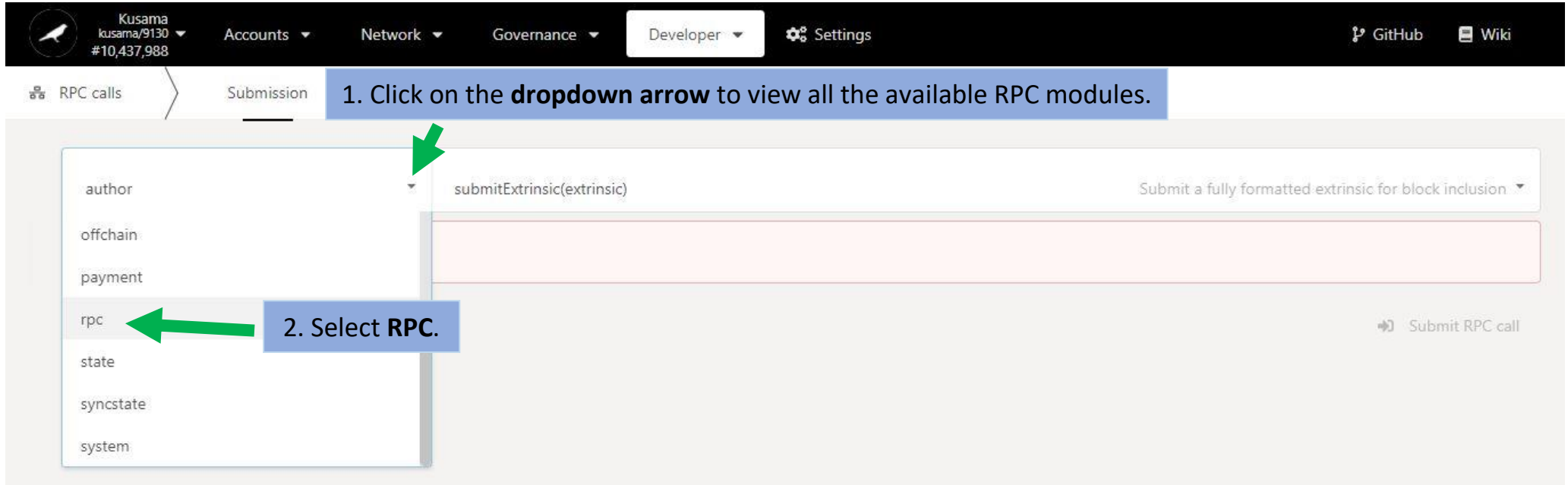
8. Click on **Submit Transaction** to continue the procedure.

### 3. RPC calls: Make RPC calls to submit data onto the chain remotely.





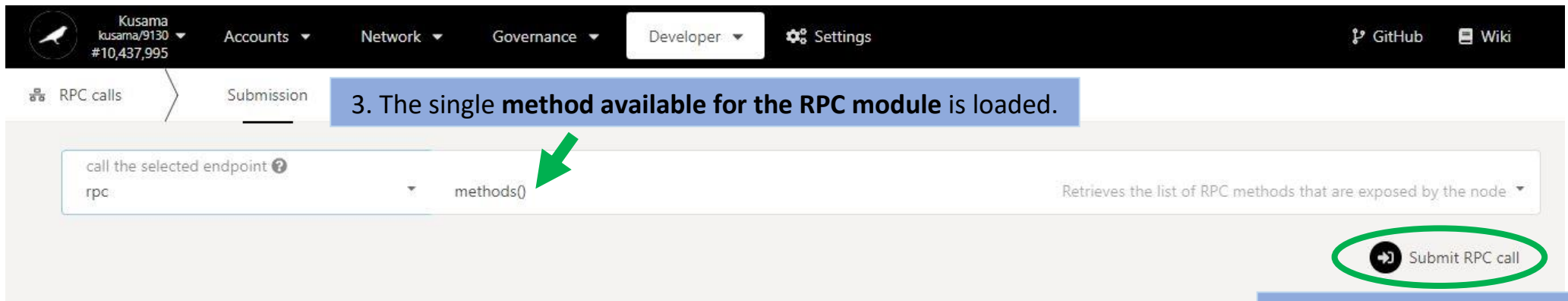
a) View JSON RPC modules and methods.



The screenshot shows the top navigation bar of the Polkadot.js Developer interface. The 'Developer' dropdown menu is open, and the 'rpc' option is selected. A green arrow points to the 'rpc' option in the dropdown menu. Another green arrow points to the 'submitExtrinsic(extrinsic)' button in the main content area.

1. Click on the **dropdown arrow** to view all the available RPC modules.


2. Select **RPC**.



The screenshot shows the same interface as the previous one, but now the 'rpc' module is selected. The 'methods()' button is highlighted with a green arrow. The 'Submit RPC call' button is circled in green.

3. The single **method** available for the RPC module is loaded.

4. Click on **Submit RPC call** to continue the procedure.




Kusama  
kusama/9130  
#10,437,999


Accounts ▾


Network ▾

Governance ▾

Developer ▾


Settings 

GitHub 

Wiki 

RPC calls

Submission


call the selected endpoint 

rpc ▾

methods() ▾

Retrieves the list of RPC methods that are exposed by the node ▾

**5. All RPC modules and methods available are now listed on the interface, ready to be used remotely!**

Submit RPC call 

```
1: rpc.methods
{
  version: 1
  methods: [
    author_hasKey
    author_hasSessionKeys
    author_insertKey
    author_pendingExtrinsics
    author_removeExtrinsic
    author_rotateKeys
    author_submitAndWatchExtrinsic
    author_submitExtrinsic
    author_unsubscribesubmitAndWatchExtrinsic
    author_unwatchExtrinsic
    babe_epochAuthorship
    beefy_subscribeJustifications
    beefy_unsubscribeJustifications
    chain_getBlock
    chain_getBlockHash
    chain_getFinalisedHead
```

## b) Submit RPC calls.

Kusama kusama/9130 #10,437,905 Accounts Network Governance Developer Settings GitHub Wiki

RPC calls Submission

Interface for submitting external information onto the chain remotely.

call the selected endpoint ?  
author submitExtrinsic(extrinsic) Submit a fully formatted extrinsic for block inclusion

extrinsic: Extrinsic  
0x

Submit RPC call

Kusama kusama/9130 #10,437,943 Accounts Network Governance Developer Settings GitHub Wiki

RPC calls Submission

1. Click on the **dropdown arrow** to view all the available RPC modules.

author submitExtrinsic(extrinsic) Submit a fully formatted extrinsic for block inclusion

author  
babe  
beefy  
chain  
childstate  
grandpa

2. Select one **module**.

Submit RPC call

## GUIDE TO POLKADOT-JS – PART VIII: Developer

Version 2.0

Kusama kusama/9130 #10,437,950 Accounts Network Governance Developer Settings GitHub Wiki

RPC calls Submission

3. Click on the **dropdown arrow** to view all the available methods for this module.

call the selected endpoint ?  
chain

hash: Option<BlockHash>  
<empty>

4. Select one **method**.

getBlock(hash)	Get header and body of a relay chain block
<b>getBlock(hash)</b>	<b>Get header and body of a relay chain block</b>
getBlockHash(blockNumber)	Get the block hash for a specific block
getFinalizedHead()	Get hash of the last finalized block in the canon chain
getHeader(hash)	Retrieves the header for a specific block

Kusama kusama/9130 #10,437,955 Accounts Network Governance Developer Settings GitHub Wiki

RPC calls Submission


call the selected endpoint ?  
chain

getFinalizedHead()

Get hash of the last finalized block in the canon chain

Submit RPC call

5. Click on **Submit RPC call** to continue the procedure.

 Kusama  
kusama/9130  
#10,437,958

Accounts ▾

Network ▾

Governance ▾

Developer ▾

⚙️ Settings

chain.getFinalizedHead sent

RPC calls

Submission

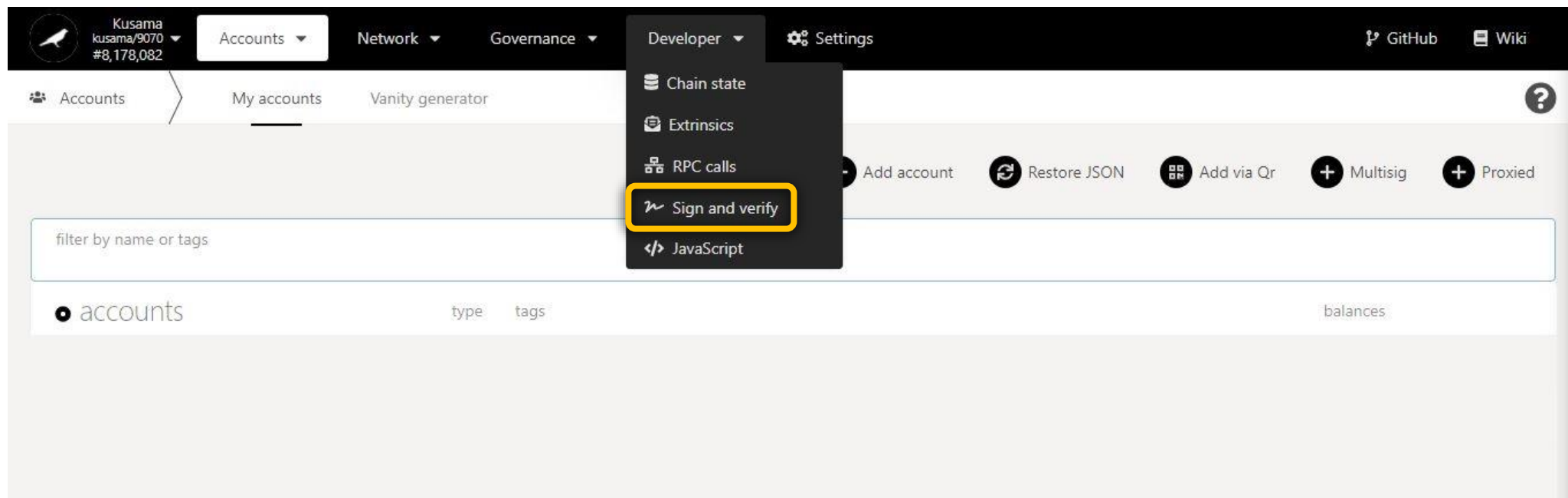
call the selected endpoint ?  
chain ▾ getFinalizedHead() Get hash of the last finalized block in the canon chain ▾

5. The information submitted has now returned a response on the interface!

Submit RPC call

1: chain.getFinalizedHead  
0x30b97bc9d2540202aed1ca08d25405bc687c45f922933ceddc7d7d951808f524

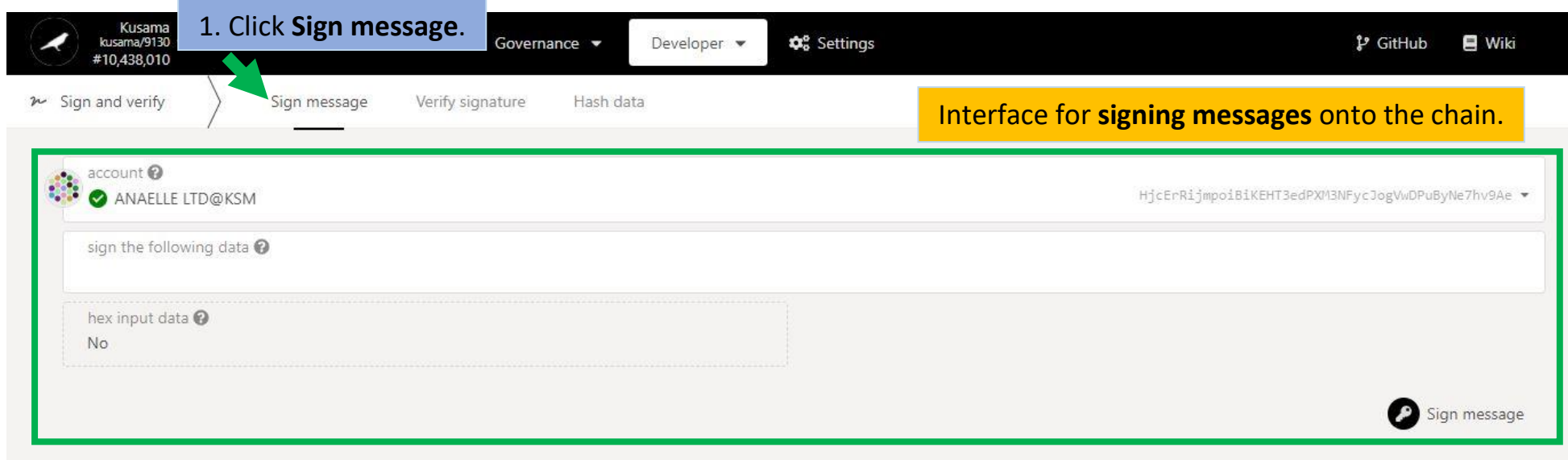
#### 4. Sign & Verify: Sign messages onto the chain and verify on-chain signatures.



## a) Sign messages.

1. Click Sign message.

Interface for signing messages onto the chain.



2. Select the **account** that will sign the message.

3. Enter the **message** to be signed.

4. Click on **Sign message** to continue the procedure.



## GUIDE TO POLKADOT-JS – PART VIII: Developer

Version 2.0

**Summary of the transaction sent via the Polkadot-JS extension.**

**5. Enter your account's password and tick the box to remember your password, if necessary.**

**6. Click on **Sign the message** to complete the procedure.**



The screenshot shows the Polkadot.js Developer interface. At the top, there's a navigation bar with 'Kusama kusama/9130 #10,438,030', 'Accounts', 'Network', 'Governance', 'Developer', and 'Settings'. Below this, there are tabs for 'Sign and verify', 'Sign message', 'Verify signature', and 'Hash data'. The 'Sign message' tab is active. In the 'Sign message' section, the account 'ANAELLE LTD@KSM' is selected. The data to be signed is 'Signing off from Kusama!'. The 'hex input data' is set to 'No'. The 'signature of supplied data' is displayed as a long hexadecimal string: '0xe0c76f3bb1aac9d0a846d700f4aa0fa169e1a850ae4e1c808b6ab3aca1c61d3bd02074d8b0c46dbae74bcfe49b5ddc3b3246e42c4aba23776828f47a49456f88'. A green box highlights this signature. A green arrow points to a copy icon in the top right corner of the signature box. Two blue callout boxes provide instructions: '7. A signature of the signed messaged is now available on the interface!' and '8. Copy the signature and store it for future use.'

Kusama  
kusama/9130  
#10,438,030

Accounts Network Governance Developer Settings

GitHub Wiki

Sign and verify Sign message Verify signature Hash data

account ?  
ANAELLE LTD@KSM

sign the following data ?  
Signing off from Kusama!

hex input data ?  
No

7. A signature of the signed messaged is now available on the interface!

signature of supplied data ?  
0xe0c76f3bb1aac9d0a846d700f4aa0fa169e1a850ae4e1c808b6ab3aca1c61d3bd02074d8b0c46dbae74bcfe49b5ddc3b3246e42c4aba23776828f47a49456f88

8. Copy the signature and store it for future use.

## b) Verify signatures.

1. Click **Verify signature.**

Interface for **verifying signatures** stored on-chain.

verify using address ?  
✓ ANAELLE LTD@KSM HjcErRijmpoiBiKEHT3edPXM3NFycJogVwDPuByNe7hv9Ae

using the following data ?

the supplied signature ?

signature crypto type ?  
Crypto not detected

hex input data ?  
No

2. Select the **account** that signed the message.

3. Enter the **message** that was signed.

4. Paste the **signature** previously stored.

5. If any of the information provided is incorrect, the signature will be flagged as invalid!

verify using address ?  
✓ ANAELLE LTD@KSM HjcErRijmpoiBiKEHT3edPXM3NFycJogVwDPuByNe7hv9Ae

using the following data ?  
Signing off from KSM!


the supplied signature ?  
0xe0c76f3bb1aac9d0a846d700f4aa0fa169e1a850ae4e1c808b6ab3aca1c61d3bd02074d8b0c46dbae74bcfe49b5ddc3b3246e42c4aba23776828f47a49456f88

signature crypto type ?  
Crypto not detected

hex input data ?  
No

## GUIDE TO POLKADOT-JS – PART VIII: Developer

Version 2.0



Kusama  
kusama/9130  
#10,438,060

Accounts ▾

Network ▾

Governance ▾

Developer ▾

Settings

GitHub


Wiki

Sign and verify

Sign message

Verify signature

Hash data

 verify using address ?  
✓ ANAELLE LTD@KSM

HjcErRijmpoiBiKEHT3edPXI3NFycJogVwDPuByNe7hv9Ae ▾

using the following data ?  
Signing off from Kusama!

6. Check and correct the message that was signed.

✓ the supplied signature ?  
0xe0c76f3bb1aac9d0a846d700f4aa0fa169e1a850ae4e1c808b6ab3aca1c61d3bd02074d8b0c46dbae74bcfe49b5ddc3b3246e42c4aba23776828f47a49456f88

signature crypto type ?  
Schnorrkel (sr25519, recommended)

hex input data ?  
No

7. Once all the information provided is correct, the signature will be marked as valid!

c) Hash data.

Kusama kusama/9130 #10,438,066 Accounts Developer Settings GitHub Wiki

Sign and verify Sign message Verify signature **Hash data**

Interface for encrypting messages to be signed onto the chain.

from the following data ?

hex input data ?  
No

the resulting hash is ?  
0x0e5751c026e543b2e8ab2eb06099daa1d1e5df47778f7787faab45cdf12fe3a8

Kusama kusama/9130 #10,438,072 Accounts Network Governance Developer Settings GitHub Wiki

Sign and verify Sign message Verify signature **Hash data**

from the following data ?  
Signing off from Kusama!

hex input data ?  
No

the resulting hash is ?  
0x18b9d6f908d8112ec045eb9a6231bc0a47dd853b07874e5404447e8ef3b13783

4. Copy the **hash** and store it for future use.

5. Click **Sign message**.

6. Select the **account** that will sign the message.

7. Paste the **hash previously stored**.

8. Click on **Sign message** to continue the procedure.

9. A **signature of the hash (encrypted message)** will be available on the interface!

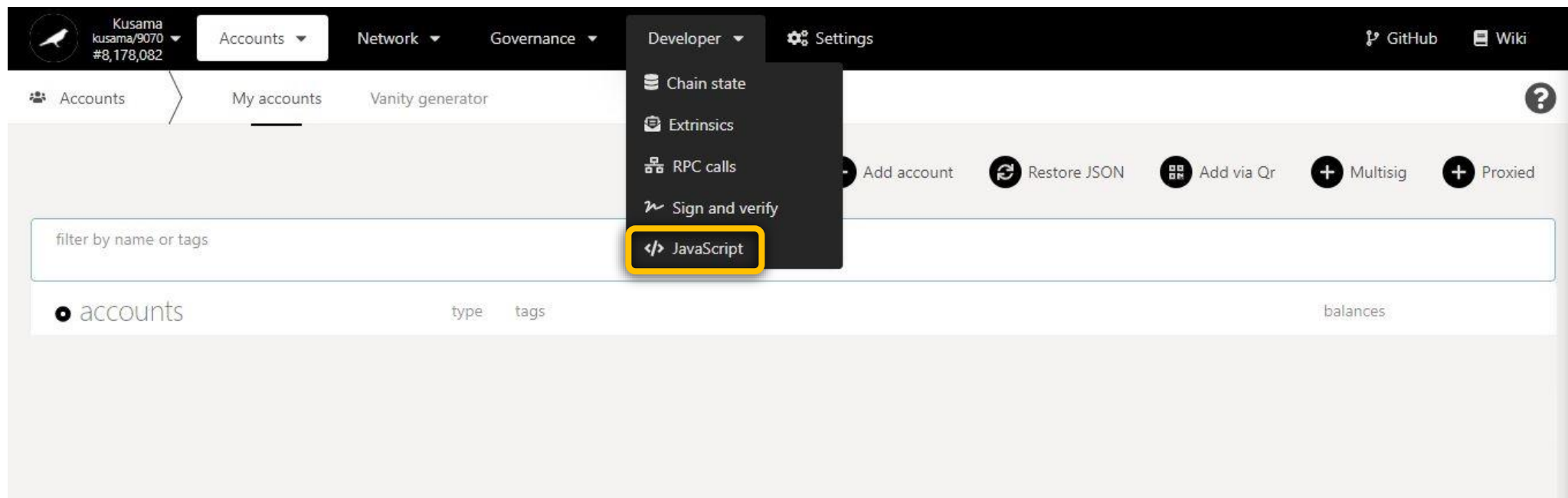
10. Click **Verify signature**.

11. Select the **account** that signed the encrypted message.

12. Enter the **hash (encrypted message)** that was signed.

13. Paste the **signature of the hash (encrypted message)** for verification.

## 5. JavaScript: Interact with on-chain data through the console.



## a) View and submit code snippets.

The screenshot displays the Polkadot.js Developer interface. At the top, a dark navigation bar includes the Kusama logo, account information (kusama/9130, #10,438,101), and dropdown menus for Accounts, Network, Governance, and Developer. A Settings gear icon and links to GitHub and Wiki are also present. Below the navigation bar, a tabbed interface shows 'JavaScript' and 'Console'. A yellow callout box on the right states: 'Interface for executing JS scripts on-chain in isolation.' The main area is a code editor with a light theme, containing a JavaScript snippet. The code is wrapped in an async closure and subscribes to new headers, logging their information. A dark console area on the right is currently empty. The entire interface is framed by a green border.

Kusama  
kusama/9130  
#10,438,101

Accounts Network Governance Developer Settings

GitHub Wiki

JavaScript Console

Interface for executing JS scripts on-chain in isolation.

Select example  
Listen to new Head

```
1 // All code is wrapped within an async closure,  
2 // allowing access to api, hashing, types, util.  
3 // (async ({ api, hashing, types, util }) => {  
4 //   ... any user code is executed here ...  
5 // })();  
6  
7 // subscribe to new headers, printing the full info for 5 Blocks  
8 let count = 0;  
9 const unsub = await api.rpc.chain.subscribeNewHeads((header) => {  
10   console.log(`#${header.number}:`, header);  
11  
12   if (++count === 5) {  
13     console.log('5 headers retrieved, unsubscribing');  
14     unsub();  
15   }  
16 });
```

## GUIDE TO POLKADOT-JS – PART VIII: Developer

Version 2.0

The screenshot shows the Polkadot.js Developer interface. At the top, there is a navigation bar with the Kusama logo, account information (kusama/9130, #10,438,169), and tabs for Accounts, Network, Governance, Developer (selected), and Settings. On the right of the navigation bar are links for GitHub and Wiki. Below the navigation bar, there are tabs for JavaScript and Console. A blue callout box with the text "1. Click on the dropdown arrow and select a ready-made code snippet to execute." points to a dropdown arrow in the top right corner of the code editor area. The code editor is divided into two panes. The left pane, titled "Select example" and "Get staking parameters", contains a code snippet. The right pane is currently empty. The code snippet in the left pane is as follows:

```
1 // All code is wrapped within an async closure,
2 // allowing access to api, hashing, types, util.
3 // (async ({ api, hashing, types, util }) => {
4 //   ... any user code is executed here ...
5 // })();
6
7 // Get SRML staking parameters as consts
8 // 'parameter_types' were added to substrate with spec_version: 101.
9 // This example will throw an error if used with versions before that.
10
11 const bondingDuration = api.consts.staking.bondingDuration;
12 const sessionsPerEra = api.consts.staking.sessionsPerEra;
13
14 console.log('Staking bonding duration: ' + bondingDuration);
15 console.log('Staking sessions per era: ' + sessionsPerEra);
```



## GUIDE TO POLKADOT-JS – PART VIII: Developer

Version 2.0

The screenshot shows the Polkadot.js Developer interface. At the top, there's a navigation bar with the Kusama logo, account information (kusama/9130, #10,438,179), and tabs for Accounts, Network, Governance, Developer (selected), and Settings. Below the navigation bar, there's a tab for JavaScript and a Console tab. The main area displays a code editor with a JavaScript example for getting staking parameters. The code is wrapped in an async closure and uses the api.consts.staking object to retrieve bondingDuration and sessionsPerEra. The console output is currently empty. A blue callout box highlights the instruction to change the labels in green for better legibility of the data.

Select example  
Get staking parameters

```
1 // All code is wrapped within an async closure,  
2 // allowing access to api, hashing, types, util.  
3 // (async ({ api, hashing, types, util }) => {  
4 //   ... any user code is executed here ...  
5 // })(  
6  
7 // Get SRML staking parameters as consts  
8 // 'parameter_types' were added to substrate with spec_version: 101.  
9 // This example will throw an error if used with versions before that.  
10  
11 const bondingDuration = api.consts.staking.bondingDuration;  
12 const sessionsPerEra = api.consts.staking.sessionsPerEra;  
13  
14 console.log('Staking bonding duration: ' + bondingDuration);  
15 console.log('Number of sessions per era: ' + sessionsPerEra);
```

2. Change the **labels** (in green) for better legibility of the data.

## GUIDE TO POLKADOT-JS – PART VIII: Developer

Version 2.0

The screenshot shows the Polkadot.js Developer interface. At the top, there's a navigation bar with a profile icon (Kusama, kusama/9130, #10,438,187), dropdown menus for Accounts, Network, Governance, and Developer, a Settings gear icon, and links to GitHub and Wiki. Below the navigation bar, there's a tabbed interface with 'JavaScript' and 'Console'. The 'JavaScript' tab is active, showing a code editor with a script. A blue callout box with the text '3. Click on the play button to run the script.' points to a play button icon in the top right corner of the code editor. The script in the editor is as follows:

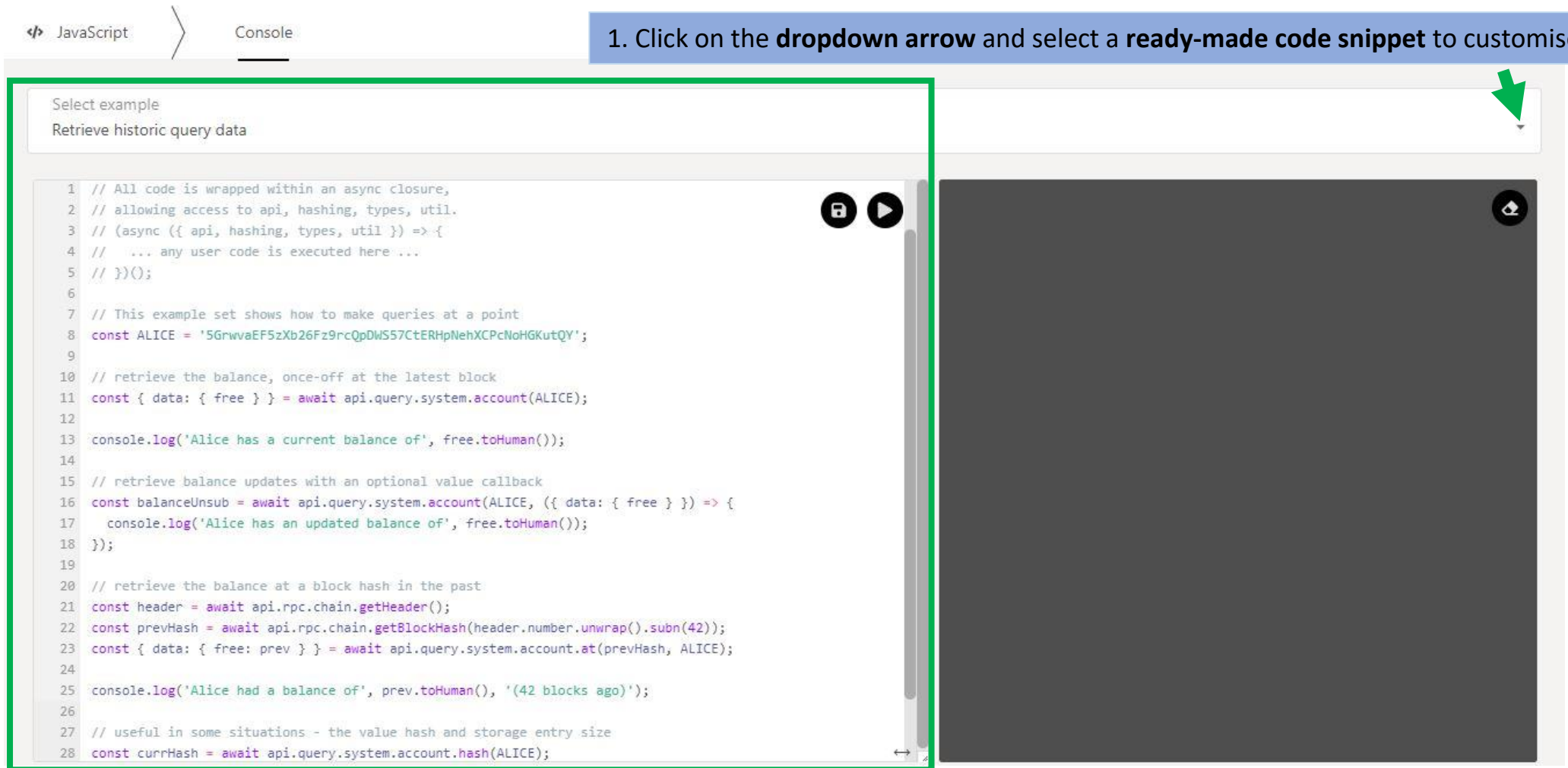
```
1 // All code is wrapped within an async closure,  
2 // allowing access to api, hashing, types, util.  
3 // (async ({ api, hashing, types, util }) => {  
4 //   ... any user code is executed here ...  
5 // })(api, hashing, types, util);  
6  
7 // Get SRML staking parameters as consts  
8 // 'parameter_types' were added to substrate with spec_version: 101.  
9 // This example will throw an error if used with versions before that.  
10  
11 const bondingDuration = api.consts.staking.bondingDuration;  
12 const sessionsPerEra = api.consts.staking.sessionsPerEra;  
13  
14 console.log('Staking bonding duration: ' + bondingDuration);  
15 console.log('Number of sessions per era: ' + sessionsPerEra);
```

Below the code editor, there's a console output area. A blue callout box with the text '4. Results are now available in the JS console!' points to the console output area. The console output shows the following results:

```
Staking bonding duration: 28  
Number of sessions per era: 6
```

## b) Customise code snippets.

1. Click on the **dropdown arrow** and select a **ready-made code snippet** to customise.



The screenshot shows the Polkadot.js Developer interface. The top bar has tabs for 'JavaScript' and 'Console'. The 'Console' tab is active, displaying a code snippet titled 'Retrieve historic query data'. The code is as follows:

```
1 // All code is wrapped within an async closure,
2 // allowing access to api, hashing, types, util.
3 // (async ({ api, hashing, types, util }) => {
4 //   ... any user code is executed here ...
5 // })();
6
7 // This example set shows how to make queries at a point
8 const ALICE = '5GrwvaEF5zXb26Fz9rcQpDWS57CtERHpNehXCPcNoHGKutQY';
9
10 // retrieve the balance, once-off at the latest block
11 const { data: { free } } = await api.query.system.account(ALICE);
12
13 console.log('Alice has a current balance of', free.toHuman());
14
15 // retrieve balance updates with an optional value callback
16 const balanceUnsub = await api.query.system.account(ALICE, ({ data: { free } }) => {
17   console.log('Alice has an updated balance of', free.toHuman());
18 });
19
20 // retrieve the balance at a block hash in the past
21 const header = await api.rpc.chain.getHeader();
22 const prevHash = await api.rpc.chain.getBlockHash(header.number.unwrap().subn(42));
23 const { data: { free: prev } } = await api.query.system.account.at(prevHash, ALICE);
24
25 console.log('Alice had a balance of', prev.toHuman(), '(42 blocks ago)');
26
27 // useful in some situations - the value hash and storage entry size
28 const currHash = await api.query.system.account.hash(ALICE);
```

A green box highlights the code editor area. A green arrow points to the dropdown arrow in the top right corner of the console, indicating where to click to select a ready-made code snippet.

JavaScript

Console

Select example  
Retrieve historic query data

```
1 // All code is wrapped within an async closure,
2 // allowing access to api, hashing, types, util,
3 // (async ({ api, hashing, types, util }) => {
4 //   ... any user code is executed here ...
5 // })();
6
7 // This example set shows how to make queries at a point
8 const ALICE = '5GrwvaEF5zXb26Fz9rcQpDWS57CtERHpNehXCPcNoHGKutQY';
9
10 // retrieve the balance, once-off at the latest block
11 const { data: { free } } = await api.query.system.account(ALICE);
12
13 console.log('Alice has a current balance of', free.toHuman());
14
15 // retrieve balance updates with an optional value callback
16 const balanceUnsub = await api.query.system.account(ALICE, ({ data: { free } }) => {
17   console.log('Alice has an updated balance of', free.toHuman());
18 });
19
20 // retrieve the balance at a block hash in the past
21 const header = await api.rpc.chain.getHeader();
22 const prevHash = await api.rpc.chain.getBlockHash(header.number.unwrap().subn(42));
23 const { data: { free: prev } } = await api.query.system.account.at(prevHash, ALICE);
24
25 console.log('Alice had a balance of', prev.toHuman(), '(42 blocks ago)');
26
27 // useful in some situations - the value hash and storage entry size
28 const currHash = await api.query.system.account.hash(ALICE);
```

2. Click on the **play button** to run the script and show the results in the JS console.

Alice has a current balance of 0  
Alice has an updated balance of 0  
Alice had a balance of 0 (42 blocks ago)

3. Click on the **eraser button** to clear the console.

JavaScript

Console

4. Change the **variables** (in blue) and the **labels** (in green) to customise the script, then run the script to show the results.

Select example

Retrieve historic query data

```
1 // All code is wrapped within an async closure,
2 // allowing access to api, hashing, types, util.
3 // (async ({ api, hashing, types, util }) => {
4 //   ... any user code is executed here ...
5 // })();
6
7 // This example set shows how to make queries at a point
8 const ANAELELTD = 'HjcErRijmpo1BiKEHT3edPX43NFycJogVwDPuByNe7hv9Ae';
9
10 // retrieve the balance, once-off at the latest block
11 const { data: { free } } = await api.query.system.account(ANAELELTD);
12
13 console.log('AnaelleLTD has a current balance of', free.toHuman());
14
15 // retrieve balance updates with an optional value callback
16 const balanceUnsub = await api.query.system.account(ANAELELTD, ({ data: { free } }) => {
17   console.log('AnaelleLTD has an updated balance of', free.toHuman());
18 });
19
20 // retrieve the balance at a block hash in the past
21 const header = await api.rpc.chain.getHeader();
22 const prevHash = await api.rpc.chain.getBlockHash(header.number.unwrap().subn(600));
23 const { data: { free: prev } } = await api.query.system.account.at(prevHash, ANAELELTD);
24
25 console.log('AnaelleLTD had a balance of', prev.toHuman(), '(600 blocks ago aka 1 hour ago)');
26
```



```
AnaelleLTD has a current balance of 5,038,674,509,552
AnaelleLTD has an updated balance of 5,038,674,509,552
AnaelleLTD had a balance of 5,038,674,509,552 (600 blocks ago aka 1 hour ago)
```

5. Submit with on-chain transactions that will further **test your custom script.**

teleport assets

send from account  
ANAELLE LTD@KSM

transferrable 0.0386 KSM

HjcErRijmpoiBiKEHT3edPXM3NFycJogVwDPuByN...

The transferred balance will be subtracted (along with fees) from the sender account.

destination chain  
Statemine

The destination chain for this asset teleport. The transferred value will appear on this chain.

send to address  
ANAELLE LTD@KSM

HjcErRijmpoiBiKEHT3edPXM3NFycJogVwDPuByN...

The beneficiary will have access to the transferred amount when the transaction is included in a block.

amount

0.0005

KSM

If the recipient account is new, the balance needs to be more than the existential deposit on the recipient chain.

destination transfer fee

10.6666

micro

The amount deposited to the recipient will be net the calculated cross-chain fee.

destination existential deposit

3.3333

micro

Teleport



JavaScript

Console

Select example

Retrieve historic query data

```
1 // All code is wrapped within an async closure,
2 // allowing access to api, hashing, types, util.
3 // (async ({ api, hashing, types, util }) => {
4 //   ... any user code is executed here ...
5 // })();
6
7 // This example set shows how to make queries at a point
8 const ANAELELTD = 'HjcErRijmpoiBiKEHT3edPXM3NFycJogVwDPuByNe7hv9Ae';
9
10 // retrieve the balance, once-off at the latest block
11 const { data: { free } } = await api.query.system.account(ANAELELTD);
12
13 console.log('AnaelleLTD has a current balance of', free.toHuman());
14
15 // retrieve balance updates with an optional value callback
16 const balanceUnsub = await api.query.system.account(ANAELELTD, ({ data: { free } }) => {
17   console.log('AnaelleLTD has an updated balance of', free.toHuman());
18 });
19
20 // retrieve the balance at a block hash in the past
21 const header = await api.rpc.chain.getHeader();
22 const prevHash = await api.rpc.chain.getBlockHash(header.number.unwrap().subn(600));
23 const { data: { free: prev } } = await api.query.system.account.at(prevHash, ANAELELTD);
24
25 console.log('AnaelleLTD had a balance of', prev.toHuman(), '(600 blocks ago aka 1 hour ago)');
26
```

## 6. Check the updated results.

AnaelleLTD has a current balance of 5,038,674,509,552  
AnaelleLTD has an updated balance of 5,038,674,509,552  
AnaelleLTD had a balance of 5,038,674,509,552 (600 blocks ago aka 1 hour ago)  
AnaelleLTD has an updated balance of 5,038,117,176,704

JavaScript

Console

Select example

Retrieve historic query data

7. Change the **functions** (in purple) to further customise your script, then run the script to show the results.

```
1 // All code is wrapped within an async closure,
2 // allowing access to api, hashing, types, util.
3 // (async ({ api, hashing, types, util }) => {
4 //   ... any user code is executed here ...
5 // })();
6
7 // This example set shows how to make queries at a point
8 const ANAELELTD = 'HjcErRjmpoiBiKEHT3edPX43NFycJogVwDPuByNe7hv9Ae';
9
10 // retrieve the balance, once-off at the latest block
11 const { data: { free } } = await api.query.system.account(ANAELELTD);
12
13 console.log('AnaelleLTD has a current balance of', free.toBigInt());
14
15 // retrieve balance updates with an optional value callback
16 const balanceUnsub = await api.query.system.account(ANAELELTD, ({ data: { free } }) => {
17   console.log('AnaelleLTD has an updated balance of', free.toHuman());
18 });
19
20 // retrieve the balance at a block hash in the past
21 const header = await api.rpc.chain.getHeader();
22 const prevHash = await api.rpc.chain.getBlockHash(header.number.unwrap().subn(600));
23 const { data: { free: prev } } = await api.query.system.account.at(prevHash, ANAELELTD);
24
25 console.log('AnaelleLTD had a balance of', prev.toHuman(), '(600 blocks ago aka 1 hour ago)');
26
```

```
AnaelleLTD has a current balance of 5,038,674,509,552
AnaelleLTD has an updated balance of 5,038,674,509,552
AnaelleLTD had a balance of 5,038,674,509,552 (600 blocks ago aka 1 hour
AnaelleLTD has an updated balance of 5,038,117,176,704
AnaelleLTD has a current balance of 5038117176704
AnaelleLTD has an updated balance of 5,038,117,176,704
AnaelleLTD had a balance of 5,038,674,509,552 (600 blocks ago aka 1 hour
```



### c) Back up code snippets.

JavaScript

Console

Select example  
Retrieve historic query data

1. Add more **custom-made code** to your script, then run the script to show the results, and clear the console.

```
1 // All code is wrapped within an async closure,
2 // allowing access to api, hashing, types, util.
3 // (async ({ api, hashing, types, util }) => {
4 //   ... any user code is executed here ...
5 // })();
6
7 const AnaelleLTD = 'HjcErRijmpoiBiKEHT3edPXM3NFycJogVwDPuByNe7hv9Ae';
8 // Retrieve the initial balance. Since the call has no callback, it is simply a promise
9 // that resolves to the current on-chain value
10 let { data: { free: previousFree }, nonce: previousNonce } = await api.query.system.account(AnaelleLTD);
11
12 console.log(`This account has a balance of ${previousFree}, nonce ${previousNonce}`);
13 console.log(`Listening to balance changes of ${AnaelleLTD}`);
14
15 // Here we subscribe to any balance changes and update the on-screen value
16 api.query.system.account(AnaelleLTD, ({ data: { free: currentFree }, nonce: currentNonce }) => {
17   // Calculate the delta
18   const change = currentFree.sub(previousFree);
19
20   // Only display positive value changes (Since we are pulling `previous` above already,
21   // the initial balance change will also be zero)
22   if (!change.isZero()) {
23     console.log(`New balance change of ${change}, nonce ${currentNonce}`);
24
25     previousFree = currentFree;
26     previousNonce = currentNonce;
27   }
28 };
```

This account has a balance of 5038117176704, nonce 220  
Listening to balance changes of HjcErRijmpoiBiKEHT3edPXM3NFycJogVwDPuByNe7hv9Ae

Kusama  
kusama/9130  
#10,585,438

Accounts ▾

Network ▾

Governance ▾

Developer ▾

Settings

GitHub

Wiki

JavaScript

Console

Select example  
Retrieve historic query data

```
1 // All code is wrapped within an async closure,
2 // allowing access to api, hashing, types, util.
3 // (async ({ api, hashing, types, util }) => {
4 //   ... any user code is executed here ...
5 // })();
6
7 const AnaelleLTD = 'HjcErRijmpoiBiKEHT3edPXM3NFycJogVwDPu
8 // Retrieve the initial balance. Since the call has no call
9 // that resolves to the current on-chain value
10 let { data: { free: previousFree }, nonce: previousNonce } = api.query.system.account(AnaelleLTD);
11
12 console.log(`This account has`);
13 console.log(`Listening to balance changes of ${AnaelleLTD}`);
14
15 // Here we subscribe to any balance changes and update the on-screen value
16 api.query.system.account(AnaelleLTD, ({ data: { free: currentFree }, nonce: currentNonce }) => {
17   // Calculate the delta
18   const change = currentFree.sub(previousFree);
19
20   // Only display positive value changes (Since we are pulling `previous` above already,
21   // the initial balance change will also be zero)
22   if (!change.isZero()) {
23     console.log(`New balance change of ${change}, nonce ${currentNonce}`);
24
25     previousFree = currentFree;
```

BalancesSubs

Save snippet to local storage

2. Click on the **save** button to back up your custom snippet locally.

3. Enter a name and click on **Save snippet** to finish the procedure.

 Kusama  
kusama/9130  
#10,585,442

Accounts ▾

Network ▾

Governance ▾

Developer ▾

⚙ Settings

🔗 GitHub

📖 Wiki


#### 4. Your snippet has been renamed!

Select example  
BalancesSubs

5. Click on the **dropdown arrow** to check the list of available snippets.



```
1 // All code is wrapped within an async closure,
2 // allowing access to api, hashing, types, util.
3 // (async ({ api, hashing, types, util }) => {
4 //   ... any user code is executed here ...
5 // })();
6
7 const AnaelleLTD = 'HjcErRijmpoiBiKEHT3edPXM3NFycJogVwDPuByNe7hv9Ae';
8 // Retrieve the initial balance. Since the call has no callback, it is simply a promise
9 // that resolves to the current on-chain value
10 let { data: { free: previousFree }, nonce: previousNonce } = await api.query.system.account(AnaelleLTD);
11
12 console.log(`This account has a balance of ${previousFree}, nonce ${previousNonce}`);
13 console.log(`Listening to balance changes of ${AnaelleLTD}`);
14
15 // Here we subscribe to any balance changes and update the on-screen value
16 api.query.system.account(AnaelleLTD, ({ data: { free: currentFree }, nonce: currentNonce }) => {
17   // Calculate the delta
18   const change = currentFree.sub(previousFree);
19
20   // Only display positive value changes (Since we are pulling `previous` above already,
21   // the initial balance change will also be zero)
22   if (!change.isZero()) {
23     console.log(`New balance change of ${change}, nonce ${currentNonce}`);
24
25     previousFree = currentFree;
```

 Kusama  
kusama/9130  
#10,585,544

Accounts ▾

Network ▾

Governance ▾

Developer ▾

Settings

GitHub

Wiki

JavaScript

Console

## 6. Your snippet is now (temporarily) stored in the dropdown menu!

BalancesSubs

Custom

Get authoring information

Listen to new Head

Get state metadata

Get system information

RPC

RPC

RPC

RPC

```
12 console.log( `This account has a balance of ${previousFree}, nonce ${previousNonce} `);
13 console.log(`Listening to balance changes of ${AnaelleLTD}`);
14
15 // Here we subscribe to any balance changes and update the on-screen value
16 api.query.system.account(AnaelleLTD, ({ data: { free: currentFree }, nonce: currentNonce }) => {
17   // Calculate the delta
18   const change = currentFree.sub(previousFree);
19
20   // Only display positive value changes (Since we are pulling `previous` above already,
21   // the initial balance change will also be zero)
22   if (!change.isZero()) {
23     console.log(`New balance change of ${change}, nonce ${currentNonce}`);
24
25     previousFree = currentFree;
```

Prepared by Anaelle LTD