# cf-specs Documentation

## Counterfactual Contributors

**Dec 12, 2019**

# Contents:

CHAPTER 1

# Introduction

The Node contains a TypeScript implementation of the Counterfactual protocol. It is responsible for executing the protocols and producing correctly constructed signed commitments that correspond to state transitions of the users' state channels.

The main areas of interest with this implementation are:

- `src/machine/instruction-executor.ts`: This is the entry point for running a protocol.
- `src/protocol` is where all the protocols are implemented
- `src/ethereum` is where the structure of the commitments is placed. It's under this folder as these commitments adhere to the Ethereum network's interface for transactions.

The specific design philosophy for this implementation is the middleware pattern. That is, all of these protocols are naturally broken down into steps, for each of which there is a middleware responsible for executing that step.

Given this design, it's easy to extend this implementation to support additional protocols, replace a default middleware with an alternative implementation, and for the machine to rely on yet delegate more advanced logic to external services.

Some specific examples of this include:

- delegating to a signing module that verifies whether a given action is safe to sign & countersign
- storing state commitments (delegating to an arbitrary, possibly non-local service implementing a desired interface)
- implementing a custom Write-Ahead-Log to tweak performance/security properties

We have *some diagrams* explaining the Node's architecture and control flow.

## 1.1 Node Address and Signing Keys

In order for the Node to produce state-update commitments, it needs access to some signing keys.

There are two ways in which this is supported:

1. An extended private key string is provided to the `Node` at the "EXTENDED_PRIVATE_KEY_PATH" key of the store service that is passed as an argument to the `Node`. If no such value exists for this key, the `Node` produces an extended private key and sets it at this key. This extended private key is then used to generate a "public identifer" for the `Node` (the address by which the `Node` is known by). It is also used to generate private keys which are specific to `AppInstance`s.
2. Instead of supplying a mnemonic, the `Node` operator supplies two other arguments:

- an extended public key which will serve as the "public identifier" of the `Node`, and will be used to generate signer addresses at `AppInstance`-specific derivation paths for signature verification in the protocols.
- a callback function that offers the generation of a private key given a specific derivation path. This enables the consumer of the `Node` (i.e. wallets) to not reveal any mnemonics but provide the ability to sign state isolated to specific `AppInstance`s.

The `Node` package exposes a reference implementation of the second approach through a function named `generatePrivateKeyGeneratorAndXPubPair` which produces these 2 arguments given an extended private key.

## 1.2 Apps and OutcomeTypes

Each application that is installed in a channel has an `OutcomeType` that defines when the app reaches a terminal state and is about to be uninstalled how the funds allocated to it will be distributed.

The currently supported outcome types are:

- TWO_PARTY_FIXED_OUTCOME
  - This is only used when the installed app is collateralized with ETH (for now) and indicates that the total amount allocated to the app will be sent to one of the two parties OR gets split evenly.
- MULTI_ASSET_MULTI_PARTY_COIN_TRANSFER
  - This is used for transferring arbitrary amounts (limited by app collateral) of arbitrary asset classes (ETH or ERC20) to some addresses.
- SINGLE_ASSET_TWO_PARTY_COIN_TRANSFER
  - This is used for transferring arbitrary amounts (limited by app collateral) of a single asset class (ETH or ERC20) to some addresses.

## 1.3 Note:

Any consumer of the Node should set up a handler for the event `DEPOSIT_CONFIRMED` so as to define how this Node behaves when a counter party has initiated a deposit and is asking this Node to make a counter deposit and collateralize the channel. The parameters passed with this event correspond to the same ones used by the initiator, tha is `DepositParams` (as defined in the `@counterfactual/types packages`).

If no such handler is defined, `No event handler for counter depositing into channel <info>` is printed indicating the Node does not know how to handle a counter deposit request.

CHAPTER 2

# Node API

The Node exposes a JSON-RPC compliant API. This can be consumed by making RPC calls and by registering event listeners.

## 2.1 Message Format

A "message" is a return value of an RPC call or the value passed to an event listener. Messages have the following fields:

- `type: string`
  - Name of the Method or Event that this message represents e.g. "getAppInstances", "install"
- `requestId?: string`
  - Unique ID for a Method request.
  - Only required for Methods. Leave empty for Events.
- `data: { [key: string]: any }`
  - Data payload for this message.
  - See "Result" section of a Method and "Data" section of an Event for details.

## 2.2 Public RPC Methods

### 2.2.1 Method: `getAppInstances`

Returns all app instances currently installed on the Node.

NOTE: This is terrible from a security perspective. In the future this method will be changed or deprecated to fix the security flaw.

Params: None

Result:

- `appInstances:`*`AppInstanceInfo`*`[]`
  - All the app instances installed on the Node

### 2.2.2 Method: `proposeInstall`

Requests that a peer start the install protocol for an app instance. At the same time, authorize the installation of that app instance, and generate and return a fresh ID for it. If the peer accepts and the install protocol completes, its ID should be the generated appInstanceId.

Params:

- `proposedToIdentifier: string`
  - Public identifier of the peer responding to the installation request of the app
- `appDefinition: string`
  - On-chain address of App Definition contract
- `abiEncodings:`*`AppABIEncodings`*
  - ABI encodings used for states and actions of this app
- `initiatorDeposit: BigNumber`
  - Amount of the asset deposited by this user
- `initiatorDepositTokenAddress?: string`
  - An optional string indicating whether an ERC20 token should be used for funding the proposer's side of the app. If not specified, this defaults to ETH.
- `responderDeposit: BigNumber`
  - Amount of the asset deposited by the counterparty
- `responderDepositTokenAddress?: string`
  - An optional string indicating whether an ERC20 token should be used for funding the peer's side of the app. If not specified, this defaults to ETH.
- `timeout: BigNumber`
  - Number of blocks until a submitted state for this app is considered finalized
- `initialState:`*`AppState`*
  - Initial state of app instance

Result:

- `appInstanceId: string`
  - Generated appInstanceId

Errors: (TODO)

- Not enough funds

### 2.2.3 Method: `rejectInstall`

Reject an app instance installation.

Params:

- `appInstanceId: string`
  - ID of the app instance to reject

Result: "OK"

Errors: (TODO)

- Proposed app instance doesn't exist

### 2.2.4 Method: `install`

Install an app instance.

Params:

- `appInstanceId: string`
  - ID of the app instance to install
  - Counterparty must have called `proposedInstall` and generated this ID

Result:

- `appInstance:`*`AppInstanceInfo`*
  - Successfully installed app instance

Errors: (TODO)

- Counterparty rejected installation

### 2.2.5 Method: `installVirtual`

Install a virtual app instance.

Params:

- `appInstanceId: string`
  - ID of the app instance to install
  - Counterparty must have called `proposedInstall` and generated this ID
- `intermediaryIdentifier: string`
  - Node identifier of hub to route the virtual app installation through

Result:

- `appInstance:`*`AppInstanceInfo`*
  - Successfully installed app instance

Errors: (TODO)

- Counterparty rejected installation

### 2.2.6 Method: `getState`

Get the latest state of an app instance.

Params:

- `appInstanceId: string`
  - ID of the app instance to get state of

Result:

- `state:`*`AppState`*
  - Latest state of the app instance

Errors: (TODO)

- App not installed

### 2.2.7 Method: `getAppInstanceDetails`

Get details of an app instance.

Params:

- `appInstanceId: string`
  - ID of the app instance to get details of

Result:

- `appInstance:`*`AppInstanceInfo`*
  - App instance details

### 2.2.8 Method: `getProposedAppInstance`

Get details of a proposed app instance.

Params:

- `appInstanceId: string`
  - ID of the app instance to get details of

Result:

- `appInstance:`*`AppInstanceInfo`*
  - App instance details

### 2.2.9 Method: `takeAction`

Take action on current app state to advance it to a new state.

Params:

- `appInstanceId: string`
  - ID of the app instance for which to take action
- `action:`*`SolidityValueType`*

  - Action to take on the current state

Result:

- `newState:AppState`
  - New app state

Errors: (TODO)

- Illegal action

### 2.2.10 Method: `uninstall`

Uninstall an app instance, paying out users according to the latest signed state.

Params:

- `appInstanceId: string`
  - ID of the app instance to uninstall

Result: "OK"

Errors: (TODO)

- App state not terminal

### 2.2.11 Method: `proposeState`

TODO

### 2.2.12 Method: `acceptState`

TODO

### 2.2.13 Method: `rejectState`

TODO

### 2.2.14 Method: `createChannel`

Creates a channel and returns the determined address of the multisig (does not deploy).

Params:

- `owners: string[]`
  - the addresses who should be the owners of the multisig

Result:

- `CreateChannelTransactionResult`
  - `multisigAddress: string`
    - the address which will hold the state deposits

### 2.2.15 Method: `deployStateDepositHolder`

Deploys a multisignature wallet contract for use by the channel participants. Required step before withdrawal.

Params:

- `multisigAddress: string`
  - address of the multisignature wallet
- `retryCount?: number`
  - the number of times to retry *deploying the multisig* using an expontential backoff period between each successive retry, starting with 1 second. This defaults to 3 if no retry count is provided.

Result:

- `DeployStateDepositHolderResult`
  - `transactionHash: string`
    - the hash of the multisig deployment transaction
      - This can be used to either register a listener for when the transaction has been mined or await the mining.

### 2.2.16 Method: `getChannelAddresses`

Gets the (multisig) addresses of all the channels that are open on the Node.

Result:

- `addresses: string[]`
  - the list of multisig addresses representing the open channels on the Node.

### 2.2.17 Method: `deposit`

If a token address is specified, deposits the specified amount of said token into the channel. Otherwise it defaults to ETH (denominated in Wei).

Params:

- `multisigAddress: string`
- `amount: BigNumber`
- `tokenAddress?: string`

Result:

- `multisigBalance: BigNumber`
  - the updated balance of the multisig

Error:

- "Insufficient funds"

### 2.2.18 Method: `getStateDepositHolderAddress`

Retrieves the address for the state deposit used by the specified owners.

Params:

- `owners: string[]`
  - the addresses who own the state deposit holder

Result:

- `multisigAddress: string`
  - the address of the multisig (i.e. the state deposit holder)

### 2.2.19 Method: `withdraw`

If a token address is specified, withdraws the specified amount of said token from the channel. Otherwise it defaults to ETH (denominated in Wei). The address that the withdrawal is made to is either specified by the `recipient` parameter, or if none is specified defaults to `ethers.utils.computeAddress(ethers.utils.HDNode.fromExtendedKey(nodePublicIdentifier).derivePath("0").publicKey)`. `deployStateDepositHolder` must be called before this method can be used to withdraw funds.

Params:

- `multisigAddress: string`
- `amount: BigNumber`
- `recipient?: string`
- `tokenAddress?: string`

Result:

- `recipient: string`
  - The address to whom the withdrawal is made
- `txHash: string`
  - The hash of the transaction in which the funds are transferred from the state deposit holder to the recipient

Error(s):

- "Insufficient funds"
- "Withdraw Failed"

### 2.2.20 Method: `withdrawCommitment`

This behaves similarly to the `withdraw` call, except it produces a commitment for the withdrawal and returns the commitment instead of sending the commitment to the network.

Params:

- `multisigAddress: string`
- `amount: BigNumber`
- `recipient?: string`
- `tokenAddress?: string`

Result:

- `transaction: { to: string; value: BigNumberish; data: string; }`

Error(s):

- "Insufficient funds"

### 2.2.21 Method: `getFreeBalance`

Gets the free balance AppInstance of the specified channel for the specified token. Defaults to ETH if no token is specified.

Params:

- `multisigAddress: string`
- `tokenAddress?: string`

Result:

```
{
    [s: string]: BigNumber;
};
```

Returns a mapping from address to balance in wei. The address of a node with public identifier `publicIdentifier` is defined as `fromExtendedKey(publicIdentifier).derivePath("0").address`.

Note: calling this with a specific token address will return Zero even if the channel has never had any deposits/withdrawals of that token.

### 2.2.22 Method: `getTokenIndexedFreeBalanceStates`

Gets the free balances for the ETH and ERC20 assets that have been deposited, indexed by token address.

Params:

- `multisigAddress: string`

Result:

```
{
  [tokenAddress: string]: {
    [beneficiary: string]: BigNumber;
  }
};
```

Returns a doubly-nested mapping. The outer mapping is the address of the token for which there is free balance in the channel. The inner mapping is a mapping from address to balance in wei. The address of a node with public identifier `publicIdentifier` is defined as `fromExtendedKey(publicIdentifier).derivePath("0").address`.

## 2.3 Events

### 2.3.1 Event: `depositEvent`

Fired if a deposit has been made by a counter party.

Data:

- `multisigAddress: string`
  - The address of the channel that the deposit was made into.
- `amount: BigNumber`
  - The amount that was deposited by the counter party.

### 2.3.2 Event: `installEvent`

Fired if new app instance was successfully installed.

Data:

- `appInstance:`*`AppInstanceInfo`*
  - Newly installed app instance

### 2.3.3 Event: `rejectInstallEvent`

Fired if installation of a new app instance was rejected.

Data:

- `appInstance:`*`AppInstanceInfo`*
  - Rejected app instance

### 2.3.4 Event: `updateStateEvent`

Fired if app state is successfully updated.

Data:

- `appInstanceId: string`
  - ID of app instance whose app state was updated
- `newState:`*`AppState`*
- `action?:`*`SolidityValueType`*
  - Optional action that was taken to advance from the old state to the new state

### 2.3.5 Event: `uninstallEvent`

Fired if app instance is successfully uninstalled

Data:

- `appInstance:`*`AppInstanceInfo`*
  - Uninstalled app instance

### 2.3.6 Event: `proposeStateEvent`

TODO

### 2.3.7 Event: `rejectStateEvent`

TODO

### 2.3.8 Event: `createChannelEvent`

Fired when a Node receives a message about a recently-created channel whose multisignature wallet's list of owners includes this (i.e. the receiving) Node's address.

Note: On the Node calling the creation of the channel, this event *must* have a registered callback to receive the multisig address *before* the channel creation call is made to prevent race conditions.

Data:

- `CreateChannelResult`
  - `counterpartyXpub: string`
    - Xpub of the counterparty that the channel was opened with
  - `multisigAddress: string`
    - The address of the multisig that was created
  - `owners: string[]`
    - The list of multisig owners of the created channel

## 2.4 Data Types

### 2.4.1 Data Type: `AppInstanceInfo`

An instance of an installed app.

- `id: string`
  - Opaque identifier used to refer to this app instance
  - No two distinct app instances (even in different channels) may share the same ID
- `appDefinition: string`
  - On-chain address of App Definition contract
- `abiEncodings:`*`AppABIEncodings`*
  - ABI encodings used for states and actions of this app
- `initiatorDeposit: BigNumber`
  - Amount of the asset deposited by this user
- `responderDeposit: BigNumber`
  - Amount of the asset deposited by the counterparty
- `timeout: BigNumber`
  - Number of blocks until a submitted state for this app is considered finalized
- `intermediaryIdentifier?: string`
  - Xpub of a hub to route the virtual app installation through. Undefined if app instance is not virtual.

### 2.4.2 Data Type: `AppABIEncodings`

- `stateEncoding: string`
  - ABI encoding of the app state
    - For example, for the Tic Tac Toe App (https://github.com/counterfactual/monorepo/blob/master/packages/apps/contracts the state encoding string is `"tuple(uint256 versionNumber, uint256 winner, uint256[3][3] board)"`.
- `actionEncoding?: string`
  - Optional ABI encoding of the app action
  - If left blank, instances of the app will only be able to update state using *`proposeState`*
  - If supplied, instances of this app will also be able to update state using *`takeAction`*
    - Again, for the Tic Tac Toe App, the action encoding string is `"tuple(uint8 actionType, uint256 playX, uint256 playY, tuple(uint8 winClaimType, uint256 idx) winClaim)"`.

### 2.4.3 Data Type: `AppState`

- Plain Old Javascript Object representation of the state of an app instance.
- ABI encoded/decoded using the `stateEncoding` field on the instance's *`AppABIEncodings`*.

### 2.4.4 Data Type: `SolidityValueType`

- Plain Old Javascript Object representation of the action of an app instance.
- ABI encoded/decoded using the `actionEncoding` field on the instance's *`AppABIEncodings`*.

CHAPTER 3

# Diagrams

These diagrams are available to help you understand the underlying architecture of the Node.

## 3.1 Ownership

arrows indicate “has a pointer to”

## 3.2 Control Flow

arrows mostly indicate “calls”