sbt-ethereum

> a terminal for the world computer

steve randy waldman — swaldman@mchange.com — https://www.sbt-ethereum.io/ — https://www.interfluidity.com/ — @interfluidity

the blockchain is the dApp

Remember this?

...The Times 03/Jan/2009 Chancellor
on brink of second bailout for banks...

meanwhile...

- We've internalized the infantilizing norms of contemporary Silicon Valley.
- Just wait for "us" (the "devs!", the "team!", "VCs!") to build you ("end users!") something complicated and awesome!

meanwhile...

- It's gonna take a lot of time and money, because "end users!" need an awesome "UX!"
- Every experiment requires a funded startup and scale sufficient to justify that
- So we run tens of big, expensive experiments rather than thousands of small, cheap ones

the blockchain is the dApp

But...

- *Ethereum* smart contracts expose a UI automatically
  - *It's called an ABI*
- Smart contracts take an order of magnitude less effort to write than the Web, mobile, and UX stuff in which people surround them
- We should prefer a world with many small-scale economic arrangements to one with a few, standard large-scale ones

the blockchain is the dApp

- Sophisticated "end users" can deploy and interact with smart contracts directly, and take full control
- Less sophisticated users can rely upon humans whom they directly know as helpers and intermediaries
- Eventually, intermediary roles can be smoothed and automated away. But that's eventually.

sbt-ethereum

- A convenient, high-level, text-based user interface for interacting with *Ethereum* and compatible blockchains
- A *smart-contract development* and deployment tool
- A high-performance framework for integrating smart contracts into *Scala* applications
- A platform for developing *app-specific CLIs*

very stateful

sbt-ethereum collects and retains...

- Node URLs
- Wallets, addresses, and address aliases
- ABIs and ABI aliases
- Default mappings of ABIs to smart contracts
- Complete compilation info about deployed contracts

friendly (sort of)

- Tab-completey
- Often interactive
- Very long but descriptive names
- Consistent internal conventions
  - Default values and session overrides
  - Set, Drop, Print

friendly (sort of)

Get started with a few basic commands

```
> ethContractAbiImport <address-as-hex-alias-or-ens>
> ethTransactionView <address-as-hex-alias-or-ens> <function-args>*
> ethTransactionInvoke <address-as-hex-alias-or-ens> <function-args>*
> ethTransactionEtherSend <address-as-hex-alias-or-ens>
> ethAddressBalance [optional-address-as-hex-alias-or-ens]
> ethAddressAliasSet <alias-name> <address-as-hex-alias-or-ens>
```

batteries included

ENS

- *ENS can be used in place of addresses* and address aliases
- Acts as a *full ENS client, including registering names*, extending registrations, creating subnodes, transfering ownership, etc.

batteries included

ERC-20

- Built in support for managing ERC-20 tokens using human-friendly values as defined in the `decimals()` function

Etherscan

- Autoimport ABIs of verified contracts if an *Etherscan* API key has bee set.

powerful

- Full smart-contract development environment
- Supports signing for EIP-155 chain IDs and seamless switching between chains
- Offline transaction-signing for cold wallets
- Sophisticated control of gas and nonces
- Name and store reusable ABIs
  - Overlay arbitrary ABIs on top of any contract

programming (Scala-centric)

- Automatic stub generation
  - Thread-pool managed async stubs or easy-to-understand synchronous stubs
  - Solidity-like embedded DSL
  - Solidity events become typesafe, pattern-matchable Scala objects
  - Standard "reactive" filter-free event subscriptions

programming (Scala-centric)

```
contract DocHashStore {
  event Stored( bytes32 docHash, uint timestamp, string name, string description, address filer );
  event Amended( bytes32 docHash, string name, string description, address updater, uint priorUpdateBlockNumber );
  event Opened( address admin, uint timestamp );
  event Closed( address closer, uint timestamp );
  event Authorized( address user );
  event Deauthorized( address user );

  address public admin;
  bytes32[] public docHashes;
  mapping ( address => bool ) public authorized;
  uint public openTime;
  uint public closeTime;
  bool public closed;

  function close() public;
  function authorize( address filer ) public;
  function deauthorize( address filer ) public;
  function canUpdate( address user ) public view returns (bool);
  function store( bytes32 docHash, string memory name, string memory description ) public;
  function amend( bytes32 docHash, string memory name, string memory description ) public;
  function isStored( bytes32 docHash ) public view returns (bool);
  function timestamp( bytes32 docHash ) public view returns (uint);
  function name( bytes32 docHash ) public view returns (string memory);
  function description( bytes32 docHash ) public view returns (string memory);
  function filer( bytes32 docHash ) public view returns (address);
  function size() public view returns (uint);
}
```

programming (Scala-centric)

```
// for simplicity, this example builds a synchronous DocStoreHash
// if we called AsyncDocStoreHash.build(...) instead, the same code would work
// but all stub return values would be Futures

val docstore = DocHashStore.build( jsonRpcUrl      = "https://mainnet.infura.com/v3/20963efa809b0178",
                                   chainId         = Some(EthChainId.Mainnet),
                                   contractAddress = EthAddress("0x1a4934109b54911a724dfa0e45d5370dbbe923b0") )

implicit val sender = stub.Sender.Basic( somePrivateKey )

val sz = docstore.view.size()

val docHash = sol.Bytes32( "0x00e2b1120d2c76a3b44640c325681c892dd3a1fcb33bf412169a2c17f5e0c171".decodeHex )
val txnInfo = docstore.txn.store( docHash, "ImportantDocument.pdf", "This is a really important document" )
```

programming (Scala-centric)

```
// inside a standard org.reactivestreams.Subscriber[DocHashStore.Event]

def onNext(evt : DocHashStore.Event) = {
  evt match {
    case _ : Stored | _ : Amended => markDirtyDocRecordSeq( address )
    case _ : Closed => {
      markDirtyOpenClose( address )
      subscriptionRef.get.foreach( _.cancel() )
      drop( address )
    }
    case evt @ Authorized( userAddress )   => markDirtyUserCanUpdate( evt.sourceAddress, userAddress )
    case evt @ Deauthorized( userAddress ) => markDirtyUserCanUpdate( evt.sourceAddress, userAddress )
    case _                                 => DEBUG.log( s"${this} encountered and ignored event ${evt}" )
  }
}
```


demo

support

- Decent documentation at www.sbt-ethereum.io
- Tag sbt-ethereum on ethereum.stackexchange.com
- DM @interfluidity on Twitter
- E-mail swaldman@mchange.com
- swaldman/sbt-ethereum on GitHub

support me

- Use the software
- Tell me what sucks so I can fix it
  - *especially if anything sucks related to security*
- If you want to offer financial support, get in touch, or contribute to sbt-ethereum.eth

acknowledgments

Waiting for Godot image nicked from

- https://www.onecolumbiasc.com/event/waiting-for-godot/