Getting Started with Factom on Linux

This guide is for technical userers who are interested in learning about Factom. By following this guide you will learn the basics of using Factom, and building simple applications that read and write data to and from the Factom Blockchain.

In this guide you will learn how to: *install Factom* understand Factom data structures and design patterns *buy Factoids* convert Factoids into Entry Credits *create Factom Chains and Entries* read data from Factom * create a simple Factom application

Make sure you understand...

This guide uses factom-cli commands on a command shell as well as library/api calls in some example applications. You will need to be familier with opening a shell and issuing commands to use the factom-cli to create and read Factom Entries. You do not need to be an expert programmer but you will need to understand some common programming concepts to create and run applications that interact with Factom.

Installing Factom

The Fastest way to install Factom is to use the Factom installation package provided by http://factom.org. The package contains binaries for factomd, fctwallet, and factom-cli.

Downloads the factom installer package for Debian GNU/Linux.

$ wget http://factom.org/downloads/factom.deb

Run the Factom Installer. The binaries have been built for 32 bit systems to ensure compatability with older hardware, so on 64 bit systems you must add the `--force-architecture` option when installing.

$ sudo dpkg --force-architecture -i ./factom.deb

Check that the packages have been installed into their correct locations.

$ which factomd /usr/bin/factomd

$ which fctwallet /usr/bin/fctwallet

$ which factom-cli /usr/bin/factom-cli

Once the Factom Binaries have been installed successfully run `$ factomd` and let it sync with the Factom Network. This may take a little time if it is the first time running factomd on a new machiene.

Buying Factoids

Section comming soon.

Converting Factoids into Entry Credits

Entry Credits allow data to be written into Factom. Each Entry Credit Address represents a key pair that allows signing of payments for data to be written into the Factom newtork. Factoids are converted into Entry Credits by adding 1 or more ecoutputs to a transaction.

Run `$ factomd` and `$ fctwallet`.

Use factomcli to create an Entry Credit address. The ec address in this example will be called app01.

$ factom-cli generateaddress ec app01 ec = EC2gigrpHsADYXbnGDhBf58z8isuiT8HffZT1gFfcQERzon4SD44

Create a new transaction 'a'.

$ factom-cli newtransaction a

Add 10 factoids as an input to the transaction from the factoid address we created earlier.

$ factom-cli addinput a fct01 10

Add 10 factoids as the ammout to convert into Entry Credits.

$ factom-cli addecoutput a app01 10

Pay the transaction fee from the same factoid address

$ factom-cli addfee a fct01

Sign and submit the transaction to the Factom network.

$ factom-cli sign a $ factom-cli submit a

After 10 minutes you should see the Entry Credits at the new address.

$ factom-cli balance ec app01 Balance of app01 = 1010

Once the Entry Credit address has been loaded with credits it may be used to create Factom Entries and Chains.

Understanding Factom data structures.

Entries

User data in the Factom network is organized into Entries and Chains. A Factom Entry is composed of a ChainID, 0 or more External IDs, and the Entry Content. The External IDs and Content are binary data but it is most common to write decoded text into these fields. It is up to the application to interperate the Entries. A Factom application might wright any data into the External IDs and Entry Content then parse or interperate the data any way it likes.

Chains

Factom Chains are a series of Factom Entries. When a new Entry is commited and revealed to the Factom network it is added to the an Entry Block for its specified Chain. At the end of the 10 minute period all of the new Entry Blocks are combined into the directory block, then anchored into the Bitcoin Blockchain.

Hello World!

Creating a new Factom Entry

In the first example a new Entry is constructed then sent to the Factom network.

package main

import ("log" "time"

"github.com/FactomProject/factom")

func main() { e := factom.NewEntry() e.ChainID = "5c337e9010600c415d2cd259ed0bf904e35666483277664d869a98189b35ca81" e.ExtIDs = append(e.ExtIDs, []byte("hello")) e.Content = []byte("Hello Factom!")

if err := factom.CommitEntry(e, "app01"); err != nil { log.Fatal(err) } time.Sleep(10 * time.Second) if err := factom.RevealEntry(e); err != nil { log.Fatal(err) } }

The easiest way to create Factom applications in golang is to import the factom package.

import (//...
"github.com/FactomProject/factom")

Create a new `factom.Entry` and fill in the relevent data. We will be adding this Entry to a testing Chain `5c337e9010600c415d2cd259ed0bf904e35666483277664d869a98189b35ca81` . the first External ID for the Entry will be "hello" and the Entry Content will be "Hello Factom!".

e := factom.NewEntry() e.ChainID = "5c337e9010600c415d2cd259ed0bf904e35666483277664d869a98189b35ca81" e.ExtIDs = append(e.ExtIDs, []byte("hello")) e.Content = []byte("Hello Factom!")

Once the Entry is ready we send the Commit Message to the Factom network. The Commit is process by fctwallet and signed with the Entry Credit Address specified here.

if err := factom.CommitEntry(e, "app01"); err != nil { log.Fatal(err) }

It is not strictly nessesary to wait between the Commit Message and the Reveal, but waiting reduces the chance of errors. When we are ready we reveal the Entry.

if err := factom.RevealEntry(e); err != nil { log.Fatal(err) }

If there are no errors, the Entry will be included in the current 10 minute Entry Block for the specified chain. After the end of the current 10 minutes the Entry Block containing the Entry will be hashed and included into the Directory Block which will be anchored into the Bitcoin Blockchain.

Creating a new Factom Chain

A new Factom Chain is created by constructing an Entry to be the first Entry of the new Chain, then constructing the Chain from the Entry. The Chain is then commited and revealed to the Factom network.

package main

import ("log" "time"

"github.com/FactomProject/factom")

func main() { e := factom.NewEntry() e.ExtIDs = append(e.ExtIDs, []byte("MyChain"), []byte("12345")) e.Content = []byte("Hello Factom!")

c := factom.NewChain(e) log.Println("Creating new Chain:", c.ChainID)

if err := factom.CommitChain(c, "app01"); err != nil { log.Fatal(err) } time.Sleep(10 * time.Second) if err := factom.RevealChain(c); err != nil { log.Fatal(err) } }

Since a new Chain is being created the Entry may be constructed without the ChainID field. The new ChainID will be computed using the ExtIDs of the Entry. Remember that the ExtIDs of the first Entry of a Chain must be unique among all first Entries (A new Chain cannot be created if a Chain with the same ID already exists).

e := factom.NewEntry() e.ExtIDs = append(e.ExtIDs, []byte("MyChain"), []byte("12345")) e.Content = []byte("Hello Factom!")

c := factom.NewChain(e)

The ChainID will be printed to the screen.

log.Println("Creating new Chain:", c.ChainID)

```
Creating new Chain: cfa35f22d4790a3f3121d6cc192da26813ee29cb0f8ad220fbe3563fa9d351d1
```

Reading data from Factom

Key signed application