Recurring Payments ('Subscriptions') smart contract system by INTONATION

Main repo: https://github.com/INTONNATION/SubscriptionManager
Free TON community repo: https://github.com/freeton-org/devex/pull/5
Network: fld.ton.dev (https://gql.custler.net)
Client DeBot: 0:497cfc5a9231cf502308ac98acf0b7b152e47300518106c4e9c42a301a8fbe30
Service Debot: 0:ea9f4601a41d84c091da3ad1a8b35f7f4d3f90be64a12f6d7fbf4e724a1edc77

Telegram contacts
@renatSK
@sostrovskyi
@azavodovskyi

Motivation

Monthly subscriptions are a key monetization channel for legacy web, and arguably they are the most healthy monetization channel for businesses (especially when compared to ad/surveillance) based models. But in the blockchain world there haven't been any successful full-featured crypto subscription payment systems yet. We aim to change this. In our project called "Subscription Manager" we focused on user experience and user interface(DeBot) to access this system. We believe it's a key to mass adoption. Goal of our subscription system is to move users and existing subscription services on Free TON with the closest to legacy web experience but with a higher level of security and guarantee. Currently we focused on subscription by TON Crystal to simplify user access to this system (in case of TIP3 users will need to have TON Crystal for gas payment and TIP3 token to subscribe), but proposed architecture can be extended in future to support TIP3 wallets as well.

Usage

There is a demo smart-contract system deployed to fld.ton.dev. You can access it using either debash or any other DeBot browser.

NOTE*:* To get some tokens on your wallet for testing in fld.ton.dev please use the following command:
tonos-cli --url https://gql.custler.net call --abi ../local_giver.abi.json 0:841288ed3b55d9cdafa806807f02a0ae0c169aa5edfe88a789a6482429756a94 sendGrams "{\"dest\":\"< your wallet >\",\"amount\":10000000000}"
ABI is placed in the contest repository.

Access using DeBash

NOTE: don't forget to specify your public key and wallet for UserInfo debot interface while testing with tonos-cli.

Example:
tonos-cli config --pubkey 0x5247ff4f623ccfcbf36ccbaa4c209b89fc5930bc32cca366f6276617e8f3e7b2 --wallet 0:99003f095dbb8a13333f6aabd2a3dd55998b8a5a08c41e78269bddf937e580da

Client DeBot - *tonos-cli --url http://gql.custler.net debot fetch 0:497cfc5a9231cf502308ac98acf0b7b152e47300518106c4e9c42a301a8fbe30*
Service DeBot - *tonos-cli --url http://gql.custler.net debot fetch 0:ea9f4601a41d84c091da3ad1a8b35f7f4d3f90be64a12f6d7fbf4e724a1edc77*

Access using TON Surf

In order to add fld.ton.dev to TON Surf networks navigate to Advanced Settings -> Network -> Push add (+) button in the upper right corner and configure fld.ton.dev with the following endpoint - *https://gql.custler.net*. Select the created network and browse for DeBots using addresses specified.

NOTE: It's recommended to use 2 different accounts (wallets) - one for clients and one for service to test real use cases.
NOTE: You can get 508 timeout errors while testing. This means fld.ton.dev DApp server hangs. Just try one more time. But anyway it's more stable than net.ton.dev.

Server side

If you are an on-chain/off-chain service provider you can register your service in our smart-contract system in order to make it visible for potential subscribers, manage payments, track service usage and statistics.

Service Registration ("Register service" button)

Subscription service DeBot
DeBot

Hello! Use this DeBot to manage your subscription service.

Available options:

Register service

Input the name of your service:

SomeService

Provide description for subscribers:

This service allows you to do something

Input payment period for subscribers (days):

30

Input a cost of your service subscription for selected period (TONs):

10

Input an address to receive payments:

My Rubies

0:af1eec292203b858a374c06992e37d6a3df90b34e2c5362dc5f332cf41b4d53e

Choose your keys to sign transactions from multisig.

Sign with Surf

Get information about your service ("Get Service info" button)

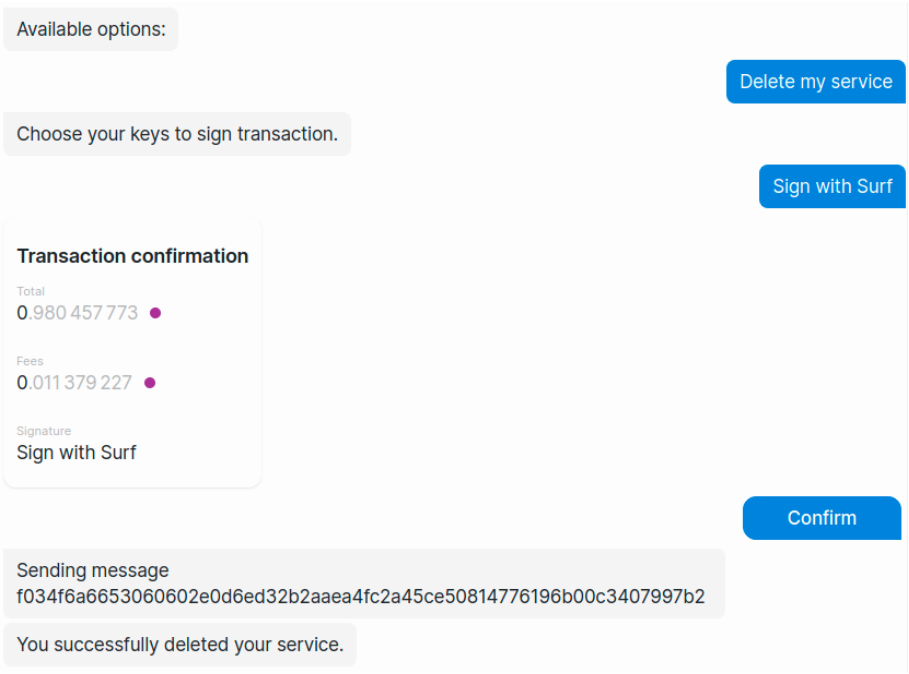
Available options:

Get service info

Name: SomeService
Description: This service allows you to do something
Period: 30
Price per period: 10
Subscribers count: 1
Expected monthly income: 10

Available options:

Delete service registration (“Delete Service” button)



Client side

As a potential customer you can find various on-chain/off-chain services which are registered in our smart-contract system and subscribe. You are able to manage your subscriptions together with a subscription wallet deployed especially for service payments.

Get a list of registered services and subscribe (“Subscribe” button). When you do this for the first time, a subscription wallet will be deployed. Subscription wallet allows only authorized services (those which are in your subscriptions list) to withdraw payments.

Available actions:

Subscribe

5 subscription services has been found.

➤ Autotest 1630443108

➤ Autotest 1630439007

➤ new111

➤ SomeService

➤ new12

Subscribe to the specific service using address

Main menu

Name: SomeService
Description: This service allows you to do something
Period: 30
Price: 10
Address:
0:a1d4ac2af7b9215726832999224941092 2ced0fe451f7e937f34f53c82817f46

Subscribe?

Yes

Choose your keys to sign transactions from your wallet.

Sign with Surf

Transaction confirmation

Total
1.000 000 000

Fees
0.016 464 483

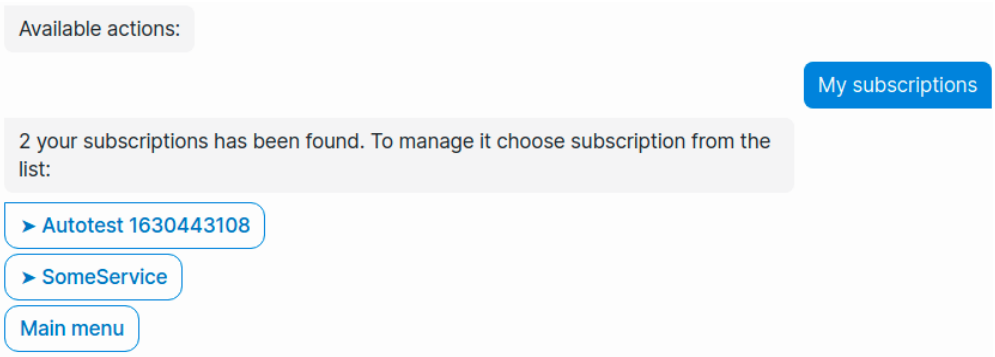
Signature
Sign with Surf

Confirm

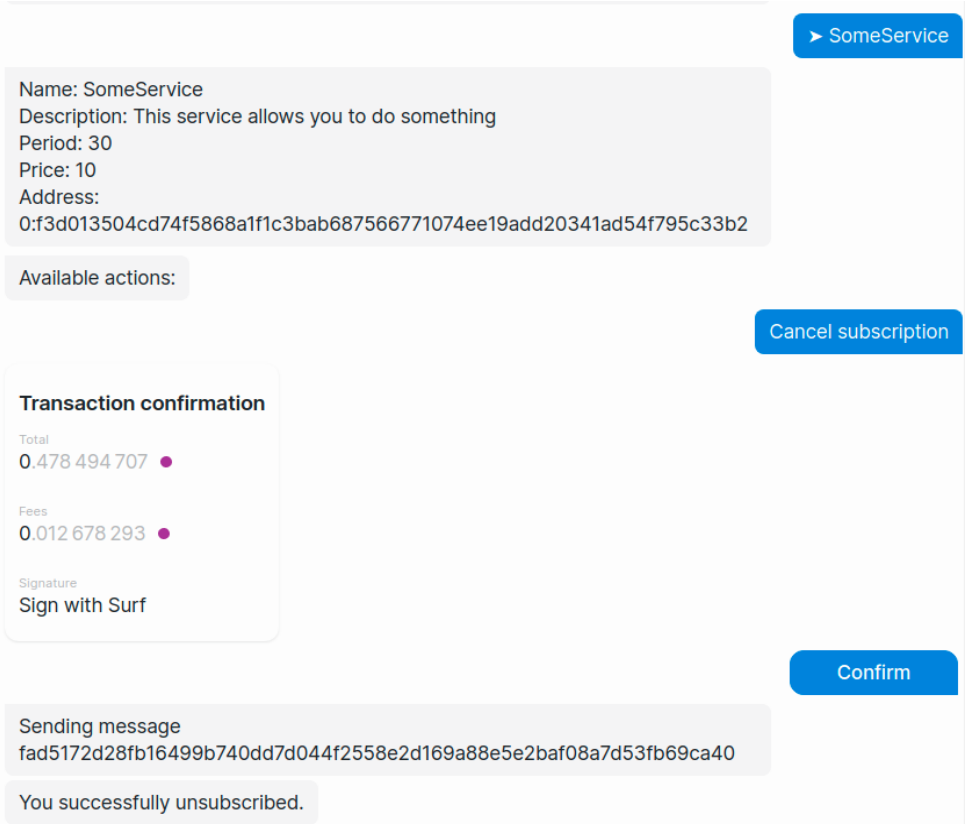
Sending message
7f56455d8337788777f0c9875548d0f95b31419c847ceed7c7039afc26398a75

You successfully subscribed.

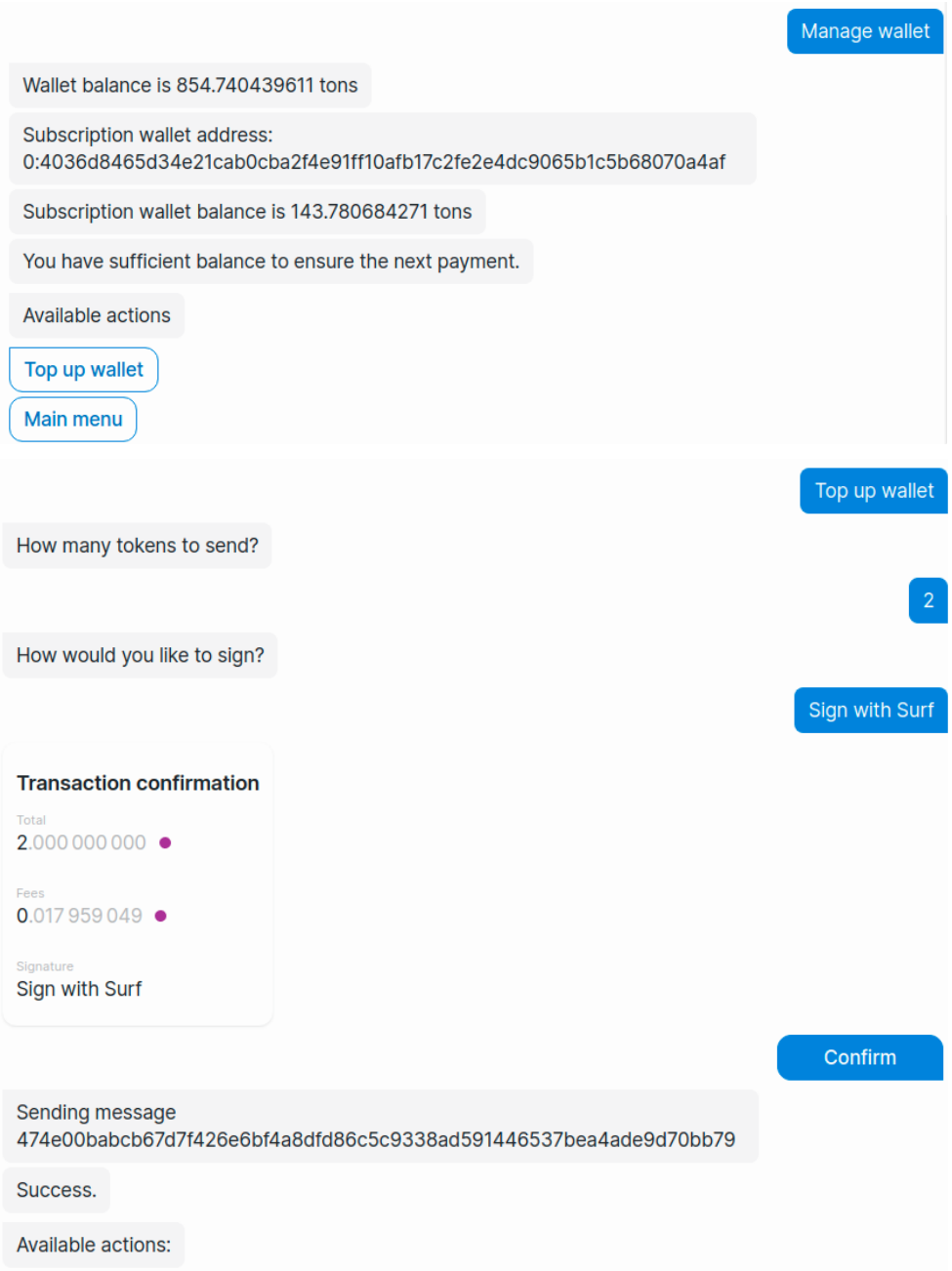
List your subscription (“My subscriptions” button)



Cancel subscription (choose one of subscriptions and push “Cancel subscription” button)



Top up your subscription wallet to have sufficient balance and make the service able to withdraw money from your subscription wallet ("Top up wallet" button). If you don't have a subscription wallet deployed, DeBot will ask to deploy it.



Architecture

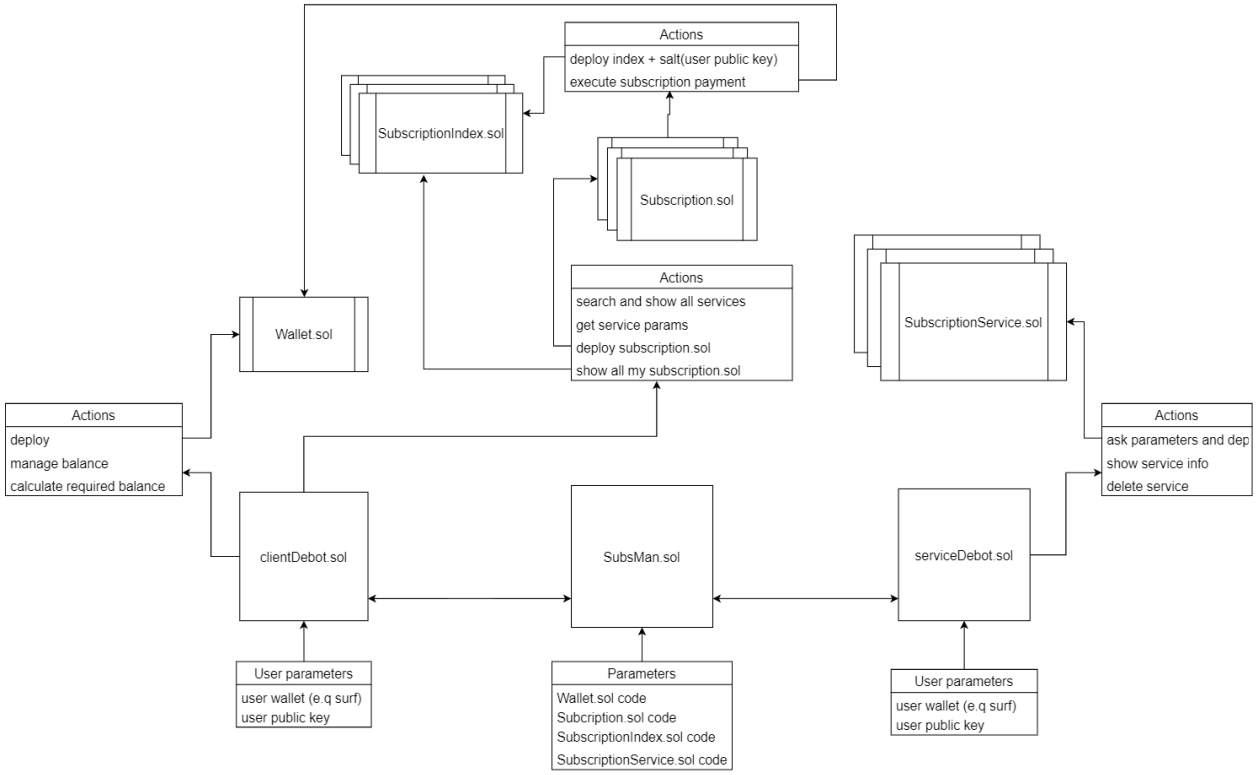


Image 1: [Subscription Manager Architecture diagram (high resolution)]

There are 3 DeBots: clientDebot, SubsMan, serviceDebot. Client and Service DeBots can be accessed by any user but SubsMan is for internal usage only. SubsMan contains all codes of contracts and is used as a single place for address calculation and contracts deployments. This system is called “Subscription Manager”. There are no mappings and loops. All operations work according to the Web Free Paradigm, including address calculations and ability to search all contracts. So this system can be infinitely scalable without increasing gas usage.

In the “Subscription Manager” system one keypair corresponds to one service and users are able to deploy a subscription contract which corresponds to a specific service. So users have as many subscriptions contracts deployed as services he is subscribed to.

Subscription service is just a well known smart contract code which contains information about your service, such as name, description, payment period, price etc. All these parameters will be shown to users. When a user wants to subscribe to a specific service it will get these parameters and deploy a subscription contract based on them. Because anyone knows the code of subscription and subscription service contract, it’s possible to search and find all subscription services in the whole network by one single query.

Additionally, subscriptionIndex contracts are used to have a possibility to search already deployed user subscriptions. In turn each subscription service is able to find all their subscribers as well with a single SDK query.

Additionally each user has a specific wallet contract which can be fully managed with clientDeBot. Subscription wallet is just a regular wallet but with additional functions. It allows users to transfer money from it by any subscription contract which the user owns, so users do not need to trust any contract except it's own.

It's obvious that in any subscription system users need to have a "subscribe and forget experience". To fulfill that we designed a subscription contract where anyone was able to execute the subscription payment.

"Subscription Manager" system supports both types of services: on-chain services (smart contracts) and off-chain (any subscription service which works in legacy web).

For off-chain services it serves the vendor's responsibility to execute payment. Anytime when a user accesses some off-chain service, the vendor can calculate the user subscription contract by code, user publey and params, and verify that this contract exists and his status. In case the user did not pay for the current period yet, service vendors can initiate this transaction. Because this operation requires some gas, these expenses should be included in the subscription price.

For on-chain services there are two possibilities: on-chain service can be designed in such a way that subscription contract status will be checked and ensured immediately before providing data to the caller. In this case service can control who will pay for initiating payment either user or service. But in any case it will happen without any additional steps for the user.

Because of this approach "Subscription Manager" does not require any additional on-chain or off-chain timers.

Subscription contract and SubscriptionIndex codes contain salt and cannot be deployed without it. SubscriptionIndex has a salt equal to tvm.pubkey to add the possibility to find user subscriptions (see 'My subscriptions' in **Usage**). Subscription contract has a code of service to easily find subscribers contracts by service vendor. Security of the system is guaranteed by signing code and verification in constructors of salt, tvm.pubkey and signature. It's impossible to deploy contracts for different users and services. By this approach we ensure that services can find only their subscribers and users can be sure that all found subscription index contracts it's really owned by them. SubscriptionIndex contract simply contains all parameters and links to Subscription contract.

Deployment

Required tools - compiler 0.47.0, linker 0.13.7, stdlib 0.47.0.

There are two deploy scripts for Linux and MacOS:

- fld-deploy_debot.sh
- fld-deploy_debot-mac.sh

Just run them without any parameters. Deploy script generates 2 msig wallets(for client and service), deploy DeBots and store all information about addresses, DeBots and keys into separate files.

NOTE: current scripts work only for fld.ton.dev. Because it's the most stable and suitable network for development. If you want to deploy to net.ton.dev for example, just change a giver address and network variables in a script.

Testing suite

NOTE: deploy contracts to fld.ton.dev network before tests.

We thought a lot about a testing solution for this submission and didn't find any ready to use tools or frameworks which would give us a chance to test DeBots. We decided to propose for the community to use the 'expect' tool for this purpose. It supports command line predictive scenarios with possibility to use with tonos-cli. This tool helps us to test all the DeBots functionality without manual work. Also the 'expect' tool provides us a possibility to avoid any unexpected issues on the TON surf interface or any other DeBot Browser.

There are 5 test cases available for Service and Client debots. It uses expect tool with prepared scenarios:

TestCreateService.exp
TestDeleteService.exp
TestDeploySubscription.exp
TestGetinfoService.exp
TestPopupWallet.exp
TestShowSubscriptions.exp

To install expect tool for your machine:

Linux: https://zoomadmin.com/HowToInstall/UbuntuPackage/expect
OSX: https://formulae.brew.sh/formula/expect

Also we have prepared script run.sh with the full test flow for our DeBots.

Future plans

1. Extend client debot to show transaction history, calculate payment dates and detailed payments estimates
2. Add on-chain service template
3. Add SDK scripts example for off-chain services
4. Add DeNS integration (service name as DeNS record)
5. Add additional party which will be able to control subscription service and cancel subscribers in case of subscription service deleted or stopped to provide service
6. Audit and security