

Development status August 2022

Development is currently focused around the backend functionality, namely the smart contracts. Websolution and front end components will follow when the first version of the complete smart contract has been developed (step 3 in spec below).

Development of smart contracts easily gets impossible to debug if too much functionality is implemented at once. Therefore, a commonly used technique is to develop in a step-stone approach where the first simple contract is created and when it works, it is expanded upon. Transitioning through increasingly complex contracts we end up with the final contract that contains the desired functionality.

To end up with a final contract that supports specific swap of NFTs, we created a specification of contracts needed for the transition.

Specific Swap contract

On chain code

Function / action	High level functionality
Swap (Called for Datum = "Swap")	Verify that the swap transaction consumes one NFT and provides one NFT part of the correct policy id
Clean (Called for Datum = "Clean")	- Verify that all consumed UtxOs contain a NFT with policy id not handled by the swap pool contracts - Verify that the transaction has been signed by the swap pool owner.

Off chain code

Function / action	High level functionality
Send NFT to contract	- Create transaction containing an NFT - Submit the NFT to the contract
Swap NFT	- Create transaction containing UTxO with an NFT_in to send to the contract - The transaction should also consume a contract UTxO with a NFT_out that is wanted in return for NFT_in - Datum must be set to be the action "Swap" - Submit the transaction to the contract
Remove NFT	- Create transaction that consumes all UtxOs containing "garbage" NFTs not part of the policy id handled by the swap pool contract - Datum must be set to be the action "Cleanup" - Submit the transaction to the contract

Contracts to develop

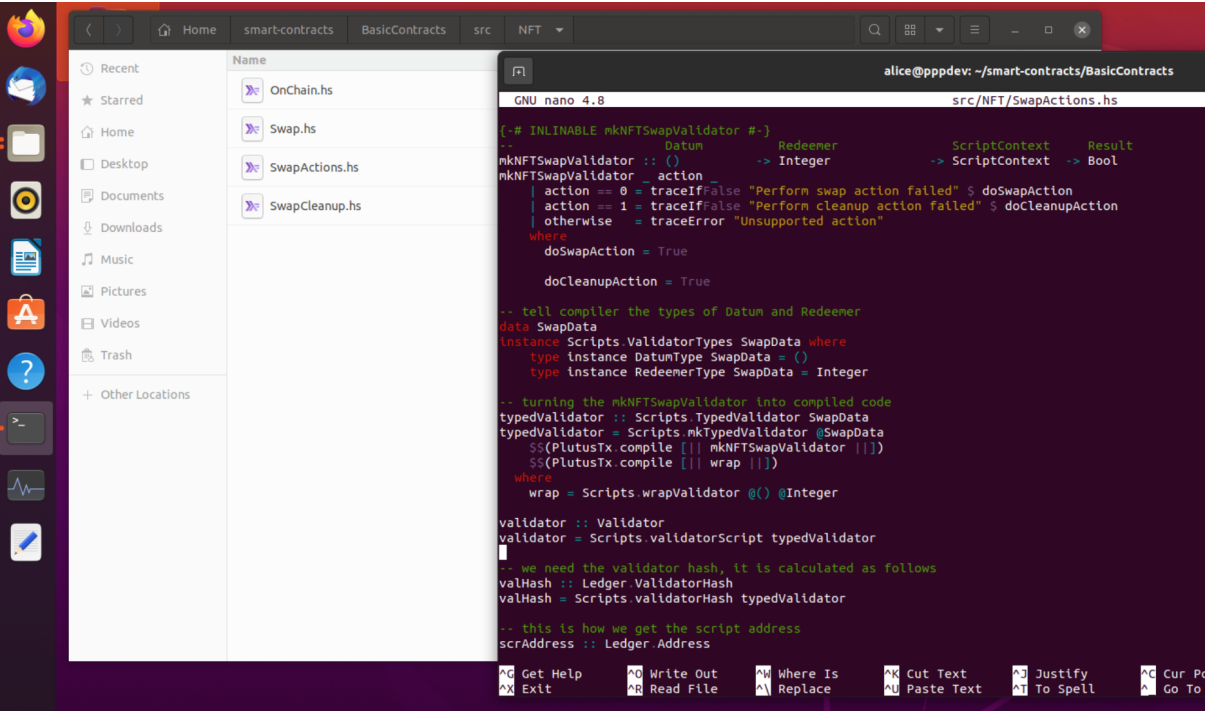
Version name	Functionality
NFT.SwapActions	Contract that calls different code depending on type of Datum received.
NFT.SwapCleanup	Implement "clean" action
NFT.Swap	Implement "swap" action

Currently we have successfully developed and verified the first contract, **NFT.SwapActions**. The contract was verified by publishing it to the Cardano testnet and running transactions with it. Currently we are working on the next contract NFT.SwapCleanup where we are increasing the functionality further.

The contract NFT.SwapActions

High level functionality of this first contract is to simply enable our platform to run different actions towards the contract and repelling actions not supported by the contract. In our contract we currently have two actions; Swap and Cleanup. Transactions trying to consume the contract funds must do this using one of these actions. If trying to consume using other actions, the contract will repel this.

Here is an image of our development environment and the core part of the NFT.SwapActions smart contract code

A screenshot of a Linux desktop environment. On the left is a sidebar with file manager icons. The main window shows a file explorer with a list of files: OnChain.hs, Swap.hs, SwapActions.hs, and SwapCleanup.hs. Overlaid on this is a terminal window running the GNU nano 4.8 text editor. The editor is open to the file src/NFT/SwapActions.hs. The code in the editor is Haskell, defining a validator for the NFT.SwapActions contract. It includes comments in green and code in white. The code defines a mkNFTSwapValidator function, a typedValidator, and a validator script. It also includes a section for calculating the validator hash and the script address.

```
--# INLINEABLE mkNFTSwapValidator #-)
--
mkNFTSwapValidator :: ()
mkNFTSwapValidator = action
  | action == 0 = traceIfFalse "Perform swap action failed" $ doSwapAction
  | action == 1 = traceIfFalse "Perform cleanup action failed" $ doCleanupAction
  | otherwise = traceError "Unsupported action"
where
  doSwapAction = True
  doCleanupAction = True

-- tell compiler the types of Datum and Redeemer
data SwapData
instance Scripts.ValidatorTypes SwapData where
  type instance DatumType SwapData = ()
  type instance RedeemerType SwapData = Integer

-- turning the mkNFTSwapValidator into compiled code
typedValidator :: Scripts.TypedValidator SwapData
typedValidator = Scripts.mkTypedValidator @SwapData
  $$(PlutusTx.compile [| mkNFTSwapValidator |])
  $$(PlutusTx.compile [| wrap |])
where
  wrap = Scripts.wrapValidator @() @Integer

validator :: Validator
validator = Scripts.validatorScript typedValidator

-- we need the validator hash, it is calculated as follows
valHash :: Ledger.ValidatorHash
valHash = Scripts.validatorHash typedValidator

-- this is how we get the script address
scrAddress :: Ledger.Address
```

The contract was published to the Cardano testnet and the address of the contract is
addr_test1wpg4hsycmapv69e6k8j9kygnvwm395j9qrts7uqc0l4lcckjzhut

We will now show some verification transactions done with the contract on the Cardano testnet. All transactions can be viewed also in the Cardano testnet explorer:
https://explorer.cardano-testnet.iohkdev.io/en/address.html?address=addr_test1wpg4hsycmapv69e6k8j9kygnvwm395j9qrts7uqc0l4lcckjzhut

Swap and cleanup actions (successful spending of contract funds)

Before trying to spend, we have transferred some tADA to the contract. Thereafter we try to spend these funds using different actions.

The following is spending using the **Swap** action

```
search01 ~/testnet : cardano-cli transaction build --babbage-era --testnet-magic 1097911063 --tx-in
b19a58d06f89f756688aacec655b754bc9c50ca17dfb83ffde7553311cfc5059#1 --tx-in-script-file smart-contracts/swap-
actions.plutus --tx-in-datum-file smart-contracts/unit.json --tx-in-redeemer-file smart-contracts/redeemer-action-
swap.json --tx-in-collateral b19a58d06f89f756688aacec655b754bc9c50ca17dfb83ffde7553311cfc5059#0 --tx-out $(cat
wallets/swapactions-payment.addr)+9696269 --change-address $(cat wallets/swapactions-payment.addr) --protocol-params-file
protocol-parameters-babbage.json --out-file tx.body
Estimated transaction fee: Lovelace 303731
search01 ~/testnet : cardano-cli transaction sign --tx-body-file tx.body --signing-key-file wallets/swapactions-
payment.skey --testnet-magic 1097911063 --out-file tx.signed
search01 ~/testnet : cardano-cli transaction submit --tx-file tx.signed --testnet-magic 1097911063
Transaction successfully submitted.
search01 ~/testnet :
```

<https://explorer.cardano-testnet.iohkdev.io/en/transaction?id=028d9bdf0b854f4ce467894b02edc152e3407dfee1d97a66ca457264084b3767>

...and this is using the **Cleanup** action

```
search01 ~/testnet : cardano-cli transaction build --babbage-era --testnet-magic 1097911063 --tx-in
06d4bdc4fd0dfc1389b5091e522d874d5dae11d187b2d3f3d99dd877545be37f#1 --tx-in-script-file smart-contracts/swap-
actions.plutus --tx-in-datum-file smart-contracts/unit.json --tx-in-redeemer-file smart-contracts/redeemer-action-
cleanup.json --tx-in-collateral 028d9bdf0b854f4ce467894b02edc152e3407dfee1d97a66ca457264084b3767#0 --tx-out $(cat
wallets/swapactions-payment.addr)+9695970 --change-address $(cat wallets/swapactions-payment.addr) --protocol-params-file
protocol-parameters-babbage.json --out-file tx.body
Estimated transaction fee: Lovelace 304030
search01 ~/testnet : cardano-cli transaction sign --tx-body-file tx.body --signing-key-file wallets/swapactions-
payment.skey --testnet-magic 1097911063 --out-file tx.signed
search01 ~/testnet : cardano-cli transaction submit --tx-file tx.signed --testnet-magic 1097911063
Transaction successfully submitted.
search01 ~/testnet :
```

<https://explorer.cardano-testnet.iohkdev.io/en/transaction?id=8111381667614bca790ef064219ac411f9c8450dbc2407e9b6de5bc148b2168f>

As can be seen, both these transactions are successful (allowed by the contract) and requested funds are transferred from the contract back to the wallet requesting the funds.

Unsupported action (prohibited by the contract)

In the following transaction we try spending the contract funds using an unsupported action. The contract repels this transaction and returns the error message to the user that this action is unsupported.

```
search01 ~/testnet : cardano-cli transaction build --babbage-era --testnet-magic 1097911063 --tx-in
06d4bdc4fd0dfc1389b5091e522d874d5dae11d187b2d3f3d99dd877545be37f#1 --tx-in-script-file smart-contracts/swap-
actions.plutus --tx-in-datum-file smart-contracts/unit.json --tx-in-redeemer-value 42 --tx-in-collateral
028d9bdf0b854f4ce467894b02edc152e3407dfee1d97a66ca457264084b3767#0 --tx-out $(cat wallets/swapactions-payment.addr)
+10000000 --change-address $(cat wallets/swapactions-payment.addr) --protocol-params-file protocol-parameters-
babbage.json --out-file tx.body
Command failed: transaction build Error: The following scripts have execution failures:
the script for transaction input 0 (in the order of the TxIds) failed with:
The Plutus script evaluation failed: An error has occurred: User error:
The machine terminated because of an error, either from a built-in function or from an explicit use of `error`.
Script debugging logs: Unsupported action
```

```
search01 ~/testnet :
```

Next steps...

As stated above, we are currently developing the next contract, NFT.SwapCleanup. This contract will advance our functionality to only allow the cleanup action to be done by one particular wallet, namely the wallet used when creating the contract. It will also return all NFTs sitting in the contract that are not a member of the NFT token policy supported by the contract. We will also enhance the Action from a simple number into an object that can contain more parameters to the contract than just which action to perform.