Exchange Booth Program Spec

Objective

In this project, you will design and implement a smart contract that leverages a price oracle (instance of the Echo program) to facilitate permissionless on-chain token exchanges. The contract will allow users to set up a “token exchange booth” for any pair of SPL tokens by specifying an oracle and spread that determines the exchange rate in either direction and depositing tokens into vaults controlled by the program. Once an exchange booth is initialized, any user can interact with the booth to swap between the pair of tokens based on the current oracle price provided the booth vaults contain enough tokens to take the other side of the trade. Once again, the protocol source code will be written in Rust and sufficient client-side code should be written to allow testing of contract functionality (including updating the oracle using your Echo program). The specification for this project is intentionally less explicitly defined to provide a more realistic representation of the Solana smart contract design process.

State

```
use borsh::{BorshDeserialize, BorshSerialize};

#[derive(BorshSerialize, BorshDeserialize, Debug, Clone)]
pub struct ExchangeBooth {
    ???
}
```

The main data struct the program will utilize is the ExchangeBooth struct which will contain all of the necessary config required for a single token exchange booth instance. At a high level, the struct will need to keep track of the booth “admin” (user who sets up the booth), the two tokens that the booth allows to be exchanged, the “exchange rate” oracle account and spread as well as any other accounts that prove necessary to **securely** implement the contract instructions.

Instructions

These are the instructions that the Exchange Booth Program will support.

```
use borsh::{BorshDeserialize, BorshSerialize};

#[derive(BorshSerialize, BorshDeserialize, Debug, Clone)]
pub enum ExchangeBoothInstruction{
    InitializeExchangeBooth{
        ???
    },
    Deposit{
        ???
    },
    Withdraw{
        ???
    },
    Exchange{
        ???
    },
    CloseExchangeBooth{
        ???
    },
}
```

Instruction 0: InitializeExchangeBooth

Description

This instruction should allow any calling user to initialize an ExchangeBooth account for a pair of SPL tokens. The calling user will henceforth be considered the “admin” for the initialized booth with the expectation that they will subsequently deposit some amount of each token into vault token accounts associated with the ExchangeBooth and controlled by the program.

Considerations

- What other accounts either need to either already exist or be created internally in order for an ExchangeBooth to be legitimate?
- Where are PDAs required (if anywhere)?
- Will the Exchange Booth program have specific requirements for the oracle account?

Instruction 1: Deposit

Description

This instruction should be callable by the booth admin and facilitate a token transfer for one of the booth's two tokens from an admin-controlled token account into a program-controlled vault.

Considerations

- What other programs will need to be called through CPIs?
- *possible gotcha*: How are amounts and balances represented by the token program?
- *note*: This instruction is not technically necessary as any user could in fact transfer tokens into the vault token accounts by directly calling the SPL token program (but why would anyone besides the admin want to?). this instruction acts as more of a convenience for the admin to help ensure the transfer is configured correctly.

Instruction 2: Withdraw

Description

This instruction should be callable by the booth admin and facilitate a token transfer for one of the booth's two tokens from a program-controlled vault to a user-specified token account.

Considerations

- What other programs will need to be called through CPIs?
- What security checks are necessary here?
- *note*: In contrast to the deposit instruction, this instruction is very necessary since the vault token accounts are controlled by the program

Instruction 3: Exchange

Description

This instruction should be callable by anyone and allow the calling user to exchange a specified amount of input token for a calculated amount of output token. The amount of output token should be determined by the current exchange rate as specified by the oracle account as well as the spread parameter for the booth. The instruction should then perform a pair of token transfers to facilitate the exchange (and fail if either side has insufficient funds).

Considerations

- How is the input token specified?
- What security checks are necessary here for all input accounts?
- What format does the oracle account use to represent the exchange rate?
- If necessary, how should floating point math be handled?
- How will the user receive the output tokens?

Instruction 4: CloseExchangeBooth

Description

This instruction should be callable by the booth admin and facilitate closing the booth. The instruction should either explicitly drain the token vaults or assert that they are already empty. It should then clean up the booth account as well as any associated accounts that are safe to delete.

Considerations

- Which accounts are safe to delete?
- Where should the lamports in any deleted accounts end up?

Possible Extensions

- Add protections to the exchange instruction that allow the user to specify a minimum amount of return token and fails the transaction if violated
    - *thought experiment*: Can you think of a way for the calling user to achieve the same protection without any changes to the smart contract?

- Add support for a protocol fee that is collected as a percentage of each exchange
    - What additional token accounts need to get created to allow this?
- What risks are being taken on by the booth admins? What protections can be incorporated into the protocol to reduce those risks?
    - What happens if the oracle has an inaccurate exchange rate?
- How could you redesign this system to allow users to maintain custody of their tokens?
- *ambitious*: Add support for multiple liquidity providers to deposit into the same booth with ownership represented by “pool tokens”
    - How is this implemented for automated market makers (AMMs)?
    - What instructions need to be added and/or modified?