Deposit

This section explains how to deposit funds into a DeFindex vault. You can choose from two different approaches depending on your use case and technical requirements.

Overview

Deposits allow you to add assets to a vault, with the option to automatically invest them into the vault's strategies or keep them as idle funds. All deposit operations require specifying amounts, user addresses, and slippage parameters.

Method 1: Using the API

Best for: Applications that don't need direct smart contract interaction and want language/framework flexibility.

The API approach abstracts away smart contract complexity and handles transaction building for you.

Implementation

```
const vaultAddress =
'CAQ6PAG4X6L7LJVGOKSQ6RU2LADWK4EQXRJGMUWL7SECS7LXUEQLM5U7';

async function deposit(
  amount: number,
  user: string,
  apiClient: ApiClient,
  signerFunction: (tx: string) => string
) {
  // Step 1: Request an unsigned transaction from the API
  const { xdr: unsignedTx } = await apiClient.postData("deposit",
vaultAddress, {
    amounts: [amount],
    from: user
  });

  // Step 2: Sign the transaction (implement your own signer)
  const signedTx = signerFunction(unsignedTx);

  // Step 3: Send the signed transaction back to the API
  const response = await apiClient.postData("send", vaultAddress, {
    xdr: signedTx
  });

  return response;
}
```

API Request Parameters

```
{
  amounts: [10000000],     // Array of amounts for each vault asset (7
decimals for XLM)
  caller: userAddress,     // User's wallet address
  invest: true,           // Auto-invest into strategies (recommended:
true)
  slippageBps: 50        // 0.5% slippage tolerance (optional, default:
0)
}
```

Method 2: Direct Smart Contract Interaction

Best for: dApps that need direct blockchain interaction without backend dependencies, or applications requiring maximum control over contract calls.

Rust Contract Function

```
fn deposit(
    e: Env,
    amounts_desired: Vec<i128>,
    amounts_min: Vec<i128>,
    from: Address,
    invest: bool,
) -> Result<(Vec<i128>, i128,
Option<Vec<Option<AssetInvestmentAllocation>>>), ContractError>
```

Parameters

PROF

- `amounts_desired`: Vector specifying the desired quantities of each asset you wish to deposit
- `amounts_min`: Vector specifying the minimum quantities of each asset to be transferred (slippage protection)
- `from`: Soroban address of the user making the deposit
- `invest`: Boolean indicating whether deposited funds should be automatically invested in vault strategies (`true`) or remain as idle funds (`false`)

Implementation Example

```
use soroban_sdk::{contract, contractimpl, Address, Env, Vec};

#[contract]
pub struct VaultContract;

#[contractimpl]
```

```rust
impl VaultContract {
    pub fn make_deposit(
        env: Env,
        vault_address: Address,
        amounts_desired: Vec<i128>,
        amounts_min: Vec<i128>,
        user_address: Address,
        auto_invest: bool,
    ) -> Result<(Vec<i128>, i128), ContractError> {
        // Call the vault's deposit function
        let client = VaultContractClient::new(&env, &vault_address);

        client.deposit(
            &amounts_desired,
            &amounts_min,
            &user_address,
            &auto_invest,
        )
    }
}
```

Return Values

All deposit methods return information about the completed transaction:

- **Deposited amounts**: The actual amounts deposited for each asset
- **Vault shares minted**: Number of vault shares issued to the depositor
- **Investment allocations** (if `invest = true`): Details of how funds were allocated across strategies