Basic examples

We will explore more use cases of Blaze in this section.

Set up

Let's go over all the things you need to create transactions in Vector.

Code

Make sure you clone Vector Blaze's repository:

```
git clone git@github.com:Apex-Fusion/vector-blaze.git
```

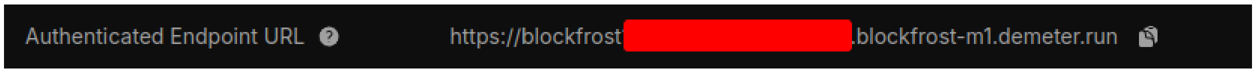
All examples here are inside the file vector-blaze/examples/blockfrost/index.ts

Let's install dependencies:

```
cd vector-blaze/examples/blockfrost
npm i
```

Blockfrost

Make sure you have an instance connected to the `` `vector-testnet` ``. You can use Demeter's Blockfrost RYO. Copy the `` `Authenticated Endpoint URL` ``



Authenticated Endpoint URL https://blockfrost[redacted].blockfrost-m1.demeter.run

and replace `` `http://localhost:3000` `` by your authenticated URL.

```
const provider = new Blockfrost({
  network: "cardano-mainnet",
  projectId,
});

provider.url = "http://localhost:3000"; // CHANGE IT FOR YOUR DEMETEF
```

Wallet with AP3X

We will not go into the details of how you can get a wallet's seed phrase. Click here for a guide.

Once you have your seed phrase. Paste it as the value of the constant `` `mnemonic` ``:

```
const blazeWithSeed = async () ⇒ {
  const mnemonic = ""; // PASTE YOUR SEED PHRASSE HERE

  if (!mnemonic) {
    throw new Error("Missing seed phrase");
```

Now that you've set up Blockfrost and your wallet with AP3X we can go into the examples themselves.

Timelock script (Native Scripts)

You can use Native Scripts to create predicates that if satisfied will allow you to consume a UTxO or mint tokens. Here's an example that locks AP3X in a "timelock" script:

This function will lock 1 AP3X in a timelock script that will be up for consumption after the slot 0. This is obviously always true, but you can change the slot to a future one.

```
const timeLockScript = async () ⇒ {
  console.log("LOCKING APEX IN NATIVE SCRIPT");
  const blaze = await blazeWithSeed();

  const lockScript = Script.newNativeScript(
    NativeScript.newTimelockStart(new TimelockStart(Cardano.Slot(0)))
  );

  const address = addressFromValidator(Core.NetworkId.Testnet, lockSc

  const scriptTx = await blaze
    .newTransaction()
    .lockAssets(address, new Value(1_000_000n), PlutusData.newIntegeı
    .complete();

  const signedTx = await blaze.signTransaction(scriptTx);

  const txId = await blaze.submitTransaction(signedTx);
  console.log(txId);
};
```

Now you can check the transaction on the explorer by copying the transaction id and pasting it on the search bar.

If you want to consume the UTxO you can use this function as well.

```
const consumeTimeLockedScript = async () ⇒ {
  const blaze = await blazeWithSeed();
```

```
    // make sure you are using the same script as in the `timeLockScrip
    const lockScript = Script.newNativeScript(
      NativeScript.newTimelockStart(new TimelockStart(Cardano.Slot(0)))
    );

    const address = addressFromValidator(Core.NetworkId.Testnet, lockSc

    // `input` is the output reference of a locked UTxO in this address
    const [input] = await blaze.provider.getUnspentOutputs(address);
    // `inputOwn` is the output reference of a UTxO in your wallet that
    const [inputOwn] = await blaze.provider.getUnspentOutputs(
      blaze.wallet.address
    );

    const scriptTx = await blaze
      .newTransaction()
      // you must specify which script you want to use
      .provideScript(lockScript)
      .addInput(input)
      .addInput(inputOwn)
      .setValidFrom(Slot(10))
      .payAssets(blaze.wallet.address, new Value(1_000_000n))
      .complete();

    const signedTx = await blaze.signTransaction(scriptTx);

    const txId = await blaze.submitTransaction(signedTx);

    // hash of the transaction that unlocks your APEX
    console.log(txId);
  };
```

Once you submit that transaction you will be able to see in your wallet that you have 1 AP3X available to spend again.

Always true script (Plutus Scripts)

Let's do a similar example that showcases the use of Plutus Scripts. We will create a script that is always true and lock 1 AP3X in it.

```
const lockIntoAlwaysTrue = async () => {
  const blaze = await blazeWithSeed();

  const alwaysTrueScript = Script.newPlutusV2Script(
    // always true plutus v2 script
    new PlutusV2Script(HexBlob("51010000322253330044a229309b2b2b9a1"
  );

  const scriptAddress = addressFromValidator(
    Core.NetworkId.Testnet,
    alwaysTrueScript
  );

  const [inputOwn] = await blaze.provider.getUnspentOutputs(
    blaze.wallet.address
  );

  const tx = await blaze
    .newTransaction()
    .provideScript(alwaysTrueScript)
    .addInput(inputOwn)
    .lockLovelace(scriptAddress, 1_000_000n, PlutusData.newInteger(99
    .complete();

  const signedTx = await blaze.signTransaction(tx);

  const txId = await blaze.submitTransaction(signedTx);

  console.log(txId);
};
```

As you can see that native and plutus scripts are handled similarly by Blaze, the only difference is the script type. You can consume it similarly to the previous example.

```
const consumeAlwaysTrueLocked = async () ⇒ {
  const blaze = await blazeWithSeed();

  const alwaysTrueScript: Script = Script.newPlutusV2Script(
    new Core.PlutusV2Script(
      Core.HexBlob("510100003222253330044a229309b2b2b9a1")
    )
  );

  const scriptAddress = Core.addressFromValidator(
    Core.NetworkId.Testnet,
    alwaysTrueScript
  );

  const [inputOwn] = await blaze.provider.getUnspentOutputs(
    blaze.wallet.address
  );

  const [scriptInput] = await blaze.provider.getUnspentOutputs(script

  const tx = await blaze
    .newTransaction()
    .provideScript(alwaysTrueScript)
    .addInput(scriptInput, PlutusData.newInteger(99n))
    .addInput(inputOwn)
    .complete();

  const signedTx = await blaze.signTransaction(tx);
  const txId = await blaze.submitTransaction(signedTx);

  console.log(txId);
};
```

Token minting (Native scripts vs Plutus scripts)

Now, let's say you want to mint a token with your own name. You can do it with Blaze as well. Here's an example that mints 1000 tokens with the name "FUSION".

```
/**
  * Mints 1000 tokens with the name "FUSION" using a native script
  */
async function mintToken() {
  const blaze = await blazeWithSeed();

  const alwaysTrueScript = Script.newNativeScript(
    NativeScript.newTimelockStart(new TimelockStart(Cardano.Slot(0)))
  );

  const policy = Core.PolicyId(alwaysTrueScript.hash());

  const name = Buffer.from("FUSION").toString("hex");
  const assetName: AssetName = Core.AssetName(name);

  const mint = new Map([[assetName, 1n]]);

  const tx = await blaze
    .newTransaction()
    .addMint(policy, mint)
    .provideScript(alwaysTrueScript)
    .setValidFrom(Slot(10))
    .complete();

  const signedTx = await blaze.signTransaction(tx);

  const txId = await blaze.submitTransaction(signedTx);
  console.log(txId);
}
/**
  * Mints 1000 tokens with the name "FUSION" using a plutus script
  */
const mintTokenV2 = async () => {
  const blaze = await blazeWithSeed();

  const alwaysTrueScript: Script = Script.newPlutusV2Script(
    new Core.PlutusV2Script(
      // always true script here
      Core.HexBlob("510100003222253330044a229309b2b2b9a1")
    )
  );

  const name = Buffer.from("FUSION").toString("hex");
  const assetName: AssetName = Core.AssetName(name);

  const mint = new Map([[assetName, 1000n]]);

  const policy = Core.PolicyId(alwaysTrueScript.hash());

  const tx = await blaze
    .newTransaction()
    .addMint(policy, mint, VOID_PLUTUS_DATA)
    .provideScript(alwaysTrueScript)
    .complete();

  const txId = await blaze.submitTransaction(await blaze.signTransact
  console.log(txId);
};
```

After these transactions you will be able to see the minted tokens in your wallet. Pretty cool, huh?

Now you are ready to build your own applications that interact with Vector. We hope this guide was helpful to you. If you have any questions or need help, feel free to reach out to us. We are always happy to help you.

Privacy policy Terms of service