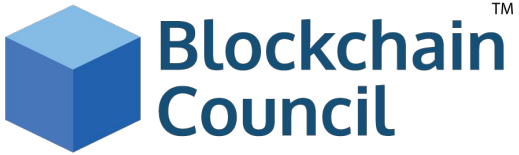

Certified Hyperledger Developer

Transaction Processor Functions

Transaction processor functions

- Scripts file are used to write the transaction processor functions that implement the transactions defined in the Business Network Definition's model files.
- These functions are automatically invoked by the runtime when transactions are submitted using the Business Network Connection API.
- A transaction processor function is the logical operation of a transaction defined in a model file.
- Each transaction type has an associated registry storing the transactions.

Base Structure

- It includes **decorators and metadata** followed by a **JavaScript function**, both parts are required for a transaction processor function to work.

```
/**
* description of a transaction processor function
* @param {org.example.basic.FirstTransaction} parameter-name
* @transaction
*/
function transactionProcessor(parameter-name) {
  //Do some things.
}
```

How to write a processor function?

- The transaction processor function defines the function name declared in the model file of the associated transaction, and also defines the parameter passed to the function.
- It then saves the original value of the asset to be changed by the transaction and replaces it with the value passed in during the submission of the transaction (the newValue property in the transaction definition).
- Then updates the asset in the registry and finally emits an event.

Sample processor function

```
/**
 * Sample transaction processor function.
 * @param {org.example.basic.FirstTransaction} tx The sample transaction instance.
 * @transaction
 */
async function FirstTransaction(tx) {
    let oldValue = tx.asset.value;            // Save the old value of the asset.
    tx.asset.value = tx.newValue;             // Update the asset with the new value.
    let assetRegistry = getAssetRegistry('org.example.basic.SampleAsset');
    await assetRegistry.update(tx.asset);     // Update the asset in the asset registry.
    let event = getFactory().newEvent('org.example.basic', 'SampleEvent');
    event.asset = tx.asset;
    event.oldValue = oldValue;
    event.newValue = tx.newValue;
    emit(event);
}
```

Some basic features

- Transaction processor functions will fail and roll back any changes already made an error is thrown.
- Changes made by transactions are atomic, either the transaction is successful and all changes are applied, or the transaction fails and no changes are applied.
- When assets, transactions, or participants involved in a transaction have a property which includes a relationship, the relationships are resolved automatically.
- All types of relationships, including nested relationships, gets resolved before the transaction processor functions runs.
- Transaction processor functions will wait for promises to be resolved before committing the transaction. If a promise is rejected, the transaction will fail.

How to use APIs in processor functions?

- The Hyperledger Composer and Hyperledger Fabric APIs can be called within transaction processor functions.
- Hyperledger Composer APIs in transaction processor functions can be called simply by calling API functions with the appropriate arguments in the transaction processor function.
  - **Ex.**
    - ```
      async function FirstTransaction(tx) {

      let asset = tx.asset;
      asset.value = tx.newValue;
      let assetRegistry = await
      getAssetRegistry('org.example.basic.SampleAsset');
      await assetRegistry.update(asset);
      }
      ```
      ```

How to use APIs in processor functions?

- To call the Hyperledger Fabric API in a transaction processor function, the function getNativeAPI must be called, followed by a function from the Hyperledger Fabric API.
- Hyperledger Fabric APIs such as **getState, putState, deleteState, getStateByPartialCompositeKey, getQueryResult** gives you access to functionality which is not available in the Hyperledger Composer API.
    - Ex.
        - ```
          const state = await getNativeAPI().getState(nativeState);
          ```
          ```

How to return data?

- Transaction processor functions can optionally return data to client applications.
- The return data for a transaction processor function must be either:
  - a primitive type like String, Integer, Long, etc.,
  - or, a type Composer modelling language type like concept, asset, participant, transaction, event or enumeration.
- The type of the return data must be specified on the model file for the transaction using the **@returns(Type)** decorator, and the return data must be the last thing returned by the transaction processor function.
- Transactions can be modelled as being read-only by specifying the @commit(false) decorator, as this type of transaction will not get committed.

Sample function returning a value

- **Model file:**

```
namespace org.sample

@returns(String)
transaction MyTransaction {
}
```

- **Script file:**

```
async function FirstTransaction(transaction) {
return 'First function!';
}
```

THANK YOU!

Any questions?

You can mail us at

hello@blockchain-council.org