LAB MANUAL - Event & Subscriptions

This exercise is intended to teach, how to use the Sawtooth JavaScript SDK to subscribe to events in sawtooth. The exercise uses a simple “HelloWorld” transaction family with very basic functionality to write a text data to the blockchain state and also read that data from the same network. The exercise has four parts.

1. **Subscribe to the “block-commit” inbuilt Sawtooth event**
2. **Create a custom event for “HelloWorld” transaction family**
3. **Subscribe to the “HelloWorld” event that you created**
4. **Additional exercise to add filters while subscribing to events**

You will be provided with the HelloWorld Client & TP in docker environment.

- You will also be provided with a skeleton code for the event subsystem in the folder “helloworldEvent”.
- Inside the folder, there is a file named EventListener.js
- EventListener.js have the skeleton code for this exercise.
- You need to complete the rest of the code in the file, based on below exercises, for the event system to work.
- Once you complete, you can turn up the docker-compose environment, and the event listening component will also be turned on
- In case of errors/modifications in the EventListener.js code, just restart the event listening container only

Information about Helloworld Transaction Family

FAMILY_NAME =’HelloWorld’ ;
FAMILY_VERSION =’1.0’;

Exercise 1 : Subscribe to the “block-commit” inbuilt Sawtooth event

Tasks to be completed:

1. Import the “Stream” class from sawtooth-sdk
2. Create an object of the Stream class
3. Create a block-commit subscription using ‘**EventSubscription’** protobuf
4. Create a **ClientEventsSubscribeRequest** message with the subscriptions
5. Create stream connection with validator
6. Send **ClientEventsSubscribeRequest** using the stream
7. Decode the **ClientEventsSubscribeResponse** message and check if subscription is OK
8. Write a function to handle event message
   - 8.1. Decode the message using **EventList** protobuf
   - 8.2. Iterate through event list for “block-commit” eventType
9. Attach the handler function to stream using **onReceive** API

Step 1 : Import the Stream class

- The stream class is available in the **sawtooth-sdk/messaging/stream** module

```
const { Stream } = require('sawtooth-sdk/messaging/stream');
```

Step 2 : Create an object of the Stream class

- Give the URL of the validator as the parameter when creating the object
- Write this and following steps, (except step 8) inside the **EventSubscribe** function given in the skeleton code of the exercise

```
let stream = new Stream(URL)
// where URL is the url of the validator
```

Step 3 : Create a block-commit subscription using '**EventSubscription'** protobuf

```
const blockCommitSubscription  = EventSubscription.create({
        eventType: 'sawtooth/block-commit'
    })
```

Step 4 : Create a **ClientEventsSubscribeRequest** message with the subscriptions

```
const subscription_request = ClientEventsSubscribeRequest.encode({
                subscriptions : [blockCommitSubscription]
       }).finish()
```

Step 5 : Create a stream connection

- Once the connection is made, the callback function given as the parameter, would be executed

```
stream.connect(() => {
// This is the callback function
})
```

Step 6 : Send **ClientEventsSubscribeRequest** using the stream

- Write this inside the callback function of the stream connection, so that the subscription request will be sent once stream connection is made.

```
stream.send(Message.MessageType.CLIENT_EVENTS_SUBSCRIBE_REQUEST,subscription_request)
```

Step 7 : Decode the **ClientEventsSubscribeResponse** message and check if subscription is OK

- Decode the response from the validator with the **ClientEventsSubscribeResponse** protobuf.
- Check if the status in the response is OK.
- You can use the "**checkStatus**" function written for you, in the skeleton code given for the exercise

```
stream.send(Message.MessageType.CLIENT_EVENTS_SUBSCRIBE_REQUEST
,subscription_request)
        .then(function (response){
          return ClientEventsSubscribeResponse.decode(response)
        })
        .then(function (decoded_Response){
          console.log(checkStatus(decoded_Response))
        })
```

Step 8 : Write a function to handle event message

Once subscription is done, the validator will send event message through the stream, when a block-commit event occur. So we need to have an function that will handle the events that are sent by the validator (as an EventList)

- A function is already written for you that takes as parameter a message from the stream
- Write your event handling code inside this function

```
function getEventsMessage (message) {
}
```

8.1 Decode the message using **EventList** protobuf

```
function getEventsMessage (message) {

let eventlist =  EventList.decode(message.content).events

}
```

8.2 Iterate through event list for “block-commit” eventType

- In our case, we only have subscribed to ‘block-commit’ event, so our list will only have 1 event type.
- Regardless of how many events we have subscribed, it is safe to check for events using the event type property
- For our purpose, we just want to log the event in the console
- You can use the .map javascript function to iterate through events in the eventlist

```
function getEventsMessage (message) {

let eventlist =  EventList.decode(message.content).events

eventlist.map(function(event){

if(event.eventType == ‘sawtooth/block-commit’) {

console.log(“Event”, event);

}

}

}
```

Step 9 : Attach the handler function to stream using **onReceive** API of Stream class

- `Write this inside the stream connection callback function`

```
stream.onReceive(getEventsMessage)
```

Exercise 2 : Create a custom event for “HelloWorld” transaction family

- This exercise should be done in the Transaction Processor of HelloWorld
- Create an event named `'HelloWorld/WordLength'` in the transaction processor, when writing a text to the state
- Use the ***addEvent*** API of context object to create the event in transaction processor
- The event should have an attribute named `'message_length'`
- The value of the attribute should be the length of the message that is getting written to state
- Add the message that is getting written, in encoded form, as the “data” value of the event.

Tasks to be completed:

1. Create an event named `'HelloWorld/WordLength'` from the transaction processor, with attribute “**message_length**” having the length of the text string as the value, and the text string (in bytes) as the data of the event

```
let msgB = encoder.encode(msg)

attribute = [['message_length',msg_len.toString()]]

context.addEvent('HelloWorld/WordLength',attribute,msgB)
```

Exercise 3 : Subscribe to the "HelloWorld" event that you created

Update the existing code to listen to your custom event

Tasks to be completed:

1. Create a subscription for the event and add to the subscription list
2. Check for the event type in the event handling function

Step 1 : Create a subscription for the event and add to the subscription list

- Just like you created a subscription for block-commit event, create a new subscription for the 'Helloworld/WordLength' event
- Add this new subscription in the ClientEventsSubscribeRequest, along with the block-commit subscription

```
const wordLengthSubscription  = EventSubscription.create({

        eventType: 'HelloWorld/WordLength'

  })

  const subscription_request = ClientEventsSubscribeRequest.encode({

        subscriptions : [blockCommitSubscription, wordLengthSubscription]

  }).finish()
```

Step 2 : Create a subscription for the event and add to the subscription list

- When iterating through the events in the handler function (that we already wrote in the initial exercise), check for your custom event type

```
function getEventsMessage (message) {

let eventlist =  EventList.decode(message.content).events

eventlist.map(function(event){

if(event.eventType == 'sawtooth/block-commit') {

console.log("Event", event);
```

```
}
else if(event.eventType == 'HelloWorld/WordLength') {
console.log("Word length event", event);
}
}
}
```

Exercise 4 : Add filtering to event subscription

We only want the WordLength event, when the length of the message is between 5 and 9.

Modify the event subscription to get the events only when the word length is between 5 and 9

(*Hint* : Refer the events & subscription session slides about event filters. 😃)

(*Hint* : Regular expression for value between 5 and 9 is `^[5-9]\d*$`)

Tasks to be completed:

1. Add a filter to the "HelloWorld/WordLength" event to filter only words with length between 5 and 9
2. Log the words received in the filtered events, in the console. (The words are available in the "data" field of the event)