Publisher-Subscriber scheme for Ethereum Smart Contracts

IMPLEMENTATION PROJECT – HARD

1 PROBLEM STATEMENT

Ethereum based smart contract come with a nifty features to call other smart contract functions or emit messages (logs), here is a guide that sheds light on what this feature. Although this feature has been around for some time and is deemed useful by solidity developers, the scheme awaits a more robust and active system for logging and event handling right out of the box. Your engineering team at Coinbase has been tasked to create a publisher-subscribed system (link1, link2) that works for Ethereum based smart contracts.

2 OUTCOMES

- Support creation/deletion of subscribers in the system.
- Support creation/deletion of publishers in the system.
- A subscriber can only subscribe to "x" event streams at a time.
- A publisher can push upto 100 messages in an event stream at a time.
- A publisher can also be a subscriber and vice-versa. INTERFACES

complete implementation of the following interfaces and interface functions or create appropriate `structs` or `contracts` whenever necessary. The implementation must be a "header" only usage implementation. **Fill in the question marks with appropriate objects, types or code constructs as necessary**.

```
// A class/smart contract that holds the
// the implemetation to create objects that are
// of type "event_stream"
event_stream stream;

function create_event_stream(struct_obj_type obj)
  public view returns (?) {
  // creates an new event stream and returns the id of the stream
}

function delete_event_stream(string event_id)
  public view returns (?) {
  // delete an existing event stream and returns the id of the stream
}
```

Fig. 1. Eventstream Interfaces to implement a PUB-SUB system in Solidity.2 Implementation Project – Hard

```
// A class/smart contract that holds the
// the implemetation to create objects that can
// be of type "publisher" or "subscriber"
publisher p,

function create_publisher(string event_id)
  public view returns (?) {
  // creates an new publisher that can publish to an event stream of id "event_id"
  // and returns the id of the publisher
}

function publish_to_event(publisher p, string event_id)
```

```
    public view returns (?) {
    // make a publisher "p" bind to an event stream of id "event_id"
    // and return acknowledgement
    // The publisher must be able publish to that event stream.
  }

  function remove_publisher(publisher p, string event_id)
    public view returns (?) {
    // make a publisher "p" un-bind from an event stream of id "event_id"
    // and return acknowledgement
    // The publisher must not be able publish to that event stream.
  }

  function delete_publisher(publisher p)
    public view returns (?) {
      // delete a publisher "p".
  }
```

```
// A class/smart contract that holds the
// the implemetation to create objects that can
// be of type "publisher" or "subscriber"
subscriber s;

function create_subscriber(string event_id)
  public view returns (?) {
  // creates an new subscriber that can publish to an event stream of id "event_id"
  // and returns the id of the publisher
}

function subscribe_to_event(subscriber s, string event_id)
  public view returns (?) {
  // make a subscriber "s" subscribe to an event stream of id "event_id"
  // and return acknowledgement
}

function unsubscribe_to_event(subscriber s, string event_id)
  public view returns (?) {
  // make a subscriber "s" unsubscribe to an existing
  // subscribed event stream of id "event_id"
  // and return acknowledgement
}

function delete_subscriber(subscriber s)
  public view returns (?) {
    // delete a subscriber "s".
}
```

Fig. 3. Subscriber Interfaces to implement a PUB-SUB system in Solidity.