GO Client Library

The SELIS Publish/Subscribe GO Library allows a GO client to interact with the SELIS Publish/Subscribe, by establishing and maintaining the TLS connection to the the SELIS Publish/Subscribe. It allows GO client to publish and receive messages by subscribing for Messages matching predefined criteria.

The SELIS Publish/Subscribe GO Library simplifies the interaction with the SELIS Publish/Subscribe system by hiding the client from the technical details of the SELIS Publish/Subscribe protocol.

Dependencies

The GO Client Library is dependent on the GO package 'gopkg.in/resty.v1', which provides the functionality of calling the REST services and processing the results and errors in user-friendly manner.

Download

The SELIS Publish/Subscribe GO library it is not publicly accessible for downloading. It is neither published in any public repository.
The sources of the library are available in the SELIS repository containing all other Pub/Sub client libraries.

API Guide

The SELIS Publish/Subscribe GO Library is implemented as a GO package ‘pubsub’ within the package path tu-dresden.de/selis/go-connector.

There are few important structures, which are used in the process of establishing connection, publishing and subscribing for messages in the SELIS Publish/Subscribe:

- PubSub - defines the connection of the SELIS Publish/Subscribe and provides the high level API for publishing and subscribing
- Subscription - defines all subscription’s parameters required to subscribe for specific kind of messages.
- Rule - defines the filtering criteria within the subscription

Connecting to the SELIS Publish/Subscribe system

To connect to the SELIS Publish/Subscribe system, the host and the port of the SELIS Publish/Subscribe API has to be defined. The default port is 20000.

```
pubsub := NewPubSub("path/to/rootCA.crt", "0.0.0.0", 20000)
```

The ‘NewPubSub’ method returns the pointer to the PubSub struct, which provides the high level API for publishing and subscribing of the messages.

Creating new subscription

To create the subscription, the user must provide authorisation token and the subscription id. The Authorisation token has to be acquired from the Single Sign-On (SSO) system. The token is used to verify that the user is authenticated and if it has enough permissions to subscribe for this specific subscription.

```
subscription := NewSubscription("clientIdHash2", "subId3")
```

Once the subscription object is created, the filtering rules have to be defined. Filtering rules allows to define criteria based on which the SELIS Publish/Subscribe system will decide which messages should be forwarded to the subscriber and which not. Each filtering rule has to specify the 3 parameters:

- key - the key which has to be present in the published message, and which value will be used for comparison
- value - the value of the defined key, which will be used for the comparison
- comparator - defines how the value from the published message will be compared to the value defined in the filtering rule. The possible values of the comparator are: EQ, NE, LT, LE, GT, GE and represents the following operations: equals, not equals, lower than, lower equals, greater than, greater equals.

In the following example, the subscription will expect the messages, which key is equal to the “_type” string and the value is equals (EQ) to the “PKI” string:

```
subscription.addRule(StringRule("_type", "PKI", EQ))
```

The library provides the functions to create filtering rules for different types of values:

- float: FloatRule(),
- boolean: BooleanRule(),
- int: IntRule().

It is possible to add more filtering rules. In such case, the SELIS Publish/Subscribe system will look for the messages which match all of the filtering rules.

When the SELIS Publish/Subscribe system receives the message from the publisher, which matches all filtering rules, it forwards it to the subscriber. The SELIS Publish/Subscribe GO client library allows to register a callback function, which will be invoked each time the SELIS Publish/Subscribe system forwards the message. The callback function has to implement the signature: func(message string, err error)

In the following example, the callback simply prints the value of the received message to the standard output.

```
callback := func(message string, err error) {
    if err != nil {
        // handle error
        return
```

```
    }
    fmt.Println("Got new message: ", message)
}
```

Finally, when the subscription object and the callback are defined, they can be used to register a new subscription. The subscribe() method from the PubSub struct has to be used.

```
err := pubsub.subscribe(subscription, callback)
if err != nil {
    // handle error
}
```

In order to unsubscribe, the unsubscribe() method from the PubSub struct has to be invoked.

```
pubsub.unsubscribe(subscription)
```

Publishing new messages

The messages exchanged within the SELIS Publish/Subscribe system contains the list of key:value pairs. In GO it can be represented by map[string][string] as presented in the following example:

```
msg := map[string]string {
    "_type": "PKI",
```

```
    "_subtype": "avg_price",
    "supplier_id": "ABC2138",
    "value": "1908",
}
```

To publish the message, the publish() method from the PubSub struct has to be invoked. In the exceptional situations, such as connection problem or unavailability of the SELIS Publish/Subscribe system, the error is returned. It is the responsibility of the developer to properly handle it, the library will not try to automatically re-send the message.

```
err := pubsub.publish(msg)
if err != nil {
    // handle error
}
```

Error handling

The SELIS Publish/Subscribe GO client library does not force the developer to handle the errors. It simplifies fast prototyping and testing the interaction with the SELIS Publish/Subscribe system, but it is not recommended for production code. Therefore, the developer should also check for the error type returned from the Publish()/Subscribe() methods and handle them accordingly.