Package state

```
import "github.com/emilyhorsman/4zp6/backend/controller/state"
```

Overview ▾

Index ▾

Package files

amqp.go config.go mqtt.go state.go

func **triggerConnect**

```
func triggerConnect(m *MQTT, client mqtt.Client)
```

triggerConnect is called on MQTT connection. It updates the internal connection status and triggers this client to notify the broker of all topics.

func **triggerDisconnect**

```
func triggerDisconnect(m *MQTT)
```

triggerDisconnect is called on MQTT disconnect.

func **triggerMessage**

```
func triggerMessage(m *MQTT, raw mqtt.Message)
```

triggerMessage is called on incoming MQTT message. It will emit an MQTTMessage on the out channel.

type **AMQP**

AMQP is the AMQP client.

```
type AMQP struct {
    routing     map[string][]string // routing is a map binding queues to
a list of routing keys
    isConnected bool                // isConnected indicate if AMQP
connection is currently established
    channel     *amqp.Channel       // channel is the internal AMQP
channel
    id          string              // id is the client identifier
    host        string              // host is the formatted AMQP host
string
    exchange    string              // exchange is common AMQP exchange
    ch          chan AMQPMessage    // ch is the message output channel,
exposed by Init()
    log         *log.Logger         // log is state structured event
logger
}
```

func (*AMQP) Close

```
func (a *AMQP) Close() error
```

Close will close the channel to the AMQP broker. It will return an error if the channel fails to gracefully close.

func (*AMQP) GetRouting

```
func (a *AMQP) GetRouting() map[string][]string
```

GetRouting returns the list queues and the list of routing queues binded to each queue.

func (*AMQP) Init

```
func (a *AMQP) Init(log *log.Logger, id, host string) (chan AMQPMessage,
error)
```

Init will attempt to initialize the AMQP client. It requires the AMQP broker's ID string and formatted AMQP host string. It will return the output channel or an error.

func (*AMQP) Publish

```
func (a *AMQP) Publish(routingKey string, payload []byte) error
```

Publish accept a routing key and a payload. It will attempt to publish the payload with the provided routing key. It will return an error if the payload fails to publish.

func (*AMQP) UpdateRouting

```
func (a *AMQP) UpdateRouting(routing map[string][]string) error
```

UpdateRouting replace the AMQP routing with the one provided. It will notify the AMQP broker of the changes. It may return an error if the transaction fails to complete.

func (*AMQP) init

```
func (a *AMQP) init() error
```

init is the internal initalization function. It is called by Init() and recover(). It establishes the connection to the AMQP broker without redeclaring the output channel. It will return an error if the connection with the AMQP broker fails to establish.

func (*AMQP) notifyRouting

```
func (a *AMQP) notifyRouting() error
```

notifyRouting will notify the AMQP broker of the routing key queue binding. It will return an error if the routing keys fail to bind.

func (*AMQP) recover

```
func (a *AMQP) recover(conn *amqp.Connection)
```

recover is called when the AMQP loses its connection. It will recursively call init() until the connection is re-established.

type **AMQPMessage**

AMQPMessage is a message sent over AMQP.

```
type AMQPMessage struct {
    RoutingKey string
    Payload    []byte
}
```

type **Config**

Config is the environment variable configuration for the controller.

```
type Config struct {
    AMQPUser string // AMQPUser is AMQP username
    AMQPPass string // AMQPPass is AMQP password
    MQTTUser string // MQTTUser is MQTT username
    MQTTPass string // MQTTPass is MQTT password
    SQLUser  string // SQLUser is SQL username
    SQLPass  string // SQLPass is SQL password
}
```

func (*Config) load

```
func (c *Config) load() error
```

load will attempt to load the required environment variables into the Config struct. An error will be returned if a required variable is not defined.

type MQTT

MQTT is the MQTT client.

```
type MQTT struct {
    topics      []string            // Topics to subscribe
    isConnected bool                // isConnected indicate if MQTT
connection is currently established
    options     *mqtt.ClientOptions // options is internal client options
    client      mqtt.Client         // client is internal client
    ch          chan MQTTMessage    // ch is the message output channel,
exposed by Init()
    log         *log.Logger         // log is state structured event
logger
}
```

func (*MQTT) Close

```
func (m *MQTT) Close()
```

Close will close the connection to the MQTT broker.

func (*MQTT) Connect

```
func (m *MQTT) Connect() error
```

Connect will attempt to establish a connection with the MQTT broker. It will return an error if the connection fails to establish.

func (*MQTT) GetTopics

```
func (m *MQTT) GetTopics() []string
```

GetTopics returns the list of currently subscribed topics.

func (*MQTT) Init

```
func (m *MQTT) Init(log *log.Logger, host, user, pass string) (chan
MQTTMessage, error)
```

Init will attempt to initialize the MQTT client. It requires the MQTT broker's host, username, and password. It will return the output channel or an error.

func (*MQTT) Publish

```
func (m *MQTT) Publish(topic string, payload []byte) error
```

Publish will publish a payload on the provided topic. It will return an error if the payload fails to publish.

func (*MQTT) UpdateTopics

```
func (m *MQTT) UpdateTopics(topics []string) error
```

UpdateTopics repleaces the subscribe topics list with the one provided. It will notify the MQTT broker of the changes. It may return an error if the transaction fails to complete.

func (*MQTT) notifyTopics

```
func (m *MQTT) notifyTopics() error
```

notifyTopics will notify the MQTT broker of the topics to subscribe to. It will return an error if the topics fail to bind.

type **MQTTMessage**

MQTTMessage is a message sent over MQTT.

```
type MQTTMessage struct {
    Topic   string // Topic is the topic the message was published on
    Payload []byte // Payload is the message payload
}
```

type State

State is the global state for the controller.

```
type State struct {
    Log      *log.Logger         // Log is a structured event logger
    IsDocker bool                // IsDocker indicates if running inside
Docker
    Config   *Config             // Config contains global configuration
    MQTT     *MQTT               // MQTT is the MQTT client
    MQTTCh   chan MQTTMessage    // MQTTCh is the MQTT output channel
    AMQP     [2]*AMQP            // AMQP is the AMQP client (two for
duplex)
    AMQPCh   [2]chan AMQPMessage // AMQPCh is the AMQP output channel
    SQL      *sql.DB             // SQL is the SQL client
    Data     chan []byte         // Data channel is populated by AMQP and
consumed by websockets
}
```

func Provision

```
func Provision() (*State, error)
```

Provision generates a global state for the controller. It will return an error if the state fails to initialize.

func (*State) amqp

```
func (s *State) amqp() error
```

amqp attempts to populate the AMQP and AMQPCh fields in State.

func (*State) config

```
func (s *State) config() error
```

config attempts to populate the Config field in State.

func (*State) mqtt

```
func (s *State) mqtt() error
```

mqtt attempts to populate the MQTT and MQTTCh fields in State.

func (*State) sql

```
func (s *State) sql() error
```

sql attempts to populate the SQL field in State.

type WebsocketFrame

WebsocketFrame is a frame emitted on the websocket connection.

```
type WebsocketFrame struct {
    UUID      string      `json:"uuid"`      // UUID is the UUID of the microcontroller
    BusAddr   int         `json:"busAddr"`   // BusAddr is the bus address where data was collected
    Timestamp string      `json:"timestamp"` // Timestamp is time which data was received by peripheral controller
    Data      interface{} `json:"data"`      // Data is the JSON payload from peripheral controller
}
```