Package db

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


Overview

Index

Package files

config.go data.go init.go provisioning.go registration.go

Constants

```
const (
    upsertProcessor = `INSERT INTO Processor(busAddr,name)
    VALUES ($1,$2)
    ON CONFLICT (busAddr) DO UPDATE
    SET busAddr=EXCLUDED.busAddr,
        name=EXCLUDED.name;`

    upsertReadDefinition = `INSERT INTO
ReadDefinition(definitionId,regIdLength,regId,regBlockLength,bytesPerReg,re
```

```
    VALUES ($1,$2,$3,$4,$5,$6)
    ON CONFLICT (definitionId) DO UPDATE
    SET definitionId=EXCLUDED.definitionId,
        regIdLength=EXCLUDED.regIdLength,
        regId=EXCLUDED.regId,
        regBlockLength=EXCLUDED.regBlockLength,
        bytesPerReg=EXCLUDED.bytesPerReg,
        readPeriod=EXCLUDED.readPeriod;`

    upsertProvisioning = `INSERT INTO Provisioning(busAddr,definitionId)
    VALUES ($1,$2)
    ON CONFLICT (busAddr,definitionId) DO UPDATE
    SET busAddr=EXCLUDED.busAddr,
        definitionId=EXCLUDED.definitionId;`
)
```

```
const (
    insertData = `INSERT INTO Data(pollUuid,uuid,busAddr,time,data)
    VALUES ($1,$2,$3,$4,$5);`

    selectAllData = `SELECT uuid,busAddr,time,data
    FROM data
    WHERE uuid=$1 AND busAddr=$2;`

    selectStartData = `SELECT uuid,busAddr,time,data
    FROM data
    WHERE uuid=$1 AND busAddr=$2 AND time>=$3;`

    selectStopData = `SELECT uuid,busAddr,time,data
    FROM data
    WHERE uuid=$1 AND busAddr=$2 AND time<=$3;`

    selectStartStopData = `SELECT uuid,busAddr,time,data
    FROM data
    WHERE uuid=$1 AND busAddr=$2 AND time between $3 and $4;`
)
```

```
const (
    registrationSchema = `CREATE TABLE IF NOT EXISTS Registration(
        uuid            text primary key        not null,
        firmware        int                             not null,
        ipv4            text                            not null,
        ipv6            text                            not null
    );`
```

```
    peripheralSchema = `CREATE TABLE IF NOT EXISTS Peripheral(
        controller      text    not null REFERENCES Registration(uuid) ON
DELETE CASCADE,
        busID           int     not null,
        busAddr         int     not null,
        callResp        bytea,
        PRIMARY KEY(controller, busID, busAddr)
    );`

    processorSchema = `CREATE TABLE IF NOT EXISTS Processor(
        busAddr int primary key not null,
        name    text                    not null
    );`

    readDefinitionSchema = `CREATE TABLE IF NOT EXISTS ReadDefinition(
        definitionId    int primary key not null,
        regIdLength     int                     not null,
        regId                   int                     not null,
        regBlockLength  int                     not null,
        bytesPerReg     int                     not null,
        readPeriod              int                     not null
    );`

    provisioningSchema = `CREATE TABLE IF NOT EXISTS Provisioning(
        busAddr                 int not null REFERENCES
Processor(busAddr) ON DELETE CASCADE,
        definitionId    int not null REFERENCES
ReadDefinition(definitionId) ON DELETE CASCADE,
        PRIMARY KEY(busAddr, definitionId)
    );`

    dataSchema = `CREATE TABLE IF NOT EXISTS Data(
        pollUuid        text            not null,
        uuid            text            not null REFERENCES
Registration(uuid) ON DELETE CASCADE,
        busAddr         int             not null REFERENCES
Processor(busAddr) ON DELETE CASCADE,
        time            timestamptz not null,
        data            jsonb           not null,
        PRIMARY KEY(pollUuid)
    );`
)
```

```
const (
    selectProvisioning = `SELECT
busAddr,name,definitionId,regIdLength,regId,regBlockLength,bytesPerReg,read
```

```
    FROM Provisioning
    LEFT JOIN Processor using(busAddr)
    LEFT JOIN ReadDefinition using(definitionId)
    WHERE provisioning.busAddr = $1;`

    selectAllProvisioning = `SELECT
busAddr,name,definitionId,regIdLength,regId,regBlockLength,bytesPerReg,read

    FROM Provisioning
    LEFT JOIN Processor using(busAddr)
    LEFT JOIN ReadDefinition using(definitionId);`
)
```

```
const (
    upsertRegistration = `INSERT INTO
Registration(uuid,firmware,ipv4,ipv6)
    VALUES ($1,$2,$3,$4)
    ON CONFLICT (uuid) DO UPDATE
    SET uuid=EXCLUDED.uuid,
        firmware=EXCLUDED.firmware,
        ipv4=EXCLUDED.ipv4,
        ipv6=EXCLUDED.ipv6;`

    deletePeripheral = `DELETE FROM Peripheral
    WHERE controller = $1;`

    insertPeripheral = `INSERT INTO
Peripheral(controller,busId,busAddr,callResp)
    VALUES ($1,$2,$3,$4);`

    selectRegistration = `SELECT uuid from Registration;`

    selectAllRegistration = `SELECT uuid,firmware,ipv4,ipv6 from
Registration;`

    selectFilterPeripheral = `SELECT busId,busAddr,callResp
    FROM Peripheral
    WHERE controller = $1;`
)
```

func **GetRegistered**

```
func GetRegistered(s *state.State) (map[string]bool, error)
```

GetRegistered returns the map of microcontrollers UUID that are in the registration table.

func **IndexData**

```
func IndexData(s *state.State, uuid string, busAddr int, ts time.Time,
payload []byte) error
```

IndexData is called when the peripheral controller publishes processed data (payload). It will attempt to index the data in the "Data" table.

func **Init**

```
func Init(s *state.State) error
```

Init initializes the Postgres database. If the tables do not already exist, the tables will be created. An error will be returned if the database cannot be queried or if the tables cannot be created.

func **QueryData**

```
func QueryData(s *state.State, uuid string, busAddr int, start
*time.Time, stop *time.Time) ([]state.WebsocketFrame, error)
```

QueryData accepts a microcontroller UUID, bus address, and option start and stop times. It will return all data points matching the parameters. An error will be returned if the query cannot be completed.

func **QueryProvisioning**

```
func QueryProvisioning(s *state.State, busAddr uint32)
(*telemetry.Provisioning, bool, error)
```

QueryProvisioning will query the "Processor", "ReadDefintion" and "Provisioning" tables using the provided bus address, returning a telemetry Protobuf instance and an indicator if

a provisioning record was found. An error will be returned if the query cannot be completed.

func **UpsertConfig**

```
func UpsertConfig(s *state.State, config JSONConfig) error
```

UpsertConfig accepts the JSON configuration from the peripheral controller. will update the "Processor", "ReadDefinition" and "Provisioning" tables with the provided data. It will return an error if the tables cannot be updated.

func **UpsertRegistration**

```
func UpsertRegistration(s *state.State, msg *telemetry.Telemetry) error
```

UpsertRegistration will insert the registration telemetry message into the "registration" and "peripheral" tables.

type **JSONConfig**

JSONConfig is the configuration from the peripheral controller.

```
type JSONConfig struct {
    BusAddr         int              `json:"busAddr"`
    Name            string           `json:"name"`
    ReadDefinitions []ReadDefinition `json:"readDefinitions"`
}
```

type **Peripheral**

Peripheral is a peripheral connected to the microcontroller.

```
type Peripheral struct {
    BusID    int    `json:"busId"`    // BusID is the bus identifier of
the peripheral
    BusAddr  int    `json:"busAddr"`  //BusAddr is the bus address of the
peripehral
    CallResp []byte `json:"callResp"` // CallResp is the general call
```

```
response
}
```

type **Provisioning**

Provisioning represnts a joined entry from "Provisioning", "Processor", and "ReadDefinition" tables.

```
type Provisioning struct {
    BusAddr        int    `json:"busAddr"`        // BusAddr is the bus
address of the peripehral
    Name           string `json:"name"`           // Name is common name
of peripheral controller
    DefinitionID   int    `json:"definitionId"`   // DefinitionID is a
definition UUID
    RegIDLength    int    `json:"regIdLength"`    // RegIDLength is
register ID length
    RegID          int    `json:"regId"`          // RegID is register ID
    RegBlockLength int    `json:"regBlockLength"` // RegBlockLenght is
register block length
    BytesPerReg    int    `json:"bytesPerReg"`    // BytesPerReg is bytes
per register
    ReadPeriod     int    `json:"readPeriod"`     // ReadPeriod is the
bus address reading period
}
```

func GetProvisioning

```
func GetProvisioning(s *state.State) ([]Provisioning, error)
```

GetProvisioning returns a list of active provisionings. An erorr will be returned if the query cannot be completed.

type **ReadDefinition**

ReadDefinition is a definition in the controller.

```
type ReadDefinition struct {
    DefinitionID        int `json:"definitionId"`
    RegisterIDLength    int `json:"registerIdLength"`
    RegisterID          int `json:"registerId"`
```

```
    RegisterBlockLength int `json:"registerBlockLength"`
    NumBytesPerRegister int `json:"numBytesPerRegister"`
    ReadPeriod          int `json:"readPeriod"`
}
```

type **RegistrationPeripherals**

RegistrationPeripherals is the registration information combined with connected Peripherals.

```
type RegistrationPeripherals struct {
    UUID        string       `json:"uuid"`        // UUID is the
microcontroller unique identifier
    Firmware    int          `json:"firmware"`    // Firmware is the
microcontroller firmware number
    IPv4        string       `json:"ipv4"`        // IPv4 is the IPv4
address
    IPv6        string       `json:"ipv6"`        // IPv6 is the IPv6
address
    Peripherals []Peripheral `json:"peripherals"` // Peripheral is the
list of connected peripherals
}
```

func GetRegistrationPeripherals

```
func GetRegistrationPeripherals(s *state.State)
([]RegistrationPeripherals, error)
```

GetRegistrationPeripherals will join the "Register" and "Peripheral" tables, returning the complete registration information. It will return an error if the

---