Deutsches Zentrum für Luft- und Raumfahrt
Experimental Gravitational Physics and Geodesy

Fichero de configuración JSON.

Elementos y valores al detalle

Autor:
Pablo Torres Anaya

23 de junio de 2017
Powered by LaTeX

Índice

Resumen

–todo general– Usar notación estándar de construcción para definir los elementos. –todo– En esta sección explicar a fondo cual es la estructura del fichero de configuración y como funciona

El script de python se configura mediante un único fichero JSON donde especificamos todos sus parámetros de funcionamiento usando una estructura concreta.

Primero vamos a ver un ejemplo resumido del fichero con algunas indicaciones y a continuación se detallan todos los posibles valores y configuraciones del fichero.

Ejemplo con comentarios

Este ejemplo contiene algunos comentarios cortos sobre cada elemento. No se puede usar directamente ya que no es un JSON válido por tener los comentarios.

```
{
  "control": { <- Main Control configuration
    "address": "localhost", <- MQTT Broker addres
    "port": 1883, <- MQTT Broker port
    "topic": "master/control", <- MQTT control topic
    "qos": 0, <- MQTT control topic qos
    "control_messages": { <- control messages
      "reload": "msg_reload",
      "shutdown": "msg_shutdown",
      "status": "msg_status"
    }
  },
  "dbs": {<- [OPTIONAL] Dictionary list of Databases
    "influxdb0":{ <- Id of the new database
      "class_name": "Influx", <- Type of database System (Only
         ↪ Influx implemented for now)
      "addres": "localhost", <- Addres of database System
      "port": "default", <- Port of database System
      "db_name": "payload_logger", <- Name of the desired
         ↪ Database
      "user": "admin",
      "password": "admin"
    },
    "influxdb1": {...}, <- Aditional databases
    "mysqldb0": {...} <- mysql not yet implemented
    },
  "joiners": { <- [OPTIONAL] List of joining functions
      "join_1":{<- Id of the join
      "synchronous": 1, <- If the join synchronice the data
      "function": "({0}+{1})/2", <- Function to implement
      "topics": { <- Topics structure explained later
        "topic_1": {},
        "topic2": {},
        "anothertopic":{},
        "one more":{}
```

```
    }
  },
  "join_2":{...},
  "another_join":{}
},
"sockets": { <- List of all sockets connections
  "id_filesocket_interferometer_1" : { <- ID of the socket
    "class_name": "FileSocket", <- Tipe of the socket
    "control":[ <- Control sistem of the socket
      {
        "topic" : "interferometer/control/power", <- Topic to
           ↪ receive commands
        "qos" : 1, <- qos to subscribe the topic
        "actions":[ <- List of actions
          {
            "message":"stop", <- if we recive this msg
            "action":0 <- we perform this action
          },
          {
            "message":"start",
            "action":1
          }
        ]
      }
    ],
    "publish": { <- How to deal with the data
      "parser": "#{0}:{1};",<- specify the format of the data
      "join": [ <- If the data goes to a join function (list)
        {
          "from": 0, <- position of the data in the parser
          "to": "join_1", <- id of the join
          "position": 0, <- position inside the join function
          "sync": 1 <- position of the sync mark the in parser
        },
        {...},
        {...}
      ],
      "topics": {<- list of actions for data
        "0": { <- list of actions for data 0
          "raw":{ <- id of the action
            "function": "Raw", <- function of the action
```

```
            "samples": 1, <- number of samples to perform the
               ↪ action
            "topic": "interferometer/value", <- where do we
               ↪ publish it,
            "to_db":{ <- if we whant to save on Database
              "table":"interferometer/raw", <- Table
              "column":"payload", <- Column
              "db_id":"db0" <- Id of the Database
            }
          },
          "variance"{...} <- Another action
                },
        "1": { <- list of actions for data 1
          "raw":{
            "function": "Raw",
            "samples": 1,
            "topic": "interferometer/sync"
          },
          "variance"{...}
        },

      },
      "file_name": "filename.txt", <- Aditional Parameter for
         ↪  FileSocket
      "delay": 0.001 <- Specify a delay to the read function
         ↪ (for soft-sync)
    }
  },
  "id_serialsocket_interferometer2": {
    "class_name": "SerialSocket",
    "eol_character":";", <- Aditional Parameter for
       ↪ SerialSocket
    "port": "/dev/ttyACM0",<- Aditional Parameter for
       ↪ SerialSocket
    "baudrate":9600,<- Aditional Parameter for SerialSocket
    "control":[{ ... },{ ... } ... ],
    "publish": {
      "parser": "#{0}:{1};",
      "join": [ {...}, ... ],
      "topics": { ... },
    }
```

```
    "delay": 0.001
  }
 }
}
```

El fichero tiene cuatro elementos principales

- control - Obligatorio
- dbs - Opcional
- joins - Opcional
- sockets - Opcional

1. Control

En esta sección configuramos la conexión al servidor de MQTT y cual va a ser el topic para controlar el programa principal y sus mensajes. Este elemento es obligatorio en el fichero JSON.

Los elementos que tiene son:

- "address" - MQTT Broker addres
- "port" - MQTT Broker port
- "topic" - MQTT control topic
- "qos" - MQTT control topic qos [Default: 0]
- "control_messages" - control messages list

1.1. Address

–todo–

1.2. Port

–todo–

1.3. Topic

–todo–

1.4. Qos

–todo–

1.5. Control Messages

"control_messages" es un diccionario donde configuramos cuales van a ser los mensajes que esperamos recibir en el topic de control para realizar las tres acciones de control que soporta el sistema actualmente.

Elementos que debe tener "control_messages":

- "reload" - string del mensaje que queremos usar para recargar la configuración
- "shutdown" - string del mensaje que queremos usar para apagar el sistema
- "status" - string del mensaje que queremos usar para consultar el estado del sistema

1.6. Ejemplo

```
"control": {
  "address": "localhost",
  "port": 1883,
  "topic": "master/control",
```

```
  "qos": 0,
  "control_messages": {
    "reload": "msg_reload",
    "shutdown": "msg_shutdown",
    "status": "msg_status"
  }
},
```

2. DBs

dbs es una lista de bases de datos donde cada entrada corresponde a una base de datos. No es obligatorio poner dbs en el fichero de configuración.

Cada entrada consta de un ID y un diccionario de configuración. El ID es arbitrario y sirve para identificar esa conexión en otros puntos del fichero.

Cada base de datos consta de los siguientes elementos:

- "class_name" - Tipo de Database. Valores: ["Influx"]
- "addres" - Dirección del servidor
- "port" - Puerto de conexión
- "db_name" - Nombre de la base de datos
- "user" - Usuario
- "password" - Contraseña

2.1. Class Name

– todo –

2.2. Address

– todo –

2.3. Port

– todo –

2.4. Db Name

– todo –

2.5. User

– todo –

2.6. Password

– todo –

2.7. Ejemplo

```
"dbs":{
  "db0":{
    "class_name": "Influx",
    "addres": "localhost",
    "port": "default",
    "db_name": "payload_logger",
    "user": "admin",
    "password": "admin"
  },
  "db1":{...},
  "anotherdb":{...}
}
```

3. Topic

Un topic es la unidad de output del script. Determina como "sale" un dato del sistema. Y consta de los siguientes elementos:

- "function" - La función de salida que queremos aplicar al dato. Valores ["Raw","Variance","Mean","StandardDeviation"]
- "samples" - Cada cuantas muestras queremos aplicar la función y hacer la salida del dato. [Default: 100]
- "topic" - Si está presente indica en que topic de MQTT queremos publicar los datos
- "retain" - Si se tiene que retener el dato en MQTT [Default: 0]
- "qos" - El qos de la publicación [Default: 0]
- "to_db" - Si está presente indica que se tiene que guardar en una base de datos. Es un diccionario con los siguientes elementos:
  - "table" - Tabla que debe almacenar este dato
  - "column" - Columna a la que hay que asignar este dato
  - "db_id" - Identificador de la base de datos donde queremos guardarlo.

3.1. Function

Valores del campo "function":

- "Raw" - Muestra el ultimo dato recibido ignorando todos los anteriores. Si se configura con "samples":1 es equivalente a mostrar todos los datos.
- "Mean" - Obtiene la media de todos los datos.
- "Variance" - Obtiene la varianza de todos los datos.
- "StandardDeviation" - Obtiene la desviación estándar de todos los datos

3.2. Samples

– todo –

3.3. Topic

– todo –

3.4. Retain

– todo –

3.5. Qos

– todo –

3.6. To Db

– todo –

3.6.1. Table

– todo –

3.6.2. Column

– todo –

3.6.3. Db Id

– todo –

3.7. Ejemplo

Ejemplo que guarda todos los datos en una base de datos y los publica por mqtt

```
"topc_id":{
  "function": "Raw",
  "samples": 1,
  "topic": "interferometer/raw",
  "retain":1,
  "qos":1,
  "to_db":{
    "table":"interferometer/raw",
    "column":"payload",
    "db_id":"db0"
  }
}
```

Ejemplo calcula la media cada 50 datos y solo los publica en mqtt

```
"topc_id":{
  "function": "Mean",
  "samples": 50,
  "topic": "interferometer/mean"

}
```

Ejemplo calcula la media cada 200 datos y solo los guarda en la base de datos

```
"topc_id":{
  "function": "Mean",
  "samples": 200,
  "to_db":{
    "table":"interferometer/mean",
    "column":"payload",
```

```
    "db_id":"db0"
  }
}
```

4. Joins

Un joiner es un elemento de unión entre diferentes datos. Sirve para poder procesar datos de diferente fuente, al contrario que los topics que son procesamientos del mismo dato.

La lista de joiner tiene que empezar la key "joiners" y ser un diccionario de joiners donde un Joiner se define mediante un ID y un diccionario con los siguientes elementos:

- "synchronous" - valor 1 si los datos tienen que sincronizarse usando una marca de tiempo 0 si se sincronizan según el orden de llegada [Default: 0]
- "function" - String donde especificamos la función exacta que queremos ejecutar. El lugar que ocupan los datos se marcan con números encapsulados en corchetes. por ejemplo: '0'. El número hace referencia a la posición en la formula. Dicha referencia se usará para indicar a donde tiene que ir cada dato.
- "topics" - Diccionario con una lista de topics.

4.1. Synchronous

– todo –

4.2. Function

– todo –

4.3. Topics

– todo –

4.4. Ejemplo

```
"joiners": {
  "join_1":{
    "synchronous": 1,+
    "function": "({0}+{1})/2",
    "topics": {
      "topic_1": {<topic_data>},
      "topic2": {<topic_data>},
      "anothertopic":{<topic_data>},
      "onemore":{<topic_data>}
    }
  },
  "join_2":{...},
  "another_join":{}
},
```

5. Sockets

Un socket es la unidad de conexión (o entrada) básica (y única) del sistema. Con ella podemos definir de donde leemos datos y como los leemos.

Al igual que los joiners tiene que ser un diccionario (lista) con los ID's y sus valores. Para definir la lista de sockets tenemos que usar la key: "sockets" seguido del diccionario con los sockets individuales.

Vamos a ver los elementos de un Socket:

- "class_name" - Nombre del tipo de socket.
- "control" - Diccionario para definir cuales son los parámetros de control del socket.

- "publish" - Lista de topics.
- "delay" - Establece un retraso entre cada intento de lectura, para mejorar sincronizaciones y no desperdiciar computo si se sabe que el emisor tiene un delay.
- parámetros adicionales - Cada tipo de socket tiene unos parámetros adicionales. Ver subsección contigua de cada tipo de socket para mas detalles.

5.1. Class Name

Las clases actualmente implementadas en el sistema son:

- SerialSocket, este socket abre una comunicación serie y trata los datos leídos.
- FileSocket, este socket lee lineas de un fichero y las trata como entrada de datos.
- MqttSocket – Underconstruction lee datos de un topic mqtt

5.1.1. SerialSocket

–todo–

5.1.2. FileSocket

–todo–

5.1.3. MqttSocket

–todo–

5.2. Control

Control es una lista de topics con acciones asociadas. El sistema se subscribe a esos topics y cuando reciben los mensajes establecidos realizan las acciones asociadas. La naturaleza de la acción está determinada por el tipo de socket en el que se encuentre. Por ejemplo en un FileSocket los únicos elementos de control son mandar un 0, que para la lectura del fichero o un 1 que continua la lectura del mismo. Mientras que en un SerialSocket el string que aparezca en action será retransmitido por el puerto serie.

Los elementos de la lista de control son los siguientes:

Una lista [] de diccionarios, donde cada diccionario corresponde a un topic de entrada que contiene las siguientes keys:

- "topic", topic al que tiene que suscribirse el sistema
- "qos", qos de la suscripción.
- "actions", lista de diccionarios con el par de claves "message" - "action" que determina el comportamiento concreto.

5.3. Publish

Aquí es donde se define que hacer con los datos leídos, costa de tres elementos (keys) "parser", "join" y "topics"

5.3.1. parser

Es un string donde se establece el formato del payload que vamos a recibir en caso de tener varios valores. Cada valor tiene que estar numerado del 0 al 9 y encerrado en corchetes. Como en el siguiente ejemplo:

```
"parser": "#{0}:{1};",
```

5.3.2. Join

Si queremos mandar los datos a un Join tenemos que usar este clausula donde determinamos que dato va a que join y en que posición. Y en caso de necesario también indicamos cual es el dato de sincronización. La key 'join' contiene una lista de diccionarios donde cada uno tiene las siguientes keys:

- "from" - Código numérico del dato que queremos mandar del 0 al 9 (corresponde al mismo número que el parser)
- "to" - Id del joiner al que queremos mandarlo
- "position" - La posición dentro de ese joiner
- "sync" - Posición en el parser que contiene el dato de sincronización.

5.3.3. topics

En topics tenemos una lista estructurada de topics. El elemento principal es un diccionario donde el primer nivel de claves corresponden a las posiciones (números) en el parser, que contienen a su vez otro diccionario que contiene a los topics concretos de ese valor.

Hasta que no tenga definido el formato de declaración formal de la estructura no creo que pueda explicarlo mejor con palabras sin enredar mucho, por lo que paso directamente a unos ejemplos donde se puede entender mejor la estructura completa del socket.

5.4. Ejemplo

Ejemplo con dos sockets:

```
"sockets": {
  "id_filesocket_interferometer_1" : {
    "class_name": "FileSocket",
    "control":[
      {
        "topic" : "interferometer/control/power",
        "qos" : 1,
```

```
    "actions":[
      {
        "message":"stop",
        "action":0
      },
      {
        "message":"start",
        "action":1
      }
    ]
  }
],
"publish": {
  "parser": "#{0}:{1};",
  "join": [
    {
      "from": 0,
      "to": "join_1",
      "position": 0,
      "sync": 1
    }
  ],
  "topics": {
    "0": {
      "raw":{
        "function": "Raw",
        "samples": 1,
        "topic": "interferometer/value",
        "to_db":{
          "table":"interferometer/raw",
          "column":"payload",
          "db_id":"db0"
        }
      }
            },
    "1": {
      "raw":{
        "function": "Raw",
        "samples": 1,
        "topic": "interferometer/sync"
      }
```

```
                }
    },
    "file_name": "filename.txt",
    "delay": 0.001
  },

  "id_serialsocket_interferometer2": {
    "class_name": "SerialSocket",
    "control":[
      {
        "topic" : "interferometer/control/msg",
        "qos" : 1,
        "actions":[
          {
            "message":"stop",
            "action":"stop"
          },
          {
            "message":"start",
            "action":"start"
          }
        ]
      }
    ],
    "eol_character":";",
    "port": "/dev/ttyACM0",
    "baudrate":9600,
    "publish": {
      "parser": "#{0}:{1};",
      "join": [
        {
          "from": 0,
          "to": "join_1",
          "position": 1,
          "sync": 1
        }
      ],
    "delay": 0.001
  }
```