Tempo2Market Websocket Notification Interface

1 General info

Default port	1777
Protocol	"ttmnotify"

2 Format

The "ttmnotify" interface uses simple **JSON** formatted messages to inform about changes. All messages represent a JSON object having the `cmd` attribute and an arbitrary amount of other attributes. They all follow the structure `{"cmd":string, ...}`. Based on `cmd` attribute the corresponding message scheme is applied in order to determine the exact message format.

3 Messages schemes

3.1 Jobs

3.1.1 Jobs-list update notification

If a new job is created or an existing job is removed, the server will broadcast a notification message with a `cmd` attribute set to **"refresh"**. A `refresh` cmd always contains the `reason` and `jobid` attributes. The `reason` attribute is set to **newjob** when a new job is added or it is set to **jobremoved** if a job has been removed. The `jobid` attribute contains the id of the job that has been added or removed. It is recommended to request the full list of jobs through the REST-Api when you receive the **refresh** `cmd`.

Note
In a productive system the application server may be configured without the support for adding or removing jobs. Thus, this notification message may never be sent.

Scheme:

```
{
  "cmd": "refresh",
  "reason": string, // "newjob" or "jobremoved"
  "jobid": integer
}
```

Example:

```
{ // a new job with id 12 was added to the list
  "cmd": "refresh",
  "reason": "newjob",
  "jobid": 12
}
```

3.1.2 Job update notification

If any job is updated, the server will broadcast a notification message with a `cmd` attribute set to **"updatejob"**. An `updatejob` cmd always contains the `jobid` and `status` attributes. The `status` attribute contains the most recent status of the job.

Note
In a productive system only the status of a job is expected to change. However, in a developer system, where jobs can be edited, all attributes and even the code of a job may change during runtime.

Scheme:

```
{
  "cmd": "updatejob",
  "jobid": integer,
  "status": integer // the enumeration status as defined below
}

enumeration status
{
    Unknown = 0,            // don't know
    Running = 10,           // running
    Starting = 11,          // about to start
    Stopping = 12,          // about to stop
    Stopped = 13,           // don't know if it is about to start
    FailedToStart = 14,     // failed to start
    Exited = 15,            // exited normally
    ExitedWithError = 16,   // exited with error/failed
    Crashed = 17,           // the process crashed or went out of memory
    KilledByWatchdog = 18   // the process was killed by the watchdog
};
```

Example:

```
{ // the job 12 has entered the "running" state
  "cmd": "updatejob",
  "jobid": 12,
  "status": 10
}
```

3.2 GPIOs

The GPIOs in a TTM system are preconfigured and ready to use. All GPIOs are accessible through their names. To query the GPIOs in the system, the REST-Api shall be used.

3.2.1 GPIO update notification

The server will send GPIO change notification every time the value of any gpio changes, regardless of input or output. A GPIO value change is notified with `cmd`=**"gpioupdate"** accompanied by the `gpio` and `value` attributes. The `gpio` attribute carries the name of the GPIO, which is unique in the system, while `value` is set to the most recent value 1 or 0.

Scheme:

```
{
  "cmd": "gpioupdate",
  "gpio": string,       // name of the GPIO
  "value": integer      // 1 or 0
}
```

Example:

```
{ // The relay A is switched to 1
  "cmd": "gpioupdate",
  "gpio": "relay-a",
  "value": 1
}
```

3.3 Database updates

The database maintained by a TTM server can contain multiple shared key-value pairs. Some of them may be persistent, others may be connected to the cloud. In case a shared value changes the server may send a value update notification. Use the REST-Api to query the shared values in the system or set new values and ultimately remove them. A shared value can be provided by any component or any job in the system and it's key must be chosen to be unique in the system. A shared value is always a key-value pair, where the key and the value are both strings.

3.3.1 Shared value update notification

The server will only send shared value notifications if the provider of the shared value didn't explicitly instructed the server to omit the notification. A notification on a shared value is always signaled with a message having `cmd`=**"dbchange"**. The notification message with `cmd`="dbchange" contains also the `key`, `value` and `reason` attributes. The `key` contains the unique name of the updated shared value, while `value` contains its value as a string. The provider of a shared value is encouraged to encode any complex object or any simple type as a string in order to provide useful data to the rest of the system. The `reason` attribute is set to **"added"**, **"updated"** or **"removed"** to provide more detail about the reason for the update. In case the `reason` attribute in a notification message is set to "removed", the `value` will contain an empty string.

Scheme:

```
{
  "cmd": "dbchange",
  "key": string,       // name of the shared value
  "value": string,     // any possible value encoded as string
  "reason": string     // the reason for the update: "added", "updated", "removed"
}
```

Example:

```
{ // a new shared value called "special-setting-A" has been added by a client
  "cmd": "dbchange",
  "key": "special-setting-A",
  "value": "Some arbitrary setting",
  "reason": "added"
}

{ // another client changes the value to "blah"
  "cmd": "dbchange",
  "key": "special-setting-A",
  "value": "blah",
  "reason": "updated"
}
```