RECSDaemon

Introduction

The RECSDaemon is a small program that can be installed on compute modules in a RECS|Box system to be able to forward OS-level moinitoring data to the integrated anagement system of the RECS|Box. It is written to be cross-platform, running on Microsoft Windows as well as Linux and on x86, x64 and ARM systems. To be able to adapt to different platforms, the RECSDaemon uses plugins for different purposes. To configure these plugins and other settings an .ini file is used. The RECSDaemon is also able to execute commands sent by the management system to the node (e.g. for shutting down the OS gracefully).

Installation

The RECSDaemon is currently available as a Debian packet (.deb) that can simply be installed using the distribution's packet manager. Packets are provided for x86, x64 and ARM hardfloat setups.

After downloading the file you can install it e.g. on Ubuntu using the following command:

```
dpkg --install RECSDaemon_3.5.0.deb
```

The installation script will try to auto-detect some of the configuration parameters, but as platforms supported by the daemon are very diverse, manual configuration of remaining parameters most probably will be necessary.

Configuration

The RECSDaemon will by default be installed to

```
/opt/RECSDaemon
```

The configuration file thus will be

```
/opt/RECSDaemon/conf/recsdaemon.ini
```

As this is a standard INI file, it is devided into different sections (denoted by square brackets) with parameters that are set to a certain value (e.g. updateInterval=1000). You can edit this file using a text editor, e.g. nano or vi. However, you probably will need root privileges to do so.

The different aspects that need to be configured will be described in the following chapters.

Communication

To be able to send sensor values and to receive commands, the RECSDaemon has to communicate with the management system of the RECS®|Box. This can happen via one of two different communication channels: On the one hand it is possible to use the internal management bus (I2C) of the RECS®|Box. On the other hand, regular TCP/IP is also supported when an external connection between the node the RECSDaemon is running on and the management Ethernet port of the RECS®|Box is made. Without this external connection, only I2C can be used.

As access to the external I2C bus of compute modules can differ between module vendors, there are multiple plugins available:

Plugin	**Use for**
LinuxCommunicatorDev	All modules that provide the external I2C bus as a /dev/i2c-* device
CommunicatorCongatec	COM Express modules from Congatec, uses CGOS
CommunicatorKontron	COM Express modules from Kontron, uses KEAPI
CommunicatorTCP	Communication via TCP/IP
CommunicatorDummy	Testing only, no external data transfer happening

Some of these plugins need further configuration. The necessary parameters are shown in the next chapters, all of which belong in the `[Comm]` section.

LinuxCommunicatorDev

If neccessary, the I2C bus to be used can be changed. This is done by setting the `i2cBus` property. The value is used to determine the device path by appending it to the base path “/dev/i2c-”.

Example configuration:

```
[Comm]
PluginName=LinuxCommunicatorDev
i2cBus=0
```

CommunicatorTCP configuration

When utilizing I2C as the communication channel, the RECSDaemon automatically can determine on which baseboard in the RECS®|Box it is running. However, when using TCP/IP this information has to be supplied in the configuration. This is done by setting the `baseboard` parameter to the number of the baseboard this module is currently plugged into. Please remember to update this value when you move the module to another baseboard.

Also neccessary is the IP addess of the RECS®|Box management system, which is set by the `controller` property. Set this value to the IP of the RCU the module is contained in.

Example configuration:

```
[Comm]
PluginName=CommunicatorTCP
baseboard=2
controller=192.168.13.56
```

Slot detection

Because some of the RECS®|Box baseboards support multiple modules, nodes need to know in which slot of the baseboard they are located. This is done by pins on the module connector that are tied to different logic levels depending on the slot position. These values can be read by the RECSDaemon using a SlotDetector plugin.

Plugin	Detection method
LinuxSlotDetectorGPIO	Accesses GPIOs via the Linux GPIO Sysfs interface

If no SlotDetector can or should be used, the slot can be manually changed by setting the `defaultSlot` property in the `[Slot]` section. Slots are configured from 0 to 3, but shown in the WebGUI of the RECS®|Box as 1 to 4.

LinuxSlotDetectorGPIO configuration

This plugin needs the numbers of the GPIO pins used to sense the slot position as used by the running kernel. These are set with the `Bit0GPIO` and `Bit1GPIO` settings.

Example configuration:

```
[Slot]
slotPluginName=LinuxSlotDetectorGPIO
Bit0GPIO=133
Bit1GPIO=134
```

Sensor providers

Sensors

Other settings

TCP/IP server

The RECSDaemon provides a simple TCP/IP server (by default on port 2023) that can be used by external programs to gain information or provide additional sensors.

The following commands are currenty supported:

Command	Action
getnodeid	Returns ID of this node as shown in the WebGUI
monitor	Returns sensor data (current, voltage, temperatures) from the baseboard as a JSON string
addsensors	Used to add sensors by giving a JSON description
updatesensors	Updates sensors added previously
exit	Terminates connection

Adding and updating sensors

Sensors added via the TCP/IP server are managed in groups that are identified by arbitrary strings. When adding sensors the group is defined via the `addsensors` command and referenced when using the `updatesensors` command.

See chapter “JSON sensor description” for the syntax of the expected JSON string for `addsensors`. To add a group “mySensors” with one sensor “mySensor”, send the following command, terminated with a newline (\n):

```
addsensors mySensors [{"name": "mySensor", "dataType": "double"}]
```

To update a sensor group, send a `updatesensors` command with the group name and a JSON array with as many values as the group contains sensors:

```
updatesensors mySensors [1.0]
```

JSON sensor description

A JSON sensor description contains one or more sensors, thus the outer element is an Array ([]). Inside that, the sensors are defined as objects ({}) that support the following properties:

Property	Possible values	Required	Description
name	String, max. 29 characters	Yes	Name of the sensor
dataType	U8, U16, U32, U64, double, string	Yes	Data type of the sensor. U8 to U64 mean unsigned integers of the given width.
maxDataSize	1 - 255	Only for dataType “string”	Maximum length of the sensor value
unit	W, A, V, °C, RPM	No	Unit of the sensor value
lowerThresholds	JSON array with two doubles	No	Lower critical and warning threshold as doubles
upperThresholds	JSON array with two doubles	No	Upper warning and critical threshold as doubles

Example:

```
[
```

```
    {
        "name": "1.0 V",
        "dataType": "double",
        "unit": "V",
        "lowerThresholds": [0.5, 0.8],
        "upperThresholds": [1.2, 1.5]
    },
    {
        "name": "Temp. Heatsink",
        "dataType": "double",
        "unit": "°C"
    }
]
```