

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 monitoring data to the integrated management 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 open source and available as [source code on github](#), including a documentation how to compile and install it from source. Pre-packed Linux packages (rpm and deb) are under construction and will follow shortly.

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.

RECSDaemon comes with both systemd and classical init scripts to allow automatically starting the daemon during system boot. Depending on your distribution, use either

```
systemctl enable RECSDaemon
```

or

```
update-rc.d RECSDaemon defaults
```

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 divided 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 necessary, 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 file. 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 necessary is the IP address 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

Sensor providers, as the name implies, add one or more sensors to the RECSDaemon's sensor list. This is in contrast to normal sensors that are manually configured (see chapter "Sensors"). Sensor providers can have two different internal designs: They either can directly provide and update sensor objects or they can use JSON as the interface to the RECSDaemon. Depending on the type, the plugins are enabled by adding them to either the SensorProviders or SensorProvidersJSON properties in the [Plugins] section.

Currently the following sensor provider plugins are available:

Plugin	Type	Sensors provided
LinuxSensorProviderEth	Direct	Ethernet link status/speed, bytes/sec RX/TX
SensorProviderSystem	Direct	CPU utilization, RAM free, System disk free

Plugin	Type	Sensors provided
SensorProviderZynq	Direct	Serial transceiver link, utilization, bandwidth, frame errors, soft errors
SensorProviderZynqModule	JSON	Sensors provided by the module's microcontroller: 10 voltage rails, 5 power measurements, 7 temperatures

Example configuration:

```
[Plugins]
SensorProviders=SensorProviderZynq, LinuxSensorProviderEth, SensorProviderSystem
JSONSensorProviders=SensorProviderZynqModule
```

The plugins with “Zynq” in their name are only usable on the christmann RECS®|Box Zynq COM Express module.

Some of the plugin need further configuration. Please see the following chapters for details.

SensorProviderZynq

This plugin needs the base address of the monitoring peripheral it should read in the Zynq's memory space. Configure it with the `auroraMonitorBaseAddress` property in the `Plugins` section.

SensorProviderZynqModule

This plugin needs the name of the serial port of the management microcontroller on the Zynq module. Configure it with the `zynqSerialPort` property in the `Plugins` section. On Linux, it typically would be `“/dev/ttyUSB0”`.

Sensors

Besides `SensorProviders` that automatically add a set of sensors when enabled, the `RECSDaemon` also allows manual configuration of additional sensors.

Currently, the following plugins can be used to instantiate sensors manually:

Plugin	Use for
<code>SensorFileReader</code>	Value read from arbitrary file

Sensors are instantiated in the `[Sensors]` section of the configuration file. The value `count` gives the total number of configured sensors. For each sensor (counting from 1 to `count`) three properties are read: `pluginNameX`, `sensorNameX` and `optionsX`, where `X` is replaced with the current count (e.g. `pluginName1`, `pluginName2`, etc.).

`pluginName` gives the name of the plugin to be used for this sensor, `sensorName` the name of the sensor (max. 29 characters) as shown in e.g. the WebGUI of the RECS®|Box. The value of the

options property is given to the plugin and is used for further configuration. It is usually a space-separated list of “option=value” pairs.

An example configuration for one manual sensor could be:

```
[Sensors]
count=1

pluginName1=SensorFileReader
sensorName1=Test
options1=path=test.txt type=U8
```

SensorFileReader configuration

This plugin reads and parses the first line of a given file each time it is requested to update the sensor value. It recognizes the following options:

Option	Possible values	Required	Description
path	String	Yes	Path of the file to be read
type	U8, U16, double	Yes	Data type of the sensor. U8 and U16 mean unsigned integers of the given width.
multiplier	Double	No	When type is “double”, the read value can be multiplied with the value in this option (e.g. to calculate a temperature from a raw sensor reading)
unit	°C	No	Unit of the sensor value
lowerCriticalThreshold	Double	No	Lower critical threshold as double
lowerWarningThreshold	Double	No	Lower warning threshold as double
upperWarningThreshold	Double	No	Upper warning threshold as double
upperCriticalThreshold	Double	No	Upper critical threshold as double

Other settings

In the Update section the rate with which the RECSDaemon updates its virtual sensors and sends them to the management system can be configured with the `updateInterval` property. The value is in milliseconds and gives the time between two updates. Beware that this only changes the update rate of the daemon. The RECS®|Box management system has its own update rate with which it collects sensor values. Thus, if you set this value smaller than the update rate of the management system, the effective update rate will still be that of the management system.

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 currently 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

Getting monitoring data

Using the `monitor` command, the RECSDaemon can be utilized to obtain measurements with a faster update rate than via the regular RECS®|Box management system. Each time the daemon receives the command, it causes the baseboard to sample its local sensors and reads the results. Because this uses the module's I2C connection, it will not work when the CommunicatorTCP plugin is used.

The returned JSON string has the following format:

```
[  
{"name": "nodeCurrent", "value": 0.0, "unit": "A"},  
{"name": "pegCurrent", "value": 0.0, "unit": "A"},  
{"name": "12vSupply", "value": 12.0, "unit": "V"},  
{"name": "temperatures", "values": [0.0, 0.0, 0.0, 0.0, 0.0], "unit": "°C"}  
]
```

The `pegCurrent` entry is optional and only available on COM Express baseboards.

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"
  }
]
```

From:

<https://recswiki.christmann.info/wiki/> - RECS®|Box Wiki

Permanent link:

<https://recswiki.christmann.info/wiki/doku.php?id=documentation:recsdaemon&rev=1506525355>

Last update: 2017/09/27 15:15

