

Chassis Manager User Manual

Key Features:

- Compliant to IPMI 2.0
- Compliant to VITA46.11-2022
- SOSA Aligned
- IPMI Firewall Supported
- HPM.1 firmware upgrade
- GUI software for configuration
- Configurable digital IOs
- Analog inputs for voltage, current or temperature measurements
- External I2C for temperature measurements and communication with payload
- Runs on top of FreeRTOS operating system

Revision history:

- 17.10.2023 REV 0.4: - Added support for ipmitool file update; revised update protocol section
- 10.10.2023 REV 0.3: - Added fan control support
- 09.10.2023 REV 0.2: - Updated draft release
- 17.08.2023 REV 0.1: - Draft Release

Table of Contents

1 Product description and functions	4
2 BIT (Built In Test).....	5
2.1 P-BIT	5
2.2 C-BIT	5
3 Supported IPMI Commands	5
4 Chassis Discovery	8
4.1 IPMC Discovery.....	8
4.2 System IPMB Discovery	8
4.3 IPMC/FRU Polling.....	8
5 Event Handling	9
5.1 System Event Log (SEL).....	9
6 Monitoring environmental parameters.....	9
6.1 Voltage monitoring.....	9
6.2 Temperature monitoring.....	9
6.3 Fan monitoring and control.....	10
6.3.1 Fan control.....	10
6.4 Digital Inputs	11
6.5 Digital Outputs.....	11
6.6 Control Bits	12
6.7 Local Sensor Numbers.....	12
7 Conditions.....	14
8 Ethernet interface	18
8.1 RMCP	18
9 RS232 serial interface	18
10 Command Line Interface (CLI)	19
10.1 bit command.....	19
10.2 channel command.....	20
10.3 conditions command	20
10.4 controlbits command.....	20
10.5 fancontrol command.....	21
10.6 firewall command.....	23
10.7 fru command	24
10.8 fruinfo command.....	24
10.9 help command	25
10.10 info command	25
10.11 ipmb command	25
10.12 lanconfig command	25
10.13 logout command	26
10.14 nvm command.....	26
10.15 payload_reset.....	26

10.16	payload_signals	26
10.17	reboot command	27
10.18	saveenv command.....	27
10.19	sel command	27
10.20	sendipmb command	28
10.21	sensor command	28
10.22	settings command	28
10.23	uptime command	29
10.24	user command	29
10.25	version command.....	29
10.26	xmodem command.....	29
11	Update protocol.....	29
11.1	Update over RS232 using xmodem.....	29
11.2	Updating the firmware.....	31
11.2.1	Firmware update via RS232 CLI.....	31
11.3	Updating the configuration files.....	31
11.3.1	SDR file update via RS232.....	31
11.3.2	SDR file update via ipmitool.....	31
11.3.3	FRU file update via RS232.....	32
11.3.4	FRU file update via ipmitool.....	32
11.3.5	Chassis FRU file update via RS232	32
11.3.6	Conditions file update via RS232.....	33
12	Electrical and Environmental Parameters	33

List of Figures

Figure 1:	Chassis Manager Interface	4
Figure 2:	Fan control algorithm	10
Figure 3:	Conditions FSM diagram	16
Figure 4:	Sending files via xmodem using TeraTerm	30

List of Tables

Table 1:	List of Supported Commands	7
Table 2:	Fan control algorithm.....	11
Table 3:	Local sensor numbers.....	14

1 Product description and functions

The purpose of the Chassis Manager is to watch over the basic health of the System Platform, to report anomalies, and takes corrective action when needed. The Chassis Manager can retrieve inventory information or sensor readings, can perform basic recovery operations such as power cycle or resets.

The Chassis Manager communicates with other Field Replaceable Units (FRUs) inside the VITA 46.11 System Platform by sending IPMI messages over I2C buses (IPMB).

Beside managing intelligent FRUs attached to the IPMB, the Samway's Chassis Manager provides dedicated inputs for external sensors that could be used for monitoring chassis parameters like: backplane voltages and currents, temperatures, fans speeds, digital signals (PSU FAIL, switch buttons...). The chassis manager also provides configurable digital outputs that could be used to turn on LEDs or shut down PSUs in case of failures. (Figure 1)

The Chassis Manager uses Sensor Data Records to describe the monitored parameters. For any measured parameter up to 6 thresholds can be defined: lower non critical, lower critical, lower non-recoverable, upper non critical, upper critical and upper non-recoverable. The measured values are retrievable at any time via the RS232 serial interface or Ethernet. In addition, limits and system parameters can be changed at any time with the unit in service. As a result, the Chassis Manager – and hence the connected system - can be controlled and monitored online via any computer with an Internet connection.

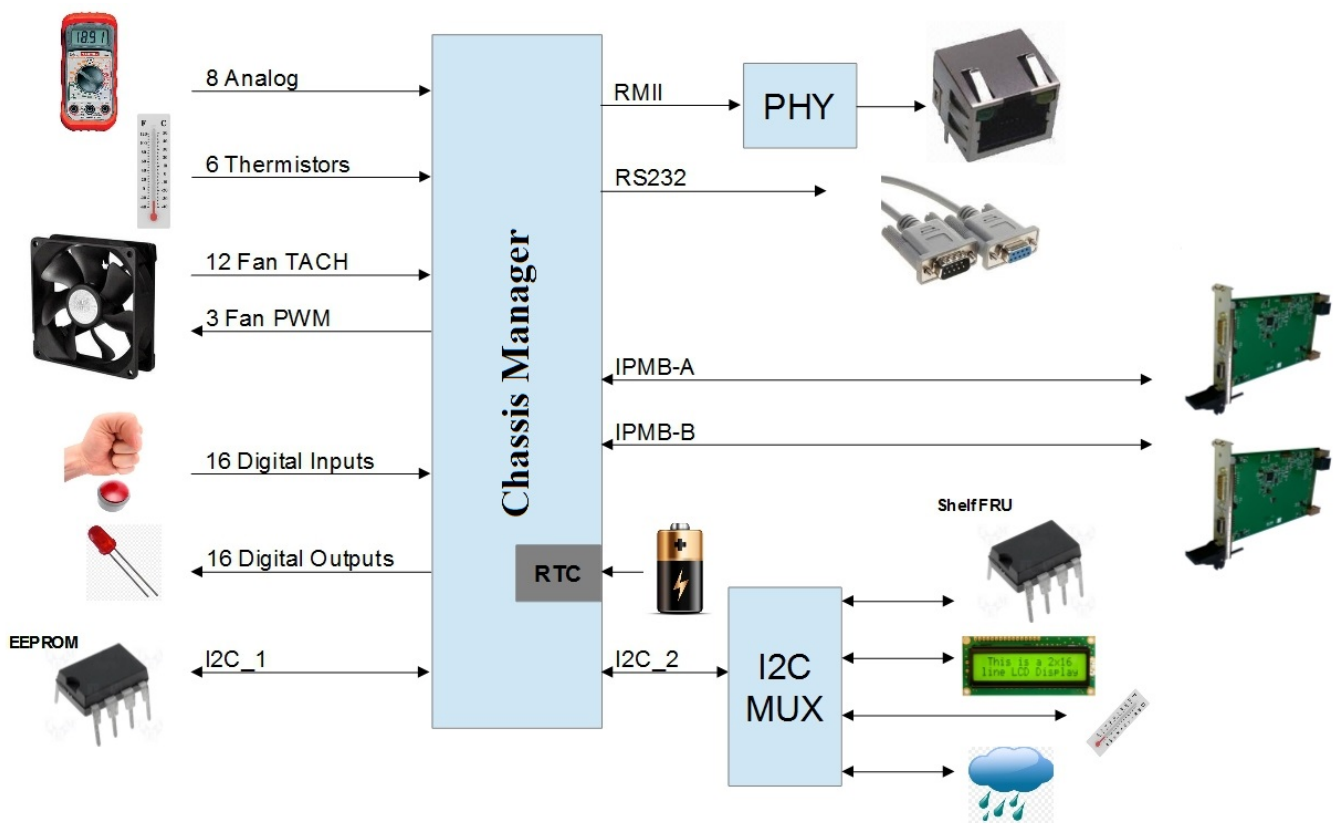


Figure 1: Chassis Manager Interface

2 BIT (Built In Test)

The Chassis Manager is equipped with 2 different BIT functions:

- **P-BIT:** power on BIT. It runs at every power on and verifies the functionality of the main hardware components of the Chassis Manager
- **C-BIT:** continuous BIT. Runs all the time and verifies that all monitored values are ok

2.1 P-BIT

This set of tests runs at every power on of the Chassis Manager and its purpose is to verify that all the hardware components are operating correctly. After all checks are done the result of the test can be “none” or different error info :

2.2 C-BIT

The Continuous BIT performs a continuous check of the Chassis Manager. Errors are triggered if one component fails or there are other problems detected by Chassis Manager

3 Supported IPMI Commands

The Chassis Manager was developed based on the IPMI v2.0 and ANSI/VITA 46.11 specifications.

IPM Device “Global” Commands	NetFn	CMD
Get Device ID	App	01h
Cold Reset	App	02h
Get Self Test Results	App	04h
Get ACPI Power State	App	06h
Send Message	App	34h
Get Channel Authentication Capabilities	App	38h
Get Session Challenge	App	39h
Activate Session	App	3Ah
Set Session Privilege Level	App	3Bh
Close Session	App	3Ch
Get Session Info	App	3Dh
Chassis Commands	NetFn	CMD
Get Chassis Capabilities	Chassis	00h
Get Chassis Status	Chassis	01h
Chassis Control	Chassis	02h
Firewall	NetFn	CMD
Get NetFn Support	App	09h
Get Command Support	App	0Ah
Get Command Sub-function Support	App	0Bh

Get Configurable Commands	App	0Ch
Get Configurable Command Sub-Functions	App	0Dh
Set Command Enables	App	60h
Get Command Enables	App	61h
Set Configurable Command Sub-function Enables	App	62h
Get Configurable Command Sub-function Enables	App	63h
Get OEM NetFn IANA Support	App	64h
Event Commands	NetFn	CMD
Set Event Receiver	S/E	00h
Get Event Receiver	S/E	01h
Platform Event	S/E	02h
Sensor Device Commands	NetFn	CMD
Get Device SDR Info	S/E	20h
Get Device SDR	S/E	21h
Reserve Device SDR Repository	S/E	22h
Set Sensor Hysteresis	S/E	24h
Get Sensor Hysteresis	S/E	25h
Set Sensor Threshold	S/E	26h
Get Sensor Threshold	S/E	27h
Set Sensor Event Enable	S/E	28h
Get Sensor Event Enable	S/E	29h
Get Sensor Reading	S/E	2Dh
FRU Device Commands	NetFn	CMD
Get FRU Inventory Area Info	Storage	10h
Read FRU Data	Storage	11h
Write FRU Data	Storage	12h
SDR Device Commands	NetFn	CMD
Get SDR Repository Info	Storage	20h
Reserve SDR Repository	Storage	22h
Get SDR	Storage	23h
SEL Device Commands	NetFn	CMD
Add SDR	Storage	24h
Partial Add SDR	Storage	25h
Clear SDR Repository	Storage	27h
Get SEL Info	Storage	40h
Reserve SEL	Storage	42h
Get SEL Entry	Storage	43h
Add SEL Entry	Storage	44h
Delete SEL Entry	Storage	46h
Clear SEL	Storage	47h
Get SEL Time	Storage	48h
Set SEL Time	Storage	49h

VITA 46.11 Commands	NetFn	Group ID	CMD
Get VSO Capabilities	Group Extension	VSO(03h)	00h
FRU Control	Group Extension	VSO(03h)	04h
Get FRU LED Properties	Group Extension	VSO(03h)	05h
Get LED Color Capabilities	Group Extension	VSO(03h)	06h
Set FRU LED State	Group Extension	VSO(03h)	07h
Get FRU LED State	Group Extension	VSO(03h)	08h
Set IPMB State	Group Extension	VSO(03h)	09h
Set FRU State Policy Bits	Group Extension	VSO(03h)	0Ah
Get FRU State Policy Bits	Group Extension	VSO(03h)	0Bh
Set FRU Activation	Group Extension	VSO(03h)	0Ch
Get Device Locator Record ID	Group Extension	VSO(03h)	0Dh
FRU Control Capabilities	Group Extension	VSO(03h)	1Eh
Get FRU Address Info	Group Extension	VSO(03h)	40h
Get FRU Persistent Control	Group Extension	VSO(03h)	41h
Set FRU Persistent Control	Group Extension	VSO(03h)	42h
FRU Persistent Control Capabilities	Group Extension	VSO(03h)	43h
Get Mandatory Sensor Numbers	Group Extension	VSO(03h)	44h
Get FRU Hash	Group Extension	VSO(03h)	45h
Get Payload Mode Capabilities	Group Extension	VSO(03h)	46h
Set Payload Mode	Group Extension	VSO(03h)	47h
Get Write Protect Capabilities	Group Extension	VSO(03h)	48h
Get Write Protect Enables	Group Extension	VSO(03h)	49h
Set Write Protect Enables	Group Extension	VSO(03h)	4Ah
Get Control Bits Capabilities	Group Extension	VSO(03h)	4Eh
Get Control Bits	Group Extension	VSO(03h)	4Fh
Set Control Bits	Group Extension	VSO(03h)	50h
Get Bridged NetFn Support	Group Extension	VSO(03h)	51h
Get Bridged Command Enables	Group Extension	VSO(03h)	52h
Set Bridged Command Enables	Group Extension	VSO(03h)	53h
Get Bridged Command Sub-function Enables	Group Extension	VSO(03h)	54h
Set Bridged Command Sub-function Enables	Group Extension	VSO(03h)	55h
Set Bridged NetFn Policy	Group Extension	VSO(03h)	56h
Get Bridged NetFn Policy	Group Extension	VSO(03h)	57h

Table 1: List of Supported Commands

4 Chassis Discovery

4.1 IPMC Discovery

The starting point of the discovery process is the *Chassis FRU Information*.

Using the *Chassis Address Table Record* the Chassis Manager determines the potential addresses of all the other IPMCs in the chassis. The list of potential addresses will also be filled out with the address of event generators.

For a Tier 1 IPMC the discovery process will be initiated only if it is instantiated in the Chassis Address Table.

Tier 2 and Tier 3 IPMCs will always be discovered, as they make their presence known by sending events.

The Discovery process uses a set of three commands: *Get Device ID*, *Get VSO Capabilities* and *Get Mandatory Sensor Numbers*. The purpose of the process is to determine if an IPMC is present and if it is compliant to VITA 46.11. Only IPMCs that satisfy both conditions are managed by the Chassis Manager.

The discovery process is an ongoing task and allows management for IPMCs/FRUs that appear at a later time.

4.2 System IPMB Discovery

The Chassis Manager supports both IPMB buses: A and B. IPMB A is always enabled and used to communicate with the other IPMCs.

At each startup the Chassis Manager scans the *Chassis FRU Information* and looks for the *Chassis IPMB Descriptor Record*. If the *IPMB_B Supported* bit is set, or the record is missing, the Chassis Manager also enables its IPMB B for chassis communication.

If the chassis supports IPMB B, its usage for communicating to an IPMC is decided by the discovery process of that address. If the IPMC sets the *Number of supported IPMB Interfaces* to 01b in the response of the Get VSO capabilities request that is part of the discovery process, the Chassis Manager will send a *Set IPMB State* command to the IPMC to enable IPMB B.

4.3 IPMC/FRU Polling

After a VITA 46.11 compliant IPMC/FRU has been successfully discovered, the Chassis Manager starts an ongoing periodic polling task and periodically checks IPMC's FRU operational State and availability.

5 Event Handling

5.1 System Event Log (SEL)

The Chassis Manager acts as an event receiver for all IPMCs inside a chassis and saves all events in a System Event Log (SEL), implemented using a non-volatile Flash memory.

Depending on the configuration, after the log reaches full capacity the Chassis Manager can behave in two ways:

1. If the SEL is a linear implementation a warning message will be displayed and all new events will be disregarded. New Events will be logged only after the sel is cleared.
2. If ageing is enabled: The oldest events will be erased in order to make room for new ones. The sel is split into multiple files and only one file will be erased

For each event, the log provides the following information:

- time stamp
- number and name of the sensor that generated the event
- event type: - for threshold sensors: UNR,UC,UNC,LNC,LC,LNR
- for discrete sensors: 1 (Asserted), 0 (De-Asserted)
- sensor value that triggered the event and threshold value (only for threshold sensors)

For more details and syntax refer to the **sel** command section of this user manual.

6 Monitoring environmental parameters

The Chassis Manager uses Sensor Data Records (SDRs), compliant to IPMI 1.5, to describe the monitored chassis parameters.

The chassis monitoring and control can be set up using a SDR file, to define the sensors implemented and a condition file to assert outputs based on sensor states. The SDR file can be opened, altered and saved using the GUI SDRCompiler Tool.

Using the SDRCompiler the user can:

- Select sensors implemented from a default pool sensor list
- Specify up to **6** thresholds for all sensors which have analogue readings
- Decide the way the software treats a limit infringement (events can be enabled or disabled for each individual threshold)
- Change the name for every sensor

6.1 Voltage monitoring

- Monitoring of up to 8 voltages (+3.3V, +5V, +12V, -12V and 4 user defined voltages).
- Any limit infringement is an internal event and can control any of the 16 digital Outputs.
- Voltage thresholds can be changed via the CLI or by upgrading the SDR file.
- CLI Commands for the voltage monitoring: **sensor**

6.2 Temperature monitoring

- Monitoring of up to 6 analog temperature sensors (10 Kohm NTC thermistors with $\beta=3950$)
- Temperature ranges from -20°C to $+100^{\circ}\text{C}$
- Temperature measurement accuracy $\pm 3^{\circ}\text{C}$ (max.)
- Monitoring of up to 8 TMP75 I2C sensors connected to SDA3/SCL3 (Digital Temp connector on the carrier card)
- Any limit infringement is an internal event and can control any of the 16 digital Outputs.

- Any temperature sensor can be used in the fan control algorithm (The user can choose for each temperature sensor the fan control groups the sensor is active for.)
- Commands for the temperature monitoring: **sensor**

6.3 Fan monitoring and control

- Monitoring of up to 12 fans.
- Control of PWM fans
- Speed control via 3 fan control groups
- Any limit infringement is an internal event and can control any of the 16 digital outputs.
- Commands for the fan monitoring: **sensor**

6.3.1 Fan control

The Speed of the fans can be controlled using one of the 3 fan control groups. Each group controls an independent PWM¹ signal using a user-defined temperature-speed characteristic.

The user can define the temperature-speed characteristic using 3 parameters: min level (the minimum level at which the fans operate), normal level, max level and 3 temperature thresholds (temp0, temp1, temp2).

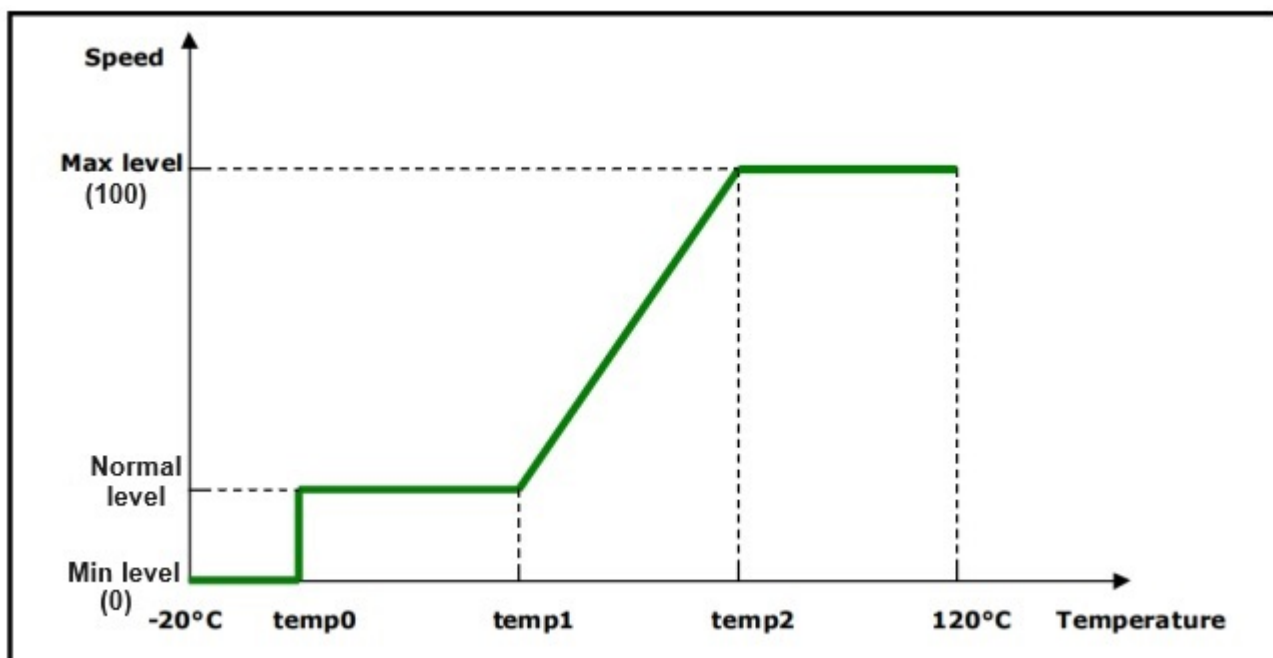


Figure 2: Fan control algorithm

¹ The Chassis Manager can output 3 PWM signals with a frequency of 1KHz.

The speed of the fans is split into 100 equal levels. At level 0 the PWM is all time low and at level the PWM is all time high.

The 3 temperature thresholds split the operating range of the fan in 4 control regions:

Temperature T	Fans Behavior:
$T < \text{temp0}$	Running at min level
$\text{temp0} < T < \text{temp1}$	Running at normal level
$\text{temp1} < T < \text{temp2}$	Running at a speed level proportional to T
$\text{temp2} < T$	Running at max level

Table 2: Fan control algorithm

Where:

- $\text{Temp0} < \text{Temp1} < \text{Temp2}$
- $\text{Min level} < \text{Normal level} < \text{Max level}$
- Min level can be set to 0
- Max level can be set to 100

The Fan control algorithm can use any of the installed temperature sensors. The temperature (**T**) that drives the algorithm is the maximum value of all the temperature sensors active. If no temperature sensors are active, the fan control algorithm sets the fans to run at the maximum level. Maximum speed level is also automatically set for all PWM groups if any fan error is detected (any monitored fan crossed the lower critical RPM threshold set in the SDR).

The 3 fan control groups are independent, each have their own fan levels, temp0, temp1, temp2 parameters. Each PWM group can have its automatic fan control algorithm disabled, this meaning the fan speed is set at a fixed manual level (override level), or it can be shut down meaning speed level 0, thus stopping all fans connected to the respective PWM signal. A PWM in shut down mode will not respond to any manual level set or automatic control algorithm, it must be reenabled in order to regain control of the fan rotation.

Commands useful for fan control: **fancontrol**

6.4 Digital Inputs

- 16 digital inputs
- Logic level: 3.3V, 5V tolerant
- Any active input generates an internal event, and can control any of the 16 digital outputs
- Commands for inputs: **sensor, controlbits**

6.5 Digital Outputs

- 16 digital outputs
- All outputs support startup sequencing
- Manual setting and clearing of outputs
- Each output can be driven by a condition. If the condition becomes true, the output is asserted. If the condition is false the output is not asserted
- Commands for outputs: **sensor, controlbits**

6.6 Control Bits

All inputs and outputs available on the Chassis Manager can be generally named control bits. The control bits are associated to VITA46.11-2022 Control Bit commands. All control bits states can be seen at once in a user friendly manner divided into banks of 8 bits each; since Samway's Chassis Manager has 16 inputs and 16 outputs, 32 total control bits are defined divided into 4 banks as follows:

Digital Inputs 1 – 8: Bank 0 (with Bit 0 = Digital Input 1 and Bit 7 = Digital Input 8)

Digital Inputs 9 – 16: Bank 1 (with Bit 0 = Digital Input 9 and Bit 7 = Digital Input 16)

Digital Outputs 1 – 8: Bank 2 (with Bit 0 = Digital Output 1 and Bit 7 = Digital Output 8)

Digital Outputs 9 – 16: Bank 3 (with Bit 0 = Digital Output 9 and Bit 7 = Digital Output 16)

See controlbits command on chapter 10 of this document for details on how to set the control bits.

***Note:** All digital outputs can be used as inputs, meaning the Chassis Manager can monitor up to 32 input signals

6.7 Local Sensor Numbers

This section refers to the local sensors implemented on the Chassis Manager. The correspondence between the sensor numbers and monitored parameters is defined in the bellow table.

! Depending on the monitored system the Chassis Manager can be set up differently. Not all the sensors described bellow are always available.

! The Chassis Manager monitors only the parameters for which a SDR has been uploaded. The monitored parameters set can be changed by uploading a new SDR set. The SDRs are encapsulated in a configuration file. Besides the SDRs, the config file also hosts several other Chassis Manager parameters.

Sensor Number	Monitored System Parameter
16	V0 Voltage
17	V1 Voltage
18	V2 Voltage
19	V3 Voltage
20	V4 Voltage
21	V5 Voltage
22	V6 Voltage
23	V7 Voltage
26	Temperature sensor 1
27	Temperature sensor 2
28	Temperature sensor 3
29	Temperature sensor 4
30	Temperature sensor 5
31	Temperature sensor 6
32	Local Temperature sensor
37	Tachometer signal for Fan1
38	Tachometer signal for Fan2
39	Tachometer signal for Fan3
40	Tachometer signal for Fan4

41	Tachometer signal for Fan5
42	Tachometer signal for Fan6
43	Tachometer signal for Fan7
44	Tachometer signal for Fan8
45	Tachometer signal for Fan9
46	Tachometer signal for Fan10
47	Tachometer signal for Fan11
48	Tachometer signal for Fan12
64	Digital Input 1
65	Digital Input 2
66	Digital Input 3
67	Digital Input 4
68	Digital Input 5
69	Digital Input 6
70	Digital Input 7
71	Digital Input 8
72	Digital Input 9
73	Digital Input 10
74	Digital Input 11
75	Digital Input 12
76	Digital Input 13
77	Digital Input 14
78	Digital Input 15
79	Digital Input 16
80	Digital Output 1
81	Digital Output 2
82	Digital Output 3
83	Digital Output 4
84	Digital Output 5
85	Digital Output 6
86	Digital Output 7
87	Digital Output 8
88	Digital Output 9
89	Digital Output 10
90	Digital Output 11
91	Digital Output 12
92	Digital Output 13
93	Digital Output 14
94	Digital Output 15
95	Digital Output 16
97	ChMC Power On
117	I2C Temperature 1 (8 bit address 0x90)
118	I2C Temperature 2 (8 bit address 0x92)
119	I2C Temperature 3 (8 bit address 0x94)
120	I2C Temperature 4 (8 bit address 0x96)
121	I2C Temperature 5 (8 bit address 0x98)
122	I2C Temperature 6 (8 bit address 0x9A)

123	I2C Temperature 7 (8 bit address 0x9C)
124	I2C Temperature 8 (8 bit address 0x9E)

Table 3: Local sensor numbers

7 Conditions

The Chassis Manager user can define scenarios (called "conditions") dependent on any local sensor by defining and uploading a special type of file called "conditions file".

A conditions file is just a regular *.txt file that included one or more conditions. Each condition is a list of definitions (directives). A condition must have a FORMULA, a set of logical operations applied on local sensors that triggers an output bit assertion when the formula result is true. The BIT directive defines the output which is controlled by condition's formula. Other directives can be set to control the timing of output's assertion/deassertion. can be edited in any text editor preferred in which for each GPIO that was first defined in the SDR records (this step is critical as in order to configure a GPIO it first needs to have a sensor number) the user can assign any desired functionality for each one of them by defining conditions. A condition checks if a certain event happened and asserts an output in a predefined way in case that all stated conditions are met. The syntax for a new condition looks as following:

```

CONDITION <condition_name>={
FORMULA:# <sensor_no_1> = | != <event_1> [OR|AND
    <sensor_no_2> = | != <event_2> [OR|AND
    <sensor_no_3> = | != <event_3> ]]
    *list can go on...

START_DELAY=<value>;
MIN_RUN=<value>;
MAX_RUN=<value>;
STOP_DELAY=<value>;
BIT=# <bit_number>;
}
```

where:

- **Condition name** – a suggestive name to identify the condition
- **FORMULA** – here the user **defines** the **desired scenario** that **triggers** an **output**. If the **formula statement** is **true**, the **condition** will be **asserted**. The "**sensor_number**" field is the **number** of the **monitored sensor** and the "**event**" field is the **sensor- state or value**. For example, to trigger a certain scenario when a value from a temperature sensor reaches the upper critical threshold, write FORMULA:#<temp_sensor_number>=UC; in this case, if the statement becomes true (meaning that the temperature sensor has reached the upper critical threshold) the condition becomes asserted. Whenever the formula statement becomes true, the condition will be asserted. In the event field user can state **LNR, LC, LNC, UNC, UNC, UC, UNR** or **NO_EVENT** in case the situation where a sensor's value hasn't reached any of the defined thresholds is desired, or state **ASSERTED** or **DEASSERTED** in case that the monitored sensor is an **input sensor**. Logical operations (**OR/AND**) can also be used to combine events; for example if the scenario where a sensor (ex number 16) has reached any of the non-recoverable thresholds is desired the formula should look like this: Ex. 1: FORMULA: #16=LNR OR #16=UNR; Ex. 2: FORMULA: #16=LC AND #15=NO_EVENT. This statement will be true when the sensor number 12 will become lower critical and at the same time sensor number 15 hasn't exceeded any thresholds.

- **START_DELAY** – Is a value that **delays** the **assertion** of the **output**. When a **condition** becomes **asserted** the **output** associated with it will become instantly **asserted**. But if the start delay value is greater than 0, at the moment the condition becomes asserted the start delay timer will begin counting; after the delay period stated in the <value> field passes, the condition's formula is checked again. If the condition is still true, the output will be asserted, if the condition becomes false, the output will remain deasserted. This feature is very useful in eliminating glitches thus preventing a false trigger of an event. For example if a push button is connected to an input that is used to reset a system, the situation when the button is accidentally pushed can be avoided by setting a desired start delay value of for example, 3 seconds. In this case, if the button is being held pushed for less than 3 seconds the reset event will not occur. The **timer period is 10ms** so to calculate the start delay value, divide the desired delay by 10ms. Now to obtain the 3 seconds delay in the above example: $3s/10ms=300$. In conclusion **START_DELAY = 300** will set a delay of 3 seconds.
- **MIN_RUN** – Is a value that **defines** the **minimum time** an **output** will be **asserted** regardless if the condition doesn't remain true for that period. For example a minimum run time of 2 seconds is set and the condition is true for only 1 second, the output will remain asserted until the minimum run time passes. If the condition stays true for longer than the minimum run period, the output will stay asserted for as long as the condition remains true.
- **MAX_RUN** – Is a value that **defines** the **maximum time** an **output** can remain **asserted**, regardless if the condition remains true. For example if a maximum run time of 5 seconds is set and the condition stays true for 10 seconds, the output will be deasserted after 5 seconds even though the condition is still true. This feature is useful to prevent excessive run of a device (a heater, a cooling unit, motor, pump etc.) in case of a sensor or input malfunction.
- **STOP_DELAY** – Is a value that **delays** the **assertion** of the output **after a previous deassertion**. If a condition with stop delay becomes true, it's output will be asserted; after the condition becomes false and the output is deasserted, the stop delay prevents a new output assertion for the set period, even though the condition becomes true again. An example where this feature is useful is when a condition whose output triggers a system reset; by setting a stop delay, a premature reset of the system can be avoided. For example, a push button used to reset the system connected to an input, after a system reset, if preventing a new reset within a period of 30 seconds is needed (can be a startup procedure), a stop delay of 30 seconds can be set. That way even if reset button is pressed the system will not restart within a period less than 30 seconds.
- **BIT** – This is the output (any bit in banks 2 and 3) assigned to the condition. In the <bit_number> field, the number of the desired bit that needs to be assert when the condition becomes true is defined. When the condition becomes true, the output will be immediately asserted in case a start delay period is set or the stop delay period triggered by a previous assertion hasn't yet passed. Bits 0 – 7 in bank 2 are BIT#16 – BIT#23; bits 0 – 7 in bank 3 are BIT#24 – BIT#31 (Bits 0 – 15 correspond to the digital inputs, bits 0 – 7 in banks 0 and 1) By **default** all the **values** for **start**, **stop** delays and **minimum**, **maximum** run time **are 0** so if any of those features are not required, they can be excluded from the condition definition. Also the timer period for minimum and maximum run time and stop delay time is the **same as the start delay feature (10ms)**. See the start delay section for a detailed description on how to calculate a timer value.

In the below figure a diagram describing the condition's finite state machine (FSM) operation can be seen. One thing missing from the description of each function that can be seen in the diagram is that If maximum run time feature is used it is mandatory to have a stop delay to prevent instant output assertion after maximum run timer expires. Therefore it is strongly recommended to set a stop delay time if maximum run timer is used; if a stop delay time is not defined, the software will automatically assign a stop delay equal to the maximum run time. If this scenario is desired then the stop delay value can be excluded from the condition syntax.

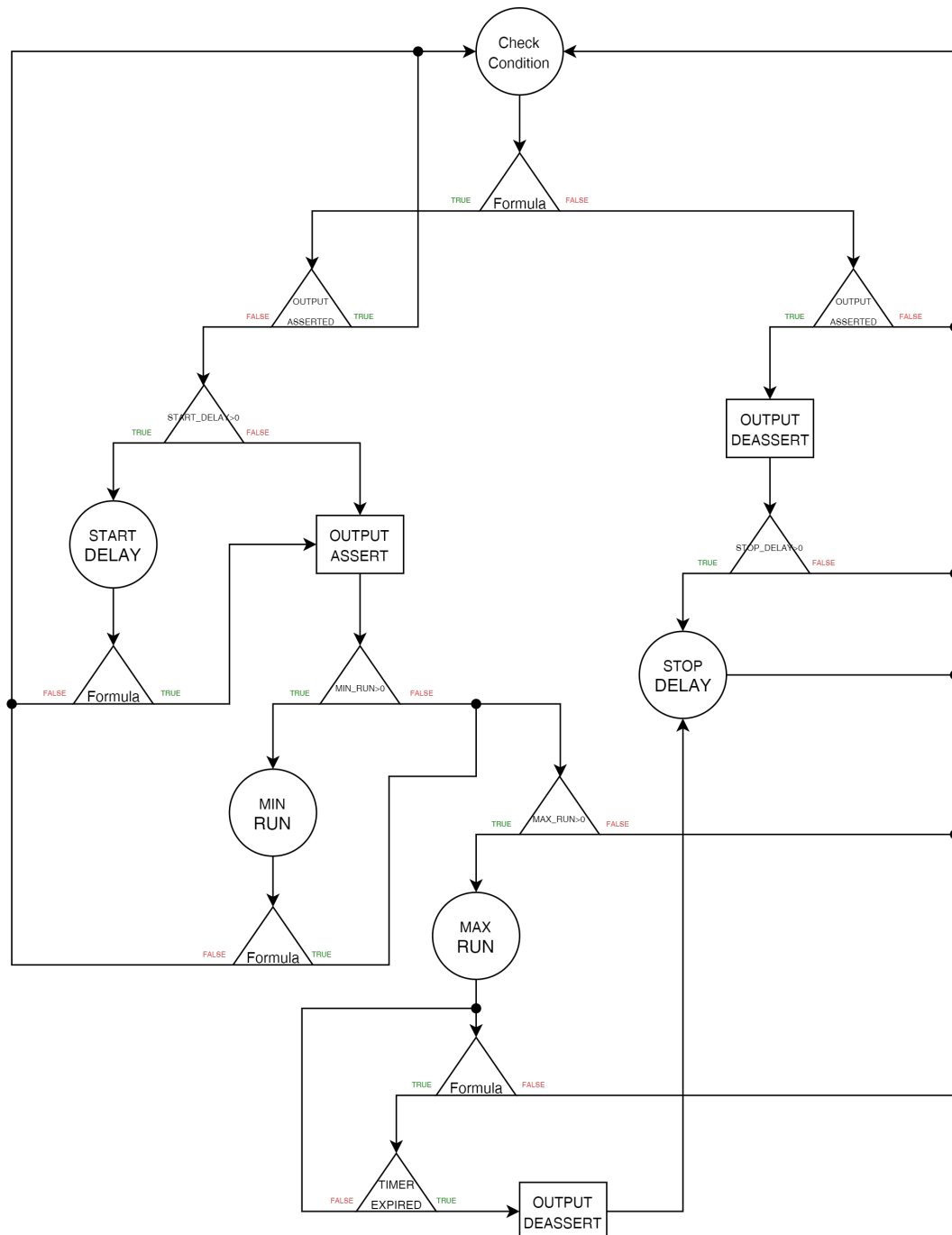


Figure 3: Conditions FSM diagram

Below there is an example of condition file where 4 GPIOs were used and 3 conditions were defined to reset the system using a push button connected to an input GPIO, to turn on a LED when a voltage is at a proper level and to blink a LED when a temperature sensor reaches upper critical threshold:

```
//$CONDFILE.V1
CONDITION RESET={
FORMULA: #64=ASSERTED;
START_DELAY = 300;
MIN_RUN = 50;
MAX_RUN = 50;
STOP_DELAY = 6000;
BIT =#18;
}
CONDITION 5V_OK={
FORMULA: #17=NO_EVENT;
BIT =#31;
}
CONDITION TEMP_UC={
FORMULA: #31=UC;
MIN_RUN = 50;
MAX_RUN = 50;
BIT =#30;
}
```

In the above example a 500ms reset pulse will be generated after a press of the push button connected to an input GPIO (sensor no. #64) for at least 3 seconds, (see the start delay) with a cool down period of 60 seconds (see the stop delay). A LED will turn on if the voltage read by the analog sensor #17 doesn't reach any threshold and a LED will start blinking with a 1 second period if the temperature sensor #31 reaches the upper critical threshold.

8 Ethernet interface

The integrated 10/100Mbps Ethernet interface allows the Chassis Manager to be linked to any existing network. The interface supports DHCP, SNMP, RMCP protocols via TCP/IP and UDP.

The use of standard protocols avoids the need for special software or drivers and so achieves platform-independence. The TCP/IP protocol supports up to 10 simultaneous connections.

! The factory default setting for the Chassis Manager is DHCP enabled so it negotiates automatically all the necessary addresses. If a fixed IP address is desired, DHCP must be disabled and the address has to be set manually. For all these operations the *lanconfig* command needs to be used.

Terminal settings:

- Local echo: *off*
- Local line editing: *off*
- Backspace key: *Control-H*

8.1 RMCP

The Chassis Manager supports Remote Management Control Protocol (RMCP). The RMCP connections requires authentication. The supported authentication protocol is MD5. Two user names are accepted for RMCP connections: "user" – with "User" privilege level and "admin" with "Administrator" privilege level. The user names and privilege levels are fixed, they cannot be changed through IPMI commands. The password used are the same passwords configured for Telnet, CLI, SNMP v1 and v2c access. The default passwords are: "USER" – for "user" profile and "ADMIN" - for admin profile. The Chassis Manager could be accessed over RMCP using standard ipmitool software.

9 RS232 serial interface

The Chassis Manager provides an RS232 serial interface through which the commands of the Command Line Interface (CLI) can be sent.

On Windows systems, we recommend the use of "TeraTerm" or "Hyperterminal" as the terminal programs.

Default Terminal settings:

- 115200 bits per second
- data bits: 8
- parity: none
- stop bit: 1

In addition, the *xmodem* command can be used via the serial interface for file transfer.

! Use a 1:1 serial cable for direct connection to the serial port of a PC.
■ When using *xmodem* in "Hyperterminal" the transfer of the desired file can take up to 10 seconds to start.

10 Command Line Interface (CLI)

The Command Line Interface (short-form: CLI) is available via the RS232 serial interface (**9 RS232 serial interface**).

The user can read or newly configure and save system parameters via the CLI.

Access is divided into 2 profiles and is password-protected.

“user” profile:

System parameters can only be read in this profile – the exception to this write-protect is the *lanconfig* command for setting the IP, subnet and gateway addresses.

“admin” profile:

Full access to all system parameters is granted. All available CLI commands can be executed. To avoid possible damage or malfunctions, the access data for this profile must only be available to trained personnel with appropriate knowledge and competence relating to the system in which the Chassis Manager is used!

The profiles can be changed using the *logout* command.

The measured values are available at any time via the RS232 serial interface. In addition, limits and system parameters can be changed at any time with the unit in service. As soon as you have established a connection, you will be prompted to log in.

Default access settings:

login: user
password: USER

login: admin
password: ADMIN



The passwords can be changed using the *user* command. The passwords can also be disabled by changing them to an empty string.

General syntax conventions

Command [parameter1 | parameter2 | parameter3 [value]]

[] = optional

The parameters separated by “|” rule out each other.

If the command line contains a *value*, this is assigned to the corresponding parameter and saved temporarily in the RAM. The change is active immediately. If the new value is desired to be valid after the reboot, the environment variables must be saved with the “*saveenv*” command. Changes not confirmed with “*saveenv*” command are lost after a reboot.

10.1 bit command

Syntax: bit [clr]

Functions:

Displays the status of the Built In Tests performed: P-BIT (Power On BIT), C-BIT (Continuous

BIT). Add the "clr" parameter to clear the current BIT status.

Example 1.

```
chm>bit

* PBIT:
None

* CBIT:
None
```

10.2 channel command

Syntax: channel [<channel_no>]

Function: Displays Chassis Manager's supported communication channels and their interfaces

Example 1.

```
chm>channel

Channel 0, Medium Type: IPMB
Channel 2, Medium Type: 802.3 LAN
```

10.3 conditions command

Syntax: conditions

Function: Displays all conditions and their state.

Example 1.

```
chm>conditions

Condition: Test
Formula: #64=ASSERTED
Value =not computed
Control Bit# 18
START_DELAY =300

Condition: 3
Formula: #16 !=UNC
Value =true
Control Bit# 31
START_DELAY =300
chm>
```

10.4 controlbits command

Syntax: controlbits [bank <bank_no> [bit <bit_no> [set|clr|in|out]]]

Function: Displays the state of each controlbit divided into banks of 8 bits each. Also each bit can be asserted/deasserted using set/clr parameters:

Example 1.

Display the state of all controlbits; to see individual banks state add the bank parameter after the main command.

```
chm>controlbits

FRU Control Bits Banks = 4

|---|--- Bank  0 ----|
|   | 7 6 5 4 3 2 1 0 |
|---|-----|
| IN|          *      |
|OUT| - - - - - - - - |
| EN| - - - - - - - - |

|---|--- Bank  1 ----|
```

```
| | 7 6 5 4 3 2 1 0 |
|---|-----|
| IN|
|OUT| - - - - - |
| EN| - - - - - |
```

```
|---|--- Bank 2 ---|
| | 7 6 5 4 3 2 1 0 |
|---|-----|
| IN|
|OUT|
| EN| - - - - - |
```

```
|---|--- Bank 3 ---|
| | 7 6 5 4 3 2 1 0 |
|---|-----|
| IN| *
|OUT| *
| EN| - - - - - |
```

```
chm>controlbits bank 2
```

```
FRU Control Bits Banks = 4
```

```
|---|--- Bank 2 ---|
| | 7 6 5 4 3 2 1 0 |
|---|-----|
| IN|
|OUT|
| EN| - - - - - |
```

```
chm>
```

Any bit can be an input, but only bits in bank 2/3 can be outputs; a "*" means the corresponding bit is asserted, and a "-" means that the feature is not available (see OUT row for banks 0/1 in above example).

To assert/deassert any output compatible bit see below example (we will assert bit 3 in bank 2):

Example 2.

```
chm>controlbits bank 2 bit 3 set
```

```
FRU Control Bits Banks = 4
```

```
|---|--- Bank 2 ---|
| | 7 6 5 4 3 2 1 0 |
|---|-----|
| IN|
|OUT|          *
| EN| - - - - - |
```

```
chm>
```

The command displays the new state of each bit in the bank.

10.5 fancontrol command

Syntax:

```
fancontrol <group_no> [override [shutdown | <l_value>]] |
[minlevel [<l_value>]] | [normallevel [<l_value>]] |
[maxlevel [<l_value>]] | [local [enable | disable]] |
[(temp0 | temp1 |temp2) [<t_value>]]
```

Function: Displays information regarding current state and configuration data for each fan control group; also, all control parameters can be modified using this command.

Command options:

- Display current state and configuration:

fancontrol

Example 1.

```
chm>fancontrol
```

```
-----
|Fan |Override|Current| Min |Normal| Max |Temp0|Temp1|Temp2| Max |Local|Fan |
|Group| Level  | Level |Level|Level |Level|      |      |      | Temp|Mode |Fail|
|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|
| 1  | Local  | 50  | 0  | 50  | 100 | 0  | 30  | 60  | 32  | ON  |   |
| 2  | Local  | 50  | 0  | 50  | 100 | 0  | 30  | 60  | 32  | ON  |   |
| 3  | Local  | 50  | 0  | 50  | 100 | 0  | 30  | 60  | 32  | ON  |   |
-----
```

- Manual override fan level:

fancontrol <group_no> override <l_value>

group_no – Group number: Fan group number; there are 3 groups numbered from 1 to 3

l_value – Level value: Fan level value to be set; any value from 0 to 100 (0 – fan stop, 100 – fan maximum speed)

Example 2.

```
chm>fancontrol 1 override 20
```

```
Override Fanlevel = 20
```

In order for the command to take effect, local control (automatic control) needs to be disabled by sending command:

fancontrol <group_no> local disable

Example 3.

```
chm>fancontrol 1 local disable
```

```
Fan group 1 local control disabled!
```

```
chm>fancontrol
```

```
-----
|Fan |Override|Current| Min |Normal| Max |Temp0|Temp1|Temp2| Max |Local|Fan |
|Group| Level  | Level |Level|Level |Level|      |      |      | Temp|Mode |Fail|
|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|
| 1  | 20     | 20  | 0  | 50  | 100 | 0  | 30  | 60  | 33  | OFF |   |
| 2  | Local  | 55  | 0  | 50  | 100 | 0  | 30  | 60  | 33  | ON  |   |
| 3  | Local  | 55  | 0  | 50  | 100 | 0  | 30  | 60  | 33  | ON  |   |
-----
```

Automatic (see section 6.3.1) control is renewable by sending command:

fancontrol <group_no> local enable

- Disable (shut down) fan group:

fancontrol <group_no> override shutdown

Example 4.

```
chm>fancontrol 1 override shutdown
```

```
Fan group 1 is turned off!
```

```
chm>fancontrol
```

```
-----
|Fan |Override|Current| Min |Normal| Max |Temp0|Temp1|Temp2| Max |Local|Fan |
|Group| Level  | Level |Level|Level |Level|      |      |      | Temp|Mode |Fail|
|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|
| 1  | Shdn   | 0    | 0  | 50  | 100 | 0  | 30  | 60  | 33  | OFF |   |
| 2  | Local  | 55   | 0  | 50  | 100 | 0  | 30  | 60  | 33  | ON  |   |
| 3  | Local  | 55   | 0  | 50  | 100 | 0  | 30  | 60  | 33  | ON  |   |
-----
```

*To restart the fans from shut down state, renewable local control

- Modify any temp parameter for a fan group:

fancontrol <group_no> [(temp0 | temp1 | temp2) [<t_value>]]

Example 5.

```
chm>fancontrol 2 temp1 55
```

```
Temp1 =55
```

```
chm>fancontrol
```

```
-----
|Fan  |Override|Current| Min |Normal| Max |Temp0|Temp1|Temp2| Max |Local|Fan |
|Group| Level  | Level |Level|Level |Level|      |      |      | Temp|Mode |Fail|
|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|
| 1   | Shdn   | 0     | 0   | 50  | 100 | 0    | 55  | 60  | 33  | OFF |
| 2   | Local  | 50    | 0   | 50  | 100 | 0    | 55  | 60  | 33  | ON  |
| 3   | Local  | 55    | 0   | 50  | 100 | 0    | 30  | 60  | 33  | ON  |
|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|
```

- Modify any fan level parameter for a fangroup:

```
fancontrol <group_no> [(minlevel | normallevel | maxlevel) [<_value>]]
```

Example 6.

```
chm>fancontrol 3 minlevel 20
```

```
Fan minimum level = 20
```

```
chm>fancontrol
```

```
-----
|Fan  |Override|Current| Min |Normal| Max |Temp0|Temp1|Temp2| Max |Local|Fan |
|Group| Level  | Level |Level|Level |Level|      |      |      | Temp|Mode |Fail|
|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|
| 1   | Shdn   | 0     | 0   | 50  | 100 | 0    | 55  | 60  | 33  | OFF |
| 2   | Local  | 50    | 0   | 50  | 100 | 0    | 55  | 60  | 33  | ON  |
| 3   | Local  | 55    | 20  | 50  | 100 | 0    | 30  | 60  | 33  | ON  |
|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|
```

```
chm>
```

10.6 firewall command

Syntax:

```
firewall [bridged] channel <channel_no> [netfn <netfn_no> [lun <lun_no>]
[command <cmd_no> [oemgroup <oem_iana|group_id>] [enable|disable|subfn
<bitmask>]]]
```

```
firewall bridged channel <channel_no> (policy | passthrough) [<value>]
```

Function: Displays or sets firewall and bridged firewall configuration.

Firewall is used to enable or disable commands, or command sub-functions, to be executed by IPMC. The configuration is made for each channel and applies only to request messages received on that channel.

Bridged firewall applies to messages forwarded by IPMC from one channel to another, using Send Message command. The configuration is based on destination channel.

Command keywords:

Bridged – if bridged parameter is entered the command will get or change the bridged firewall configuration. If bridged parameter is missing, the command will get or set the firewall configuration.

channel – the message request source channel for firewall configuration or the message destination channel for bridged message

netfn – network function

lun – LUN number. If LUN parameter is not present it is assumed to be 0

command – command number

oemgroup – group number for netfn 0x2C or oem IANA private enterprise number for netfn 0x2E

enable – enables the specified command

disable – disables the specified command

subfn <bitmask> - sets the sub-function mask bits. Parameter bitmask could be one or two 32 bits numbers, representing the sub-function mask.

Command options:

```
firewall [bridged] channel <channel_no>
```

This command returns all netfn supported on a given channel

firewall [bridged] channel <channel_no> netfn <netfn_no>

This format of firewall command returns all commands supported on a given channel – netfn combination and the current status: enabled or disabled. Some enabled commands are not configurable and cannot be disabled.

firewall [bridged] channel <channel_no> netfn <netfn_no> [lun <lun_no>] command <cmd_no>

This syntax returns detailed information about a command: status, privilege level, command name and supported sub-functions details, if the command has sub-functions

firewall [bridged] channel <channel_no> netfn <netfn_no> [lun <lun_no>] command <cmd_no> enable

Enables the specified command is the command is supported and configurable

firewall [bridged] channel <channel_no> netfn <netfn_no> [lun <lun_no>] command <cmd_no> disable

Disables the specified command is the command is supported and configurable

firewall [bridged] channel <channel_no> netfn <netfn_no> [lun <lun_no>] command <cmd_no> subfn <bitmask>

set a command sub-function mask according to bitmask value. The bitmask could one 32 bits number or two 32 bits numbers, in case the commands support extended sub-functions (up to 64 subfunctions)

firewall bridged channel <channel_no> policy [<value>]

This command returns or sets (if value provided) the policy bits for bridged messages. The policy value is a 3 bits mask, where:

bit 2 allows, if set, unknown requests (commands which are not “known” by bridged firewall) to be forwarded to selected channel

firewall bridged channel <channel_no> passthrough [<value>]

This command returns or sets (if value provided) the passthrough value for bridged messages. The passthrough value is a bit mask, where each bit represents a channel.

In case a bit is set, the commands originating for that channel are forwarded to destination channel <channel_no> directly, without applying the bridged firewall rules.

10.7 fru command

Syntax: fru [refresh]

Functions:

Displays a list of all discovered FRUs.

If an IPMC was once detected but later extracted or not responding, it will remain present in the FRUs list until a “fru refresh” command is executed. The “fru refresh” command clears the saved FRU list and this will be initialized again at next restart

10.8 fruinfo command

Syntax: fruinfo <impb_address> [<fru_id>]

Functions:

The command displays the FRU information for the desired FRU. If the fru_id parameter is missing, the command displays the information for the IPMC (FRU Id 0) identified by the requested impb address.

10.9 help command

Syntax: help [-v]

Functions:

Displays a list of all available commands or a more detailed description if the verbose attribute is used (-v).

10.10 info command

Syntax: info

Functions:

Displays information about an IPMC specified by its address. The information is a parsing of data obtained using IPMI requests.

Example 1.

```
chm> info
IPMB Addr =0xF0
Boot Count =336
Supported FRUs =1
LEDs on FRU 0 =2
```

10.11 ipmb command

Syntax: ipmb [a|b [reset|disable|(enable [100khz|400khz])]]

Functions:

Displays information about the state of the IPMB. Also any of the two busses can be reset/disabled and enabled (with 100/400KHz bus speed)

10.12 lanconfig command

Syntax:

Syntax:

```
lanconfig [dhcp [hostname | clientid] [on | off]] |
[ip | mask| gateway [ address]] |
[settings]]
```

Functions:

Readout or setting of network parameters. When used without parameters the command return the IP, mask and gateway addresses of the LAN interface

- **dhcp** – displays the current dhcp state -enabled/disabled and dhcp configuration options.
- **hostname** – if turned on will enable the DHCP option 12 hostname.
- **clientid** – if turned on will enable the DHCP option 61 client id. The format of the client ID option follows the requirements form PICMG HPM.3 R1.0 specification
- **settings** – interface settings: dhcp or local control for addresses

! The configuration changes done using this command can only be applied at startup so in order to make the changes effective, they must be saved with **saveenv** and the Chassis Manager restarted, either using the reboot command or cycling power.

Example 1: Readout of the IP address

```
chm>lanconfig ip
IP=193.155.166.51
```

Example 2: Changing the IP address

```
chm>lanconfig ip 196.100.100.1
IP=196.100.100.1
```

Example 3: Enable dhcp

```
chm>lanconfig dhcp on
DHCP on
```

10.13 logout command

Syntax: logout

Function:

Logs out the current user and permits a new log in.

10.14 nvm command

Syntax:

nvm [destination (embedded) [(protect|unprotect) identification |configuration | log]] | [category (identification|configuration|log) [(protect|unprotect) (embedded)]]

Function: Get or set the local NVM write protect configuration which is used when NVMRO is not protecting the global memories.

The **nvm** command allows the memory to be protected based on destination: embedded, local, remote or by category: identification, configuration, log. The current IPMC implementation has only embedded memory, therefore only embedded keyword is shown.

Example:

```
chm>nvm destination embedded
Write protected categories: none
Write unprotected categories: identification, configuration, logs
```

```
chm>nvm category configuration protect embedded
Done!
```

This command variant protects the configuration category located inside embedded memory.

10.15 payload_reset

Syntax: payload_reset [<timeout_tensofms> | assert | deassert]

Function: Asserts/deasserts the payload reset signal, keeps it active for the time value entered as a parameter and then toggles it.

10.16 payload_signals

Syntax: payload_signals

Function: Displays the status for the payload signals.

Example:

```
chm>payload_signals
Payload Signals status:
```

```
NVMRO: Deasserted / Pin: Deasserted
GDiscrete1: Deasserted / Pin: Not Available
SHDN_REQ#: Deasserted
SHDN_RDY#: Deasserted
PP_OFF#: Deasserted
PowerGood: Asserted
PP_RST#: Deasserted
```

10.17 reboot command

Syntax: `reboot [-b]`

Function:

Restarts the Chassis Manager. If parameter “-b” is added, Chassis Manager will be rebooted in bootloader mode.

10.18 saveenv command

Syntax: `saveenv`

Function:

Saves the parameters that were changed. If the modified parameters are not saved, they will be lost at the next reboot.

10.19 sel command

Syntax:

Linear log: `sel clr | count | print [start_record_no [end_record_no]]`

Circular log: `sel (ageing en| di) | clr | info | (print [(startup [length]) | (index [length])])`

Functions:

The syntax of the command is different depending on the type of log implemented: linear log on older firmware releases and legacy distributions or circular log on newer releases of firmware.

The circular log has an ageing feature that prevents the sel to fill up and stop recording new events. Once the sel is full, if ageing is enabled, the oldest events are deleted in order to make room for new ones.

Linear SEL

The command can display the number of sel entries, display a particular set of these entries or clear the sel.

To print the whole log use: **sel print**

To print all the records starting with a particular one use: **sel print record_no**

To print all records in a given interval: **sel print start_record_no end_record_no**

Example 1: Readout of SEL

```
chm>sel print
```

```
-----
```

Sensor Event Log										
Rec.ID.	dd.mm.yyyy	hh:mm:ss	Owner	Sensor	Name	Dir	Data1	Data2	Data3	
			Id	LUN	No. type		St	PrSt	Fru	Id
					0xF0		Dir	Thr	Value	Thr_Value
					Thr					
19404	01.01.2012	02:28:04	0xF0	0	1	0xF1	System	IPMB	As	IPMB-A enabled, IPMB-B enabled
19405	01.01.2012	02:28:05	0xF0	0	0	0xF0	FRU State	As	FRU#0	M1<-M0 Inactive
19406	01.01.2012	02:28:05	0xF0	0	99	0xF0	FRU 1 State	As	FRU#1	M1<-M0 Inactive
19407	01.01.2012	02:28:05	0xF0	0	67	0xC0	NVMRO	As	State	Asserted
19408	01.01.2012	02:28:05	0xF0	0	0	0xF0	FRU State	As	FRU#0	M4<-M1 Active
19409	01.01.2012	02:28:05	0xF0	0	17	0x02	+1.5V [ETH SW]	As	LC	0.59 V
19410	01.01.2012	02:28:05	0xF0	0	18	0x02	+3.3V [ETH SW]	As	LC	0.55 V
19411	01.01.2012	02:28:05	0xF0	0	19	0x02	+2.5V [GBIC-2]	As	UC	4.66 V
19412	01.01.2012	02:28:05	0xF0	0	20	0x02	+2.5V [GBIC-0/1]	As	LC	1.94 V
19413	01.01.2012	02:28:05	0xF0	0	21	0x02	+3.3V_CTRL/AUX	As	LC	1.84 V
19414	01.01.2012	02:28:05	0xF0	0	22	0x02	+12V_PRIM/AUX	As	LC	6.11 V
19415	01.01.2012	02:28:05	0xF0	0	23	0x02	+5V_PRIM	As	LC	2.94 V
19416	01.01.2012	02:28:05	0xF0	0	3	0x02	FRU Voltage	De	Limit	Not Exceeded
19417	01.01.2012	02:28:05	0xF0	0	3	0x02	FRU Voltage	As	Limit	Exceeded
19418	01.01.2012	02:28:05	0xF0	0	99	0xF0	FRU 1 State	As	FRU#1	M4<-M1 Active
19419	01.01.2012	02:28:05	0xF0	0	95	0xC1	PWR2_ENABLE#	As	State	Asserted
19420	01.01.2012	02:28:06	0xF0	0	37	0x04	Fan1	As	LC	0 RPM
19421	01.01.2012	02:28:06	0xF0	0	38	0x04	Fan2	As	LC	0 RPM

```
-----
```

```
19422 01.01.2012 02:28:07 0xF0 0 1 0xF1 System IPMB De IPMB-A enabled, IPMB-B disabled
19423 01.01.2012 02:28:07 0xF0 0 1 0xF1 System IPMB As IPMB-A enabled, IPMB-B enabled
```

Circular SEL

The command can display the number of sel entries, display a particular set of these entries or clear the sel.

To print all the events of the current startup use : ***sel print startup***

To print a number of records starting with a particular one use : ***sel print record_no no_of_records***

Example 2: SEL clearing

```
chm>sel clr
Done! Sel is empty!
```

10.20 sendipmb command

Syntax: **sendipmb** <slave_addr> netFn_Lun cmd [<data>]

Function:

Sends a raw IPMI commands over IPMB to the specified *slave_addr* slave address.

Example 1: Get device id request to IPMC 0x82

```
chm> sendipmb 0x82 0x18 0x01
```

10.21 sensor command

Syntax: **sensor** [<sensor_number> **set** <value>] | [<ipmb_addr>]

Function:

Parses sdr info for all the sensors, or a particular one on a IPMC specified by its IPMB address (<*ipmb_addr*>) and displays info. The command displays sensor information in a user friendly manner.

For system debug purposes a sensor value can be manually assigned to test functionality by using the **set** parameter; ex: **sensor 16 set 3.3** – for analog sensor 16 (voltage sensor), the ADC read value will be overwritten with 3.3 (a Chassis Manager reboot is necessary in order to reverse this effect)

10.22 settings command

Syntax: **settings** [tier [1|2]] | [payload_sd [en|di]] | [default_act [0|1]] | [deact_ignored [0|1]] [ipmb_speed [100|400]]

Function: displays or changes the current IPMC settings

tier: IPMC Tier Level. The IPMC could be configured to operate as either Tier1 or Tier2

payload_sd: if this process is enabled, the IPMC will negotiate the shutdown with the payload before turning off the power

default_act: non-volatile value for the Default Activation Locked bit. If default activation is disabled, the board will stay in M1 (payload power OFF) until it is enabled by Chassis Manager.

deact_ignored: non-volatile value for the Deactivation Ignored bit. If this bit is set a deactivation command is ignored.

ipmb_speed: The speed of the IPMC could be configured as either 100KHz or 400KHz

Example:

```
chm>settings tier 1
Done!
chm>settings default_act 1
Done!
```

10.23 uptime command

Syntax: uptime

Function:

Displays the amount of time past since the last ChMC power up/reboot.

10.24 user command

Syntax: user [*<id>* [(enable|disable) | (username *<new_name>*) | (password *<new_password>*)]]

Function: Displays information about supported users, change usernames and passwords

10.25 version command

Syntax: version

Function:


Displays information about the Chassis Manager: Part Number, Software Version, MAC Address and Serial Number.

10.26 xmodem command

Syntax: xmodem chassisfru | chmcfriu | conditions | firmware | sdr

Functions:

Sends via RS232 a fru file for the chassis, or the Chassis Manager, a condition file, a firmware image or a configuration file. After the command is entered, the Chassis Manager goes into data receive mode and waits for the data to be sent. You can then start the file transfer with your terminal program and select XMODEM as the protocol.

 When using "xmodem" in "Hyperterminal" the transfer of the desired file can take up to 10 seconds to start.
Only the admin can use the xmodem command.

Example 1: Sending of the config file

```
chm>xmodem sdr
Please upload the file...
chm>...Done!
```

11 Update protocol

All update procedures are accomplished by uploading specific files to the Chassis Manager. The files can be sent over the RS232 interface using the xmodem protocol, or over Ethernet using ipmitool.

11.1 Update over RS232 using xmodem

For connecting to the RS232 Command Line Interface (CLI) you will have to use a terminal program. On Windows OS we recommend the use of **Tera Term**.

By default the terminal settings are: 115200 bits per second, data bits: 8, parity: none, stop bit: 1

The **Port** setting should describe the physical port of the computer on which the Chassis Manager is connected.

The default Baud rate is 115200, but it can be changed to: 9600, 19200,38400 or 115200.

At start-up the Command Line Interface (CLI) of the Chassis Manager will request a login.
The default login data for the admin account:

login: **admin**
password: **ADMIN**

If the RS232 connection is used, the files are uploaded using the xmodem protocol. The update is started by using the CLI command "**xmodem**" with the parameter that corresponds to the file type uploaded.

Main Firmware CLI:

chm> xmodem chassisfru | chmcfriu | conditions | firmware | sdr

Bootloader CLI:

To access the bootloader, from the main firmware execute "**reboot -b**" command. The Chassis Manager will reboot; stop the bootloader by pressing 'x' before the bootloader countdown expires.

#xmodem firmware

After typing the command you will be prompted to upload the file. At this point you have to send the file using your terminal program.

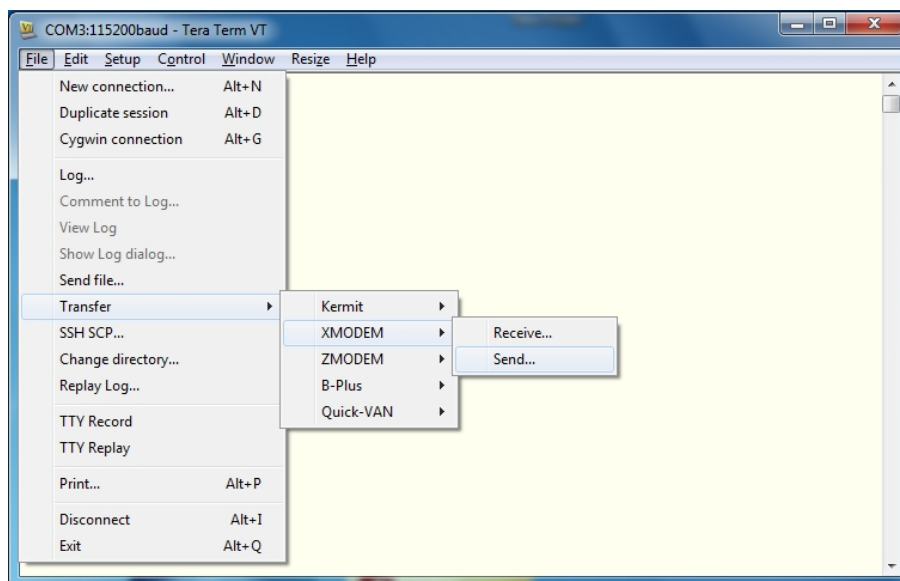


Figure 4: Sending files via xmodem using TeraTerm

After the transfer is complete a confirmation message is displayed: **chm>Done!**

***Note:**For boards running bootloader versions older than Rev 1.00 b 6 the *.firm file must be selected within 15 seconds from the moment xmodem firmware command is sent, otherwise the update process won't start. From Rev 1.00 b 6 and beyond the time was increased to 1 minute.

11.2 Updating the firmware

11.2.1 Firmware update via RS232 CLI

For updating the firmware over RS232 the Bootloader framework will be used. The update procedure is:

1. Connect to the RS232 interface using a terminal program.
2. Login to gain full access
3. Issue “**reboot -b**” command
4. Stop the bootloader at startup by pressing “x” before the timeout expired (3 seconds).
5. Use the *xmodem* command and the *firmware* parameter. In the bootloader the cursor is different than the one used in the main firmware: # instead of chm>. The bootloader xmodem command allows only the firmware parameter.

#xmodem firmware

***There are boards that come without any firmware loaded; in this case the board will start in bootloader mode by default.**

6. Upload the firmware image: **.firm/*.bin*
7. After the transfer is complete a confirmation message is displayed
8. Reboot: **#reboot**

11.3 Updating the configuration files

11.3.1 SDR file update via RS232

The RS232 SDR file update procedure is:

1. Connect to the RS232 interface using a terminal program.
2. Log in using the admin profile.
3. Use the *xmodem* command and the *sdr* parameter.
chm>xmodem sdr
4. Upload the SDR file: **.sdr*
5. After the transfer is complete a confirmation message is displayed
6. The configuration will be updated at the next start-up. To perform a restart the *reboot* command can be used: **chm>reboot.**

11.3.2 SDR file update via ipmitool

The ipmitool SDR file update procedure is:

1. Connect the chassis manager to the ethernet via the ETH port on the front panel.
2. Connect to the RS232 interface using a terminal program.
3. Use the *lanconfig* command to determine the chassis manager IP address.

```
chm>lanconfig
```

```
IP . . . 192.168.1.107
```

```
Mask . . 255.255.255.0
```

```
gateway. 192.168.1.1
```

```
chm>
```

4. Open **ipmitool** and send the following command:

```
ipmitool -H <ChassisManager_IP_Address> -U admin -P <Password_For_admin> -t 0xF0 sdr fill file "<SDR_File_Path>"
```

Example: `ipmitool -H 192.168.1.107 -U admin -P ADMIN -t 0xF0 sdr fill file <sdr_filename>`

5. Log in using the admin profile.
6. The configuration will be updated at the next start-up. To perform a restart the *reboot* command can be used: `chm>reboot`.

11.3.3 FRU file update via RS232

The RS232 FRU file update procedure is:

1. Connect to the RS232 interface using a terminal program.
2. Log in using the admin profile.
3. Use the *xmodem* command and the *chmcfru* parameter.
chm>xmodem chmcfru
4. Upload the FRU configuration file: **.fru*
5. After the transfer is complete a confirmation message is displayed
6. The configuration will be updated at the next start-up. To perform a restart the *reboot* command can be used: `%>reboot`.

11.3.4 FRU file update via ipmitool

The ipmitool FRU file update procedure is:

1. Connect the chassis manager to the ethernet via the ETH port on the front panel.
2. Connect to the RS232 interface using a terminal program.
3. Use the *lanconfig* command to determine the chassis manager IP address.

```
chm>lanconfig
```

```
IP . . . 192.168.1.107
Mask . . 255.255.255.0
gateway. 192.168.1.1
chm>
```

4. Open **ipmitool** and send the following command:

```
ipmitool -H <ChassisManager_IP_Address> -U admin -P <Password_For_admin> -t 0xF0 fru write 0 "<FRU_File_Path>"
```

Example: `ipmitool -H 192.168.1.107 -U admin -P ADMIN -t 0xF0 fru write 0 <FRU_File_Path>`

Expected answer:

```
Fru Size      : 3840 bytes
Size to Write : xxx bytes
```

***Response will differ based on actual file size.**

5. Log in using the admin profile.
6. The configuration will be updated at the next start-up. To perform a restart the *reboot* command can be used: `chm>reboot`.

11.3.5 Chassis FRU file update via RS232

The RS232 Chassis FRU file update procedure is:

1. Connect to the RS232 interface using a terminal program.

2. Log in using the admin profile.
3. Use the *xmodem* command and the *chassisfru* parameter.
chm>xmodem chassisfru
4. Upload the Chassis FRU configuration file: *.fru
5. After the transfer is complete a confirmation message is displayed
6. The configuration will be updated at the next start-up. To perform a restart the *reboot* command can be used: %>**reboot**.

11.3.6 Conditions file update via RS232

The RS232 conditions file update procedure is:

1. Connect to the RS232 interface using a terminal program.
2. Log in using the admin profile.
3. Use the *xmodem* command and the *conditions* parameter.
chm>xmodem conditions
4. Upload the conditions file: *.txt
5. After the transfer is complete a confirmation message is displayed.
6. The configuration will be updated at the next start-up. To perform a restart the *reboot* command can be used: %>**reboot**.

12 Electrical and Environmental Parameters

Power supply: 3.3VDC

Current consumption: 500mA

PWM: Open drain (100mA max). Pull-up required on fans.

Digital Input/Output signal level: Depending on VIO setting (3.3V or 5V)

Digital Output type: open drain with 10K pull-up to VIO.

Digital Output max loading: 25mA on single output but all 16 outputs shall not exceed 200mA.

Operating Temperature: -40..85 deg C

Storage Temperature: -40..85 deg C