Open CloudServer OCS Chassis Management Specification Version 2.0

Author:
Badriddine Khessib, Director of Systems Software Development, Microsoft

Revision History

Date	Description
10/30/2014	Version 2.0

Contents

Table of Figures

Table of Tables

1 Summary

This specification, *Open CloudServer Chassis Management Version 2.0*, describe systems management for the Open CloudServer (OCS) system. System management uses the Chassis Manager (CM) to present a consistent, optimized interface for the complete rack infrastructure, including the in-band and out-of-band (OOB) management paths.

The Chassis Manager provides the front end through an applications interface (RESTful web API) for automated management and through a command-line interface (CLI) for manual management. The Chassis Manager manages all devices within the rack and communicates directly with the blade management system through a serial multiplexor.

2 Chassis Manager

The Chassis Manager printed circuit board assembly (PCBA) is a general-purpose processing assembly integrated into the plenum of the chassis and attaching directly to the Power Distribution Backplane (PDB). The Chassis Manager communicates directly with the server blades and provides management for all devices within the rack, including power supplies and fans. Figure 1 shows a CAD representation of the Chassis Manager.

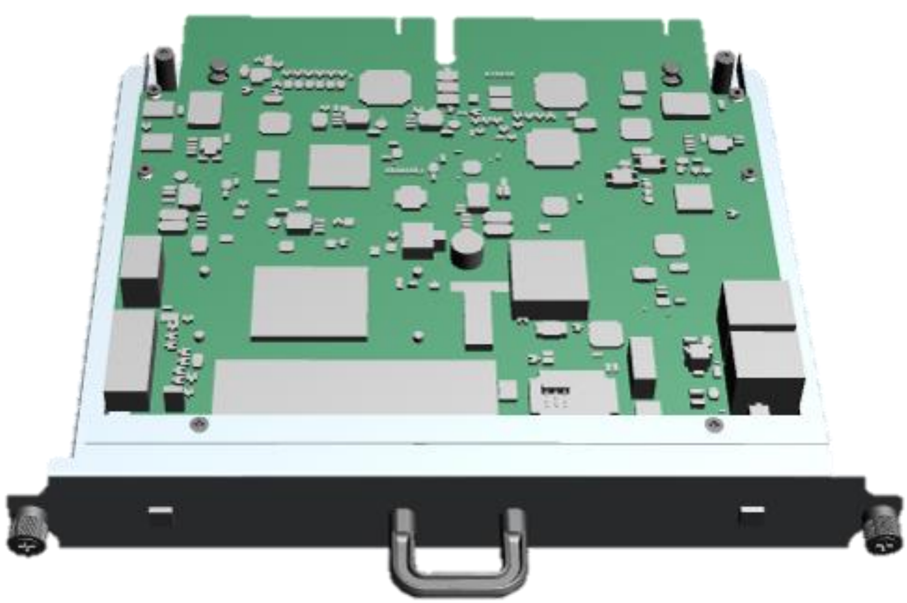


Figure 1: CAD representation of the Chassis Manager

2.1 Chassis Manager Features

The Chassis Manager features include:

- Input/Output (I/O):
  - 2 x 1GbE Ethernet (general purpose to be used for network access or for direct connector to the top-of-rack [TOR] management ports)
  - 4 x RS-232 (network switch management for boot strap initial start-up)
  - Remote power control (one input signal, three output signals to the power distribution unit [PDU] or to another Chassis Manager for remote power control)
- Windows Server 2012 R2 operating system
- Hot repair, no downtime during replacement
- Server hardware
  - Embedded x86 processor
  - Memory—4GB with error-correcting code (ECC)
  - Storage—64GB solid-state drive (SSD)
  - Trusted Platform Module—enabling a secure solution
  - Embedded serial multiplexor for hard-wired communication to blades
  - Blade power on/off signals hard-wired for definitive control of blade power

Figure 2 shows a top-level representation of the Chassis Manager, trays, power supply units, and fans.

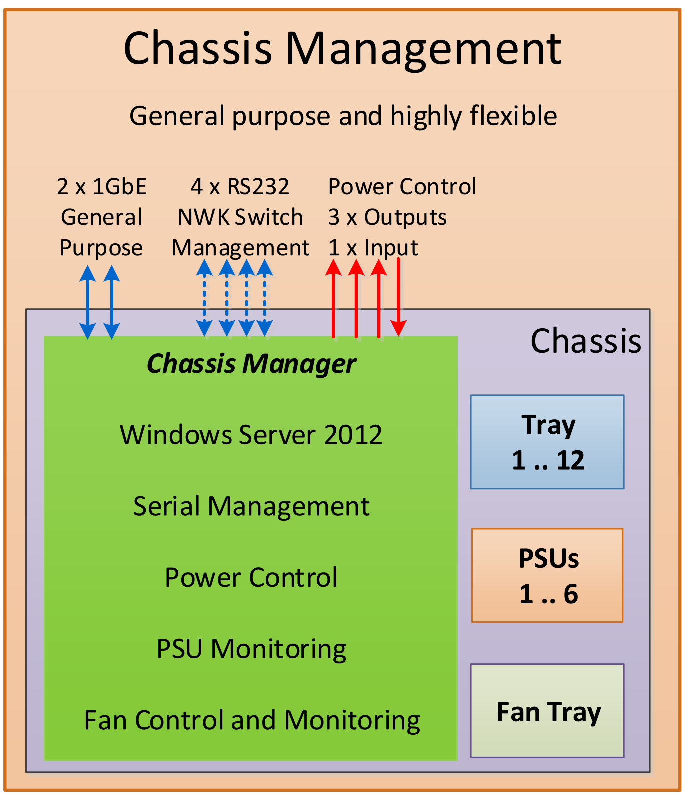


Figure 2: Top level representation of Chassis Manager

Figure 3 shows the hardware block diagram for the Chassis Manager.

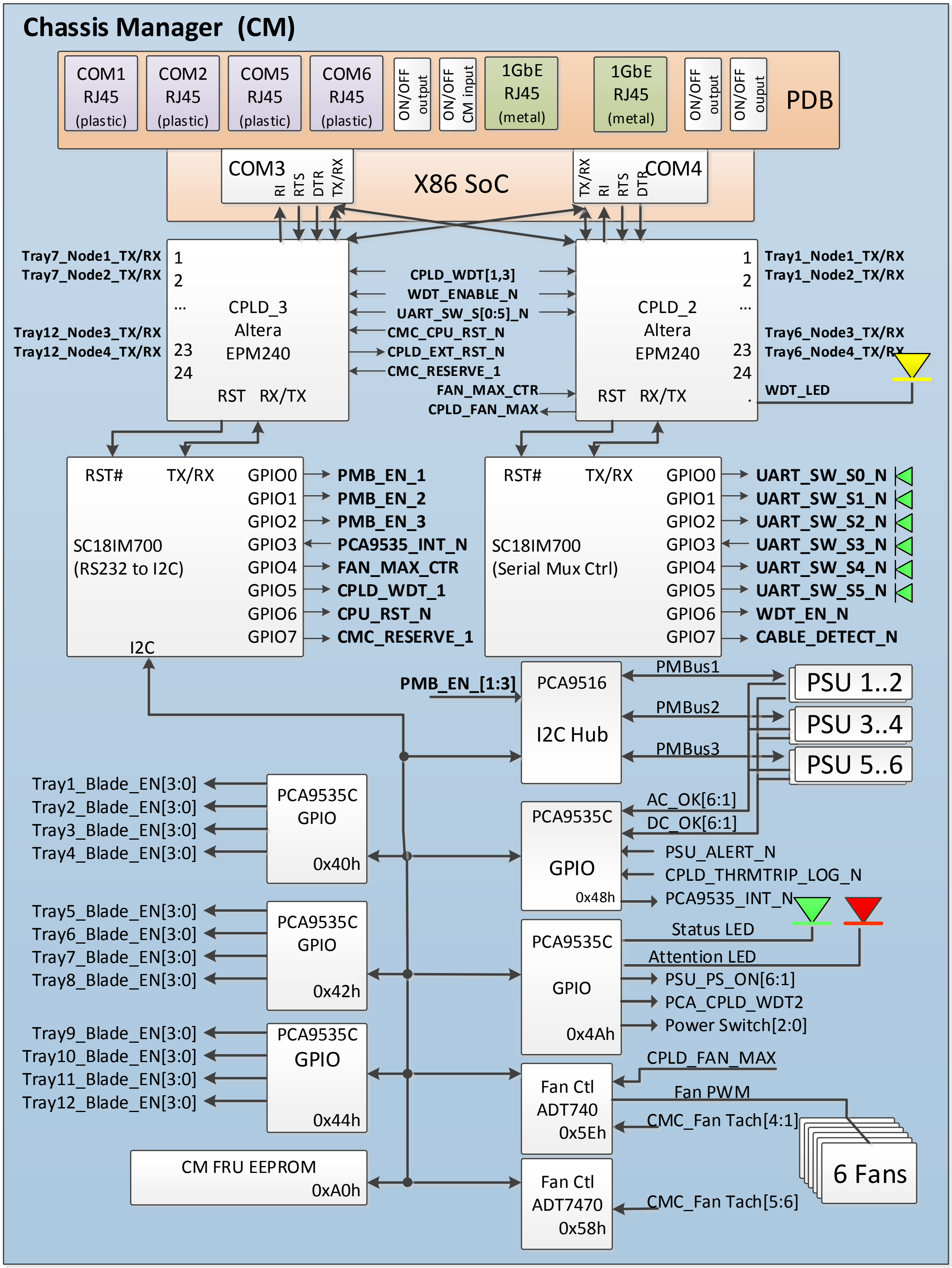


Figure 3: Chassis manager hardware block diagram

2.2 Signal Interface

The Chassis Manager interfaces to the power distribution board through two PCIe x16 connectors. Table 1 and Table 2 show the pinout information for these connectors. Refer to the schematics for connector reference designators.

Table 1 shows the pinout for the J18 PDB connector interface.

Table 1: PDB connector interface (J18)

Pin	Signal	Pin	Signal
B1	GND	A1	GND
B2	LAN2_P1_P	A2	GND
B3	LAN2_P1_N	A3	LAN2_P3_P
B4	GND	A4	LAN2_P3_N
B5	LAN2_P2_P	A5	GND
B6	LAN2_P2_N	A6	LAN2_P4_P
B7	GND	A7	LAN2_P4_N
B8	GND	A8	GND
B9	GND	A9	POWER_SW3_PWR_CTR_12V
B10	I2C_PDB_SCL	A10	POWER_SW2_PWR_CTR_12V
B11	I2C_PDB_SDA	A11	POWER_SW1_PWR_CTR_12V
B12	P3V3_NODE1	A12	NC
B13	NC	A13	NC
B14	NC	A14	NC
B15	NC	A15	NC
B16	NC	A16	NC
B17	NC	A17	GND
B18	GND	A18	UART_RTS_RP6_L_N
B19	UART_RTS_RP5_L_N	A19	UART_DSR_RP6_L_N
B20	UART_DSR_RP5_L	A20	UART_RX_RP6_L
B21	UART_RX_RP5_L	A21	UART_TX_RP6_L
B22	UART_TX_RP5_L	A22	UART_DTR_RP6_L_N
B23	UART_DTR_RP5_L_N	A23	UART_CTS_RP6_L
B24	UART_CTS_RP5_L_N	A24	UART_RI_RP6_L_N

Pin	Signal	Pin	Signal
B25	UART_RI_RP5_L_N	A25	GND
B26	GND	A26	FAN4_TACH
B27	FAN1_TACH	A27	FAN5_TACH
B28	FAN2_TACH	A28	FAN6_TACH
B29	FAN3_TACH	A29	GND
B30	GND	A30	FAN_PWM
B31	T12_NODE1_EN	A31	GND
B32	T12_NODE2_EN	A32	I2C_PSU3_SCL
B33	T12_NODE3_EN	A33	I2C_PSU3_SDA
B34	T12_NODE4_EN	A34	GND
B35	GND	A35	T11_NODE1_EN
B36	UART_T12_NODE1XD	A36	T11_NODE2_EN
B37	UART_T12_NODE1_TXD	A37	T11_NODE3_EN
B38	UART_T12_NODE2_RXD	A38	T11_NODE4_EN
B39	UART_T12_NODE2_TXD	A39	GND
B40	GND	A40	UART_T11_NODE1_RXD
B41	UART_T12_NODE3_RXD	A41	UART_T11_NODE1_TXD
B42	UART_T12_NODE3_TXD	A42	UART_T11_NODE2_RXD
B43	UART_T12_NODE4_RXD	A43	UART_T11_NODE2_TXD
B44	UART_T12_NODE4_TXD	A44	GND
B45	GND	A45	UART_T11_NODE3_RXD
B46	T10_NODE1_EN	A46	UART_T11_NODE3_TXD
B47	T10_NODE2_EN	A47	UART_T11_NODE4_RXD
B48	T10_NODE3_EN	A48	UART_T11_NODE4_TXD
B49	T10_NODE4_EN	A49	GND
B50	GND	A50	T9_NODE1_EN
B51	UART_T10_NODE1_RXD	A51	T9_NODE2_EN
B52	UART_T10_NODE1_TXD	A52	T9_NODE3_EN
B53	UART_T10_NODE2_RXD	A53	T9_NODE4_EN
B54	UART_T10_NODE2_TXD	A54	GND
B55	GND	A55	UART_T9_NODE1_RXD

Pin	Signal	Pin	Signal
B56	UART_T10_NODE3_RXD	A56	UART_T9_NODE1_TXD
B57	UART_T10_NODE3_TXD	A57	UART_T9_NODE2_RXD
B58	UART_T10_NODE4_RXD	A58	UART_T9_NODE2_TXD
B59	UART_T10_NODE4_TXD	A59	GND
B60	GND	A60	UART_T9_NODE3_RXD
B61	T8_NODE1_EN	A61	UART_T9_NODE3_TXD
B62	T8_NODE2_EN	A62	UART_T9_NODE4_RXD
B63	T8_NODE3_EN	A63	UART_T9_NODE4_TXD
B64	T8_NODE4_EN	A64	GND
B65	GND	A65	T7_NODE1_EN
B66	UART_T8_NODE1_RXD	A66	T7_NODE2_EN
B67	UART_T8_NODE1_TXD	A67	T7_NODE3_EN
B68	UART_T8_NODE2_RXD	A68	T7_NODE4_EN
B69	UART_T8_NODE2_TXD	A69	GND
B70	GND	A70	UART_T7_NODE1_RXD
B71	UART_T8_NODE3_RXD	A71	UART_T7_NODE1_TXD
B72	UART_T8_NODE3_TXD	A72	UART_T7_NODE2_RXD
B73	UART_T8_NODE4_RXD	A73	UART_T7_NODE2_TXD
B74	UART_T8_NODE4_TXD	A74	GND
B75	GND	A75	UART_T7_NODE3_RXD
B76	T6_NODE1_EN	A76	UART_T7_NODE3_TXD
B77	T6_NODE2_EN	A77	UART_T7_NODE4_RXD
B78	T6_NODE3_EN	A78	UART_T7_NODE4_TXD
B79	T6_NODE4_EN	A79	GND
B80	GND	A80	PSU_ALERT_R_N
B81	UART_T6_NODE1_RXD	A81	T5_NODE1_EN
B82	UART_T6_NODE1_TXD	A82	T5_NODE2_EN

Table 2 shows the pinout for the J35 PDB connector interface.

Table 2: PDB connector interface (J35)

Pin	Signal	Pin	Signal
B1	GND	A1	GND
B2	UART_T6_NODE2_RXD	A2	T5_NODE3_EN
B3	UART_T6_NODE2_TXD	A3	T5_NODE4_EN
B4	GND	A4	GND
B5	UART_T6_NODE3_RXD	A5	UART_T5_NODE1_RXD
B6	UART_T6_NODE3_TXD	A6	UART_T5_NODE1_TXD
B7	UART_T6_NODE4_RXD	A7	GND
B8	UART_T6_NODE4_TXD	A8	UART_T5_NODE2_RXD
B9	GND	A9	UART_T5_NODE2_TXD
B10	I2C_PSU2_SCL	A10	UART_T5_NODE3_RXD
B11	I2C_PSU2_SDA	A11	UART_T5_NODE3_TXD
B12	T4_NODE1_EN	A12	UART_T5_NODE4_RXD
B13	T4_NODE2_EN	A13	UART_T5_NODE4_TXD
B14	T4_NODE3_EN	A14	GND
B15	T4_NODE4_EN	A15	T3_NODE1_EN
B16	GND	A16	T3_NODE2_EN
B17	UART_T4_NODE1_RXD	A17	T3_NODE3_EN
B18	UART_T4_NODE1_TXD	A18	T3_NODE4_EN
B19	UART_T4_NODE2_RXD	A19	GND
B20	UART_T4_NODE2_TXD	A20	UART_T3_NODE1_RXD
B21	GND	A21	UART_T3_NODE1_TXD
B22	UART_T4_NODE3_RXD	A22	UART_T3_NODE2_RXD
B23	UART_T4_NODE3_TXD	A23	UART_T3_NODE2_TXD
B24	UART_T4_NODE4_RXD	A24	GND
B25	UART_T4_NODE4_TXD	A25	UART_T3_NODE3_RXD
B26	GND	A26	UART_T3_NODE3_TXD
B27	T2_NODE1_EN	A27	UART_T3_NODE4_RXD
B28	T2_NODE2_EN	A28	UART_T3_NODE4_TXD
B29	T2_NODE3_EN	A29	GND
B30	T2_NODE4_EN	A30	T1_NODE1_EN

Pin	Signal	Pin	Signal
B31	GND	A31	T1_NODE2_EN
B32	UART_T2_NODE1_RXD	A32	T1_NODE3_EN
B33	UART_T2_NODE1_TXD	A33	T1_NODE4_EN
B34	UART_T2_NODE2_RXD	A34	GND
B35	UART_T2_NODE2_TXD	A35	UART_T1_NODE2_RXD
B36	GND	A36	UART_T1_NODE2_TXD
B37	UART_T2_NODE3_RXD	A37	UART_T1_NODE3_RXD
B38	UART_T2_NODE3_TXD	A38	UART_T1_NODE3_TXD
B39	UART_T2_NODE4_RXD	A39	GND
B40	UART_T2_NODE4_TXD	A40	UART_T1_NODE4_RXD
B41	GND	A41	UART_T1_NODE4_TXD
B42	I2C_PSU1_SCL	A42	UART_T1_NODE1_RXD
B43	I2C_PSU1_SDA	A43	UART_T1_NODE1_TXD
B44	GND	A44	GND
B45	PSU1_AC_OK	A45	PSU1_DC_OK
B46	PSU2_AC_OK	A46	PSU2_DC_OK
B47	PSU3_AC_OK	A47	PSU3_DC_OK
B48	PSU4_AC_OK	A48	PSU4_DC_OK
B49	PSU5_AC_OK	A49	PSU5_DC_OK
B50	PSU6_AC_OK	A50	PSU6_DC_OK
B51	GND	A51	GND
B52	UART_RTS_P5_L_N	A52	UART_RTS_P6_L_N
B53	UART_DSR_P5_L_N	A53	UART_DSR_P6_L_N
B54	UART_RX_P5_L	A54	UART_RX_P6_L
B55	UART_TX_P5_L	A55	UART_TX_P6_L
B56	UART_DTR_P5_L_N	A56	UART_DTR_P6_L_N
B57	UART_CTS_P5_L_N	A57	UART_CTS_P6_L_N
B58	UART_RI_P5_L_N	A58	UART_RI_P6_L_N
B59	GND	A59	GND
B60	UART_RTS_P1_L_N	A60	UART_RTS_P2_L_N

Pin	Signal	Pin	Signal
B61	UART_DSR_P1_L_N	A61	UART_DSR_P2_L_N
B62	UART_RX_P1_L	A62	UART_RX_P2_L
B63	UART_TX_P1_L	A63	UART_TX_P2_L
B64	UART_DTR_P1_L_N	A64	UART_DTR_P2_L_N
B65	UART_CTS_P1_L_N	A65	UART_CTS_P2_L_N
B66	GND	A66	GND
B67	NC	A67	NC
B68	NC	A68	NC
B69	NC	A69	NC
B70	NC	A70	NC
B71	GND	A71	GND
B72	LAN1_P3_P	A72	LAN1_P1_P
B73	LAN1_P3_N	A73	LAN1_P1_N
B74	GND	A74	GND
B75	LAN1_P4_P	A75	LAN1_P2_P
B76	LAN1_P4_N	A76	LAN1_P2_N
B77	GND	A77	GND
B78	CM_PWR_CTL_IN	A78	P12V_PDB
B79	P12V_PDB	A79	P12V_PDB
B80	P12V_PDB	A80	P12V_PDB
B81	MATE_R_N	A81	P12V_PDB
B82	P12V_PDB	A82	P12V_PDB

2.3 Blade Management

The Chassis Manager is connected to each blade via two blade enable signals and two serial connections. Figure 4 shows the blade management connectivity from the Chassis Manager to the blades via the power distribution board and tray backplane.

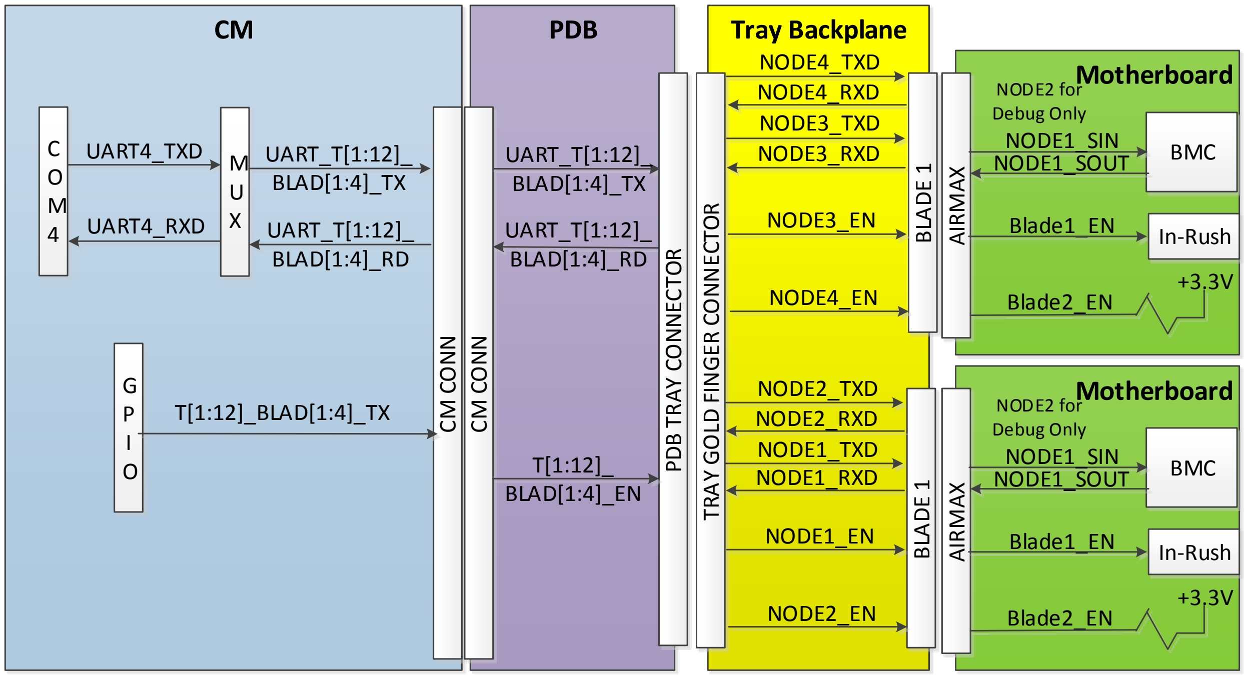


Figure 4: Blade management connectivity

2.4 Power Control

The Chassis Manager provides controls for power of the blades and remote devices. At web-scale, definitive control over power is necessary to provide a clean reset in the event of locked-up or hung circuits.

2.4.1 Blade Power Control

The blade enable signals indicate on/off to the in-rush controller on the blade to emulate a full power off.

The blade enable signals are implemented via General Purpose I/O (GPIO) registers. The default is to float high via pull-ups on the blade, so that when the Chassis Manager is removed or when the power is cycled for service, blade operation is not disrupted.

The signals are grouped to make it possible for all blades on a single tray to be powered on/off coincidently, and to let blades on different trays be powered on/off either coincidently or through timed power control.

Timed power control can be used to control servers and JBODs (for example, to prevent incorrect errors from being reported when using a head server node and JBOD blades). The following steps can be used:

1. Power off the head (server) node, and allow a short delay.

2. Power off the JBOD blade, and allow a five second delay.
3. Power on the JBOD blade, and allow a short delay.
4. Power on the head (server) node.

Timed delays can be easily adjusted to match the needs of the hardware and storage though the Chassis Manager.

2.4.2 Remote Power Control

It is important to have full power control over remote devices such as network switches or other Chassis Managers.

The remote power controls are 12V on/off signals. The default is 0V (ON). Off is 12V, so that if a Chassis Manager is removed or if the power is cycled, remote device operation is not disrupted.

The individual signals are:

- ON/OFF input
  Accepts the 12V signal from a remote Chassis Manager for power control
- Three ON/OFF outputs
  Allows control of remote devices

2.5 Communication Ports

Table 3 lists the definition and translation of the serial COM port signals of the RJ-45 connector for ports 1 and 2 (see Figure 3).

These definitions match the functionality of switch consoles that use RJ-45, but can be used to select cables that convert from DB-9 or DB-25 connections. The system swaps received data/transmitted data (RxD/TxD), clear to send/request to send (CTS/RTS), and data terminal ready/data set ready (DTR/DSR) signals so a straight Ethernet cable can be used.

Table 3: Serial RJ-45 cable definition for ports 1 and 2

	PDB connection ports 1 and 2	**Typical switch management port**	
Signal on CM board	**RJ-45 connector pin**	**RJ-45 connector pin**	**Switch console port**
RTS	1	1	CTS
DSR	2	2	DTR
RxD	3	3	TxD
GND	4	4	GND
GND	5	5	GND

	PDB connection ports 1 and 2	Typical switch management port	
Signal on CM board	**RJ-45 connector pin**	**RJ-45 connector pin**	**Switch console port**
TxD	6	6	RxD
DTR	7	7	DSR
CTS	8	8	RTS

Table 4 lists the definition and translation of the serial COM port signals of the RJ-45 connector for serial COM ports 5 and 6 (see Figure 3). COM port 3 and 4 are internal to the Chassis Manager board.

Table 4: Serial RJ-45 cable definition for ports 5 and 6

	PDB connection ports 5 and 6	Typical switch management port	
Signal on CM board	**RJ-45 connector pin**	**RJ-45 connector pin**	**Switch console port**
RTS	1	1	CTS
DSR	2	2	DTR
RxD	3	3	TxD
RI	4	4	GND
GND	5	5	GND
TxD	6	6	RxD
DTR	7	7	DSR
CTS	8	8	RTS

2.6 Mechanical Specifications

Figure 5 shows the mechanical control outline for the Chassis Manager. Also shown (circled in red) are the edge fingers that provide connectivity to the power distribution board.

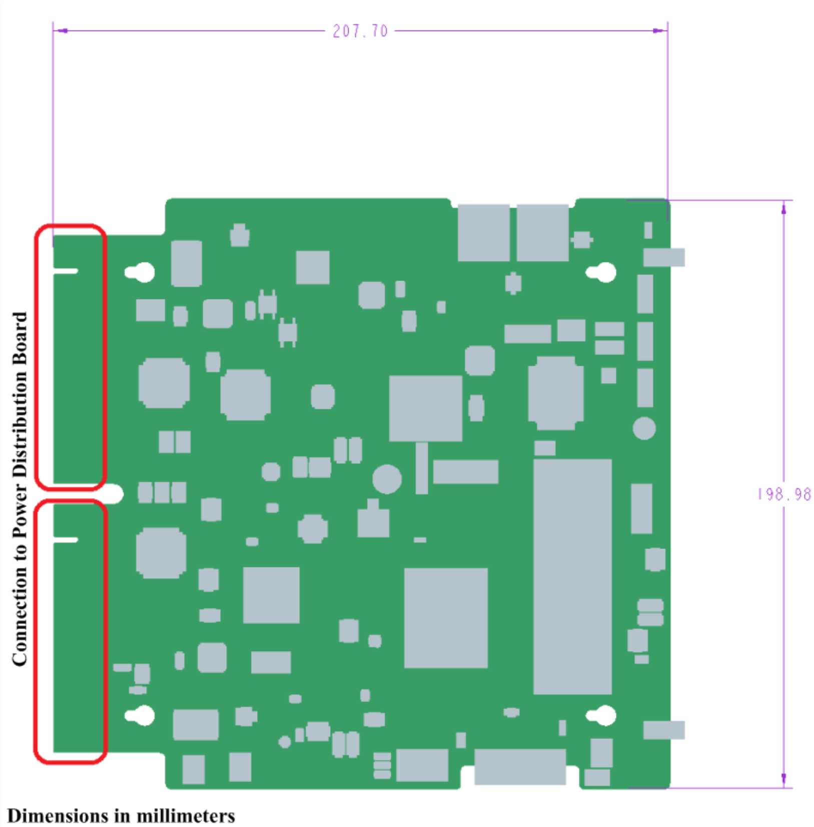


Figure 5: Chassis manager dimensions and edge finger locations

2.7 Chassis LEDs

Each chassis has two light-emitting diodes (LEDs) on the Chassis Manager: a health status LED that is green and an attention LED that is red. Both LEDs are driven by a single GPIO bit off the blade's management i2c tree.

2.7.1 Chassis Health Status LED

The chassis health status LED indicates whether the Chassis Manager has booted. Note that if the 12V power is off, the LEDs on the power supplies are also off. Table 5 describes the operation of the chassis health status LED.

Table 5: Chassis health status LED description

LED Status	Condition
Off	Backplane 12V power is off if power supply LEDs are off Backplane 12V power is on, but Chassis Manager never booted if power supply LEDs are on
Solid green on	Backplane 12V power is on and Chassis Manager has booted

2.7.2 Chassis Attention LED

The chassis attention LED is visible from the rear of the chassis without opening the fan tray. This LED directs service technicians to the correct chassis during repair. When possible, blade diagnostics are used to direct repairs; alternately, the scale-out management software can be used. In both cases, logs of the repair work are available.

The chassis attention LED indicates the following conditions:

- **Operator directed**
  An operator can manually set the chassis attention LED (for example, identification of chassis cables)
- **Power supply failure**
  Chassis Manager has detected a power supply failure
- **Fan failure**
  Chassis Manager has detected a fan failure

Note that the chassis attention LED must be turned off after service is complete.

Table 6 describes the operation of the chassis attention LED.

Table 6: Chassis Attention LED description

LED Status	Condition
Off	No attention indicated
Solid red	Operator directed Power supply or fan failure

3 Systems Management Operations

The following sections describe the system management operations for the rack infrastructure, and for the in-band and out-of-band (OOB) management paths.

Systems management is designed to present a consistent, optimized interface. The Chassis Manager provides the front end through an applications interface (RESTful web API) for automated management and a CLI for manual management. The Chassis Manager manages all devices within the rack and communicates directly with the blade management system through a serial multiplexor.

There are two possible paths for systems management: in-band and out-of-band.

- The in-band management path is through the primary network interface card (NIC) while the operating system is running.
- The out-of-band path is through the Chassis Manager.

In-band is the preferred path for systems management whenever possible.

3.1 Rack and Chassis Manager Commands

The Chassis Manager is responsible for managing the blades and the infrastructure within the rack. It monitors the health of the power supplies and the fans and sets the fan speeds. Because the Chassis Manager is the gateway into the system, it also gathers all information necessary to perform wiring checks by identifying the nodes in each slot and identifying their media access control (MAC) addresses, the switches, and the cable MAC addresses.

Table 7 lists the functionality of the commands that apply to the rack infrastructure.

Table 7: Rack infrastructure command functionality

Requirement	Details
Switch power control (complete ON/OFF)	Cycle the power to the switch (two 12V signals are driven by the Chassis Manager)
Rack identification	Provides an inventory of rack hardware: • Chassis manager information • Chassis numbers installed (part number, serial number) • Power supply unit and fan status
Blade identification	For every blade slot in every chassis: • Present • Type • Location and network connections • Network MAC addresses • Asset information (such as globally unique identifier [GUID]), device type, and versions) • Fan cubic feet per minute (CFM) requirement for each blade
Network identification	For every network switch in the rack managed by the CM: • Status • GUID • Status and MAC addresses connected to each port

Requirement	Details
Power supply unit (PSU), fan status	• PSU DC-OK • AC-OK • Fan tachometer • Fan setting
Fan speed control	• Required fan speed
Chassis power consumption	For use with advanced power capping, integration with rack-level uninterrupted power supply (UPS), high temperature operation
Manageability firmware update	• To update microcontrollers within the chassis through the CM • To update PSU and fan control firmware, if applicable
User management	• Security privilege levels for users connected to the CM (for example, not all users logged into the CM will be able control fan speed or powering blades ON/OFF) • Security privilege mechanisms are still TBD

3.2 In-Band Management Commands

The primary method for managing servers is in-band through the operating system. The system can use a unified extensible firmware interface (UEFI) that allows more functionality through the primary NIC, though this is not considered to be in-band and is not mandatory for interoperability. UEFI can initialize the NIC very early in the boot sequence, and can implement serial-over-local area network (SOL) to complement Windows 8 SOL support for system debug.

The path through the operating system uses the native Windows intelligent platform management interface (IPMI) driver and native IPMI Windows management instrumentation (WMI) provider. The baseboard management controller (BMC) firmware must be compatible with the native Windows IPMI keyboard controller style (KCS) interface driver.

A number of functions are implemented with a utility that is traditionally run under the operating system; these utilities could be run under UEFI (method to be determined).

Table 8 lists command functionality that is supported by the in-band path. Note that it is assumed that a hardware monitoring device (HMD) will be present on the motherboard to provide management functionality.

Table 8: Command functionality supported by in-band path

Function	Method	Function details
Identification	WMI	• HMD ID/version information • System GUID • HMD MAC address (only if sideband LAN access is provided) • Server MAC address • Asset tag
Chassis power	WMI	• Power ON • Power OFF • Warm reset
Console	Windows 8 and UEFI SOL, Chassis Manager	• Serial console
Basic input/output system (BIOS) firmware update	Utility	• Motherboard flash
Manageability firmware update	Utility	• Motherboard flash
Host configuration BIOS settings	Utility	• Boot order • Power policy • Real-time clock
Event logging	WMI	• Get log in IPMI SEL format • Clear log • Logs for AP/FC remote management agent (RMA) diagnosis and ticketing
Temperature monitoring	WMI	• Inlet • Exhaust • CPU temperature
Power management	WMI	• Set power limit • Get power limit • Get power reading
Performance/power state	WMI	• Power capping

Function	Method	Function details
Failure and wear indicators	WMI	Includes HDD and SSD self-monitoring, analysis and reporting technology (SMART) data, memory error logs, and more

3.3 Out-of-Band Management Commands

A few server management functions use the out-of-band management path through the serial link that is connected to the Intel® Manageabilty Engine (ME) or HMD.

Table 9 lists the blade management command functionality that is supported by out-of-band path.

Table 9: Command functionality supported by out-of-band path

Function	Function details	Descriptions
Identification	• HMD MAC address • Server MAC address • Asset tag	• MAC addresses used for discovery and wiring checks • Asset tag is used for tax compliance audits
Blade node power	• Hard power ON • Hard power OFF	• Drives the Blade_EN to the in-rush controller to perform full power ON/OFF • OFF also turns off power to the Intel ME~~/HMD~~
Blade node power	• Soft power ON • Soft power OFF • Warm reset	• Under direction of the Intel ME ~~ME or HMD~~
Event logging	• Power status (ON/OFF/sleep) • Health status (healthy/error) • Fan/CFM request	• Provides at-a-glance status of server • Fan/CFM information is polled every 10 seconds

4 Chassis Manager Services

Figure 6 shows the services that the Chassis Manager provides.

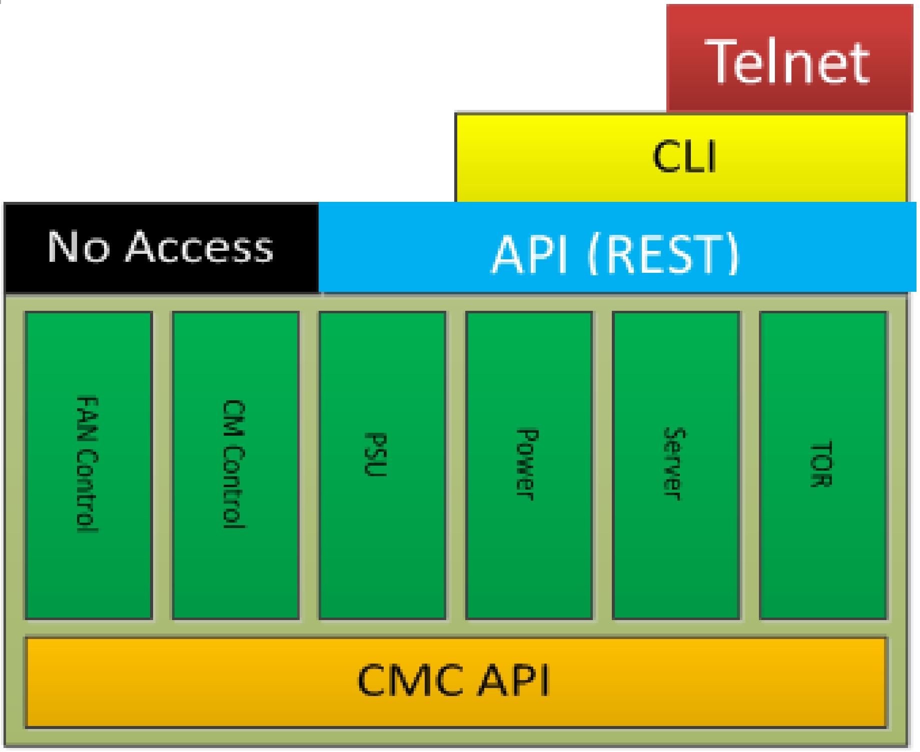


Figure 6: Chassis manager services

Table 10 lists the Chassis Manager services, and the sections that follow provide additional description.

Table 10: Chassis manager services

Chassis manager service	Description
Fan service	Controls the fan speed of all the fans housed in the chassis to keep the servers cool and operational Checks the status of every fan and alerts when a fan speed becomes unrealistic
PSU service	Reads the status of every power supply unit (PSU) in the chassis and sends an alert if a PSU goes down or malfunctions Reports power reading for every PSU
Power control service	Provides the service to power ON/OFF every blade in the chassis

Chassis manager service	Description
Blade management service	Provides the following blade management services: • Chassis power management (ON/OFF/reset) • Field replaceable unit (FRU) management • Sensor management • Serial console redirection • Blade ID (through LED)
Top-of-rack (TOR) service	Provides a serial connection to the TOR and acts as a gateway to all serial communication to the TOR
Security	Allows for creating users, deleting users, and updating user properties such as passwords
Chassis manager control services	Exposes commands to manage the Chassis Manager itself (for example, NIC settings of the NIC ports)

4.1 Fan Control Protocol

This section describes the fan control protocol, and provides an example for calculating the pulse-width modulation (PWM) sensor reading.

Table 11 lists the input and output for the fan control software.

Table 11: Input and output for fan control software

Description	
Input	• PWM sensor reading from each blade • Time period for sampling
Output	• Fan PWM for setting fan speed for the entire chassis

4.1.1 Determining Fan Speed

The Chassis Manager uses the following steps to determine the fan speed:

1. Get PWM sensor values from the blade (see the example in the section that follows).
   a. The IPMI **Get Sensor Reading** command is used to get the PWM sensor values from the blade. Note that the PWM sensor ID is 1. (This sensor should also be the first sensor enumerated in the IPMI sensor data record [SDR].)
   b. This sensor reading is sampled for each blade periodically; this time period is determined by the configuration parameter when the Chassis Manager starts (currently

this configuration parameter is set by default to 10 seconds, which means each blade is polled exactly every 10 seconds).

2. Set PWM value as fan speed.
   a. The Chassis Manager identifies the maximum value from all its constituent blade PWM values, and then sets the fan PWM to that value. Note that currently all the fans are set to the same speed.
3. Correct for altitude.
   a. The fan speed is corrected for altitude using linear interpolation (see the hardware specification for more detail).
   b. Current altitude is obtained from the configuration file; if no altitude is present, this correction is not performed.
4. Correct for one fan failure if necessary.
   a. If there is one fan failure, the PWM calculated from step 2 is scaled with the multiplier (maximum number of fans)/(maximum number of fans - 1).
   b. This linear scaling of fan PWM requested (within maximum limit of 100) is sufficient to handle one fan failure.
   c. The Chassis Manager also logs an error and sets the attention LED.
5. Correct for more than one fan failure if necessary.
   a. If there is more than one fan failure, the Chassis Manager sets the fan speed to maximum PWM (100). It also logs an error and sets the attention LED.

4.1.2 Sample PWM Calculation

Following is an example of calculating the blade PWM sensor ID. This value should be exposed as a logical IPMI sensor with a lower limit of 20 and an upper limit of 100.

The algorithm is based on a feedback control loop that checks every sensor on the list (specified as priority level 1 to N) against a target specified by the component manufacturer for maximum reliability. The algorithm then computes the relative PWM increase or decrease required and sends this value to the blade.

Table 12 lists the input and output for the main algorithm.

Table 12: Input and output for fan control software

Description	
Input	• Sensor priority • Sensor target
Output	• PWM value • Increase/decrease request

Following is the pseudocode for the process:

1. Create a sensor reading table T[1..N]. See Table 13 for the example.
2. For each sensor (with priority 1..N), run the following statements:

```
          If (sensor.currentValue != sensor.target)
               Difference = sensor.currentvalue - sensor.target
               PWMstep = ExponentialFunction(Step, Difference)
               Input PWMstep into table location T[sensor.priority]
          EndIf
     EndDo
```

3. Find the maximum value from table T[1..N], and supply the absolute PWM value and a code that specifies whether to decrease or increase the PWM (based on negative or positive difference value) as part of the response packet.

The function `ExponentialFunction(step, difference)` enables the decrease or increase in PWM based on distance from the target value. If the current value is very close to target value, the exponential function will return smaller step; if the current value is very far away from target value, the exponential function will return larger step. This keeps decay slow and controls linear swings between extremes. The exponential function is tuned specifically for each sensor.

If none of the sensor values are valid, then the appropriate PWM should be determined by the blade vendor (this could be maximum or minimum PWM based on thermal profiles). Note that if the current temperature is higher than the high temperature, the PWM value is reset to `Max PWM`.

The inlet temperature sensor is treated differently from the other sensors. It is used to define a base fan speed relative to the intake temperature to provide cooling for all components that are not directly monitored, and should ramp fan speeds up with inlet temperature increases as appropriate.

Table 13 shows an example of a sensor reading table.

Table 13: Example sensor readings

Sensor	Target	High critical	Priority
Inlet	30-55	38	1
CPU 2 (downstream)	74	90	2
CPU 1	74	90	3
HDD 3 (downstream)	40	60	4
HDD 4 (downstream)	40	60	5
HDD 1	40	60	6
HDD 2	40	60	7

Sensor	Target	High critical	Priority
PCH	90	95	8
DIMM 10 (downstream)	80	90	9
DIMM 11 (downstream)	80	90	10
DIMM 12 (downstream)	80	90	11
DIMM 13 (downstream)	80	90	12
DIMM 14 (downstream)	80	90	13
DIMM 15 (downstream)	80	90	14
DIMM 16 (downstream)	80	90	15
DIMM 9 (downstream)	80	90	16
DIMM 1	80	90	17
DIMM 2	80	90	18
DIMM 3	80	90	19
DIMM 4	80	90	20
DIMM 5	80	90	21
DIMM 6	80	90	22
DIMM 7	80	90	23
DIMM 8	80	90	24

The Chassis Manager polls all these PWM values from each individual blade, computes the maximum, and sets the fan speed accordingly.

4.2 Blade State Management

The Chassis Manager needs to keep track of the state of all its blades to discover new blades and to optimize the code behavior. For example, if only one out of N blades is powered on, the Chassis Manager should not try to get sensor readings from the rest of the blades because the sensor-read command is time consuming.

Figure 7 lists states and transitions for managing the blades. The Chassis Manager maintains these values for blades it controls.

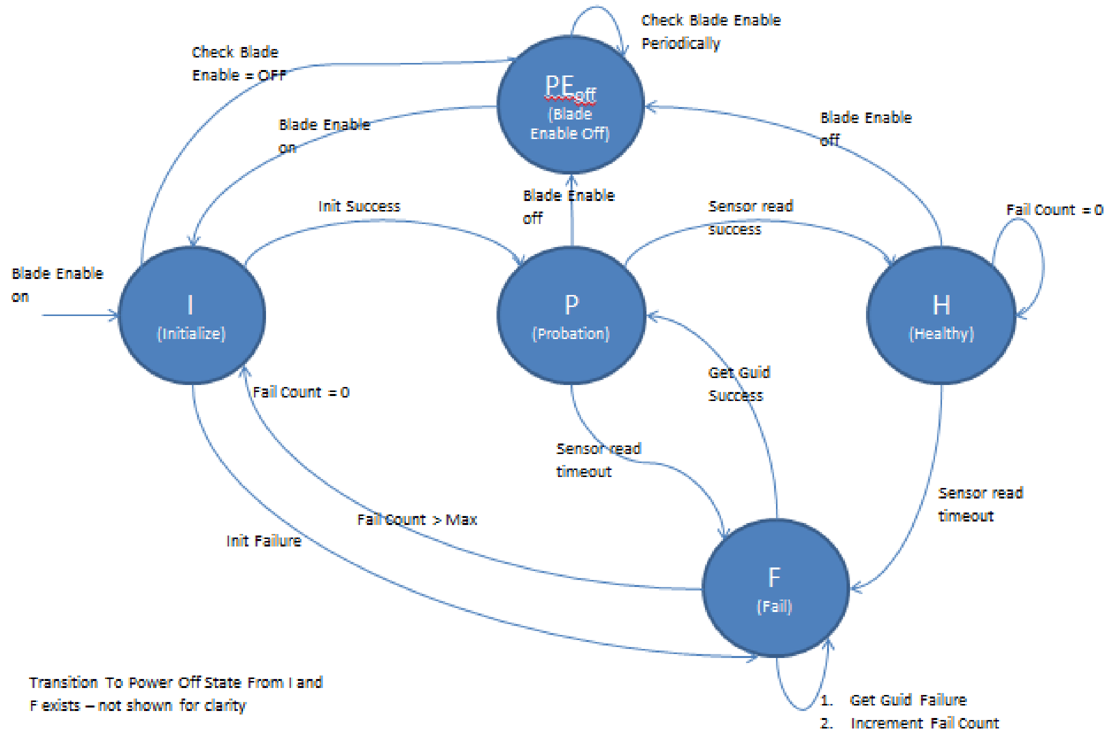


Figure 7: Blade state management

Note that there are only two user-facing commands that can change the state of the blade: power ON and power OFF (Blade Enable ON and Blade Enable OFF) commands. IPMI commands are not part of the blade state because they are served in operational states (probation and healthy states. See Table 14 for more detail.

Table 14: Blade states

Blade state	Description
Initialize (I)	Captures the blade initialization steps when Chassis Manager starts or when a new blade is inserted. Creates client objects and of obtains IPMI session authentication, the GUID of the blade, and the SDR, which supplies the low and high threshold values for the fan algorithm. Checks the power enable status of the blade and moves the blade to: • Blade Enable Off (PEoff) if the blade is set to OFF • Probation (P) if initialization succeeds • Fail (F) if initialization does not succeed
Blade Enable OFF (PEoff)	Captures the state where the blade enable is off. All commands fail in this state except power ON. Periodically checks the power enable state to see if it is still in Blade Enable OFF.

Blade state	Description
Probation (P)	Transitory state that logically separates the operational states, making it possible to use a light-weight "get GUID" command as the heartbeat in the fail state. Prevents the fail count from being reset (catching "Get GUID" success and read sensor failure event loops). Chassis manager tries to serve one sensor read request in this state; if that succeeds, it moves to the healthy state.
Healthy (H)	Most blades should be in this state; transition to fail only when all temperature sensor reads fail Keep serving IPMI requests
Fail (F)	Non-operational, cannot serve any requests Increment fail count every time state is encountered from outside or through self-loops At each iteration, try to get GUID with light-weight heartbeat: • If heartbeat succeeds and GUID has changed, re-initialize client • If heartbeat succeeds and GUID has not changed, blade moves to probation and back to healthy Note that if the fail count goes beyond a maximum (fail count > max), tries an initialization action for IPMI; this prevents infinite loops and discovers newly inserted blades

4.3 Chassis Manager Component Failure Scenarios

The Chassis Manager handles only fan failures:

- Single fan failure:
  The Chassis Manager logs an error, sets fan speed to 6/5, and turns on the attention LED.
- Two or more fan failures:
  The Chassis Manager logs an error, sets fan speed to high, and turns on the attention LED.

For all other failures, the Chassis Manager only logs an error and turns on the attention LED; by design, the Chassis Manager takes no pre-emptive action (for example, no action other than logging the error and turning on the attention LED is taken for PSU or blade-specific failures).

5 Chassis Manager/Blade APIs

The Chassis Manager/blade protocol is a small subset of the intelligent platform management interface (IPMI2.0) protocol. IPMI2.0 is therefore not required for the blade and Chassis Manager to communicate; only message format compatibility with IPMI2.0 is required.

The Chassis Manager communicates with the target blade BMC firmware using IPMI basic mode over the IPMI Serial/Modem Interface (Reference: IPMI 2.0—14.4 Basic Mode).

Note: For the purpose of completeness, this document contains abstracts and references to the IPMI 2.0 specification http://www.intel.com/content/www/us/en/servers/ipmi/ipmi-home.html and to the DCMI 1.5 specification http://www.intel.com/content/www/us/en/data-center/dcmi/data-center-manageability-interface.html. References to these documents are not subject to the Microsoft OWF CLA 1.0 commitment for this specification.

Table 15 lists the IPMI commands that are required or optional for compute blades and storage blades. (Note that “M” is mandatory and “O” is optional.)

Note: Storage blade command requirements differ from compute blade commands.

Table 15: Required IPMI fields for blades

Command name	Reference	Type	Fn	Cmd	Compute blade	JBOD blade
Get Device ID	20.1	App	06h	01h	M	M
Get System GUID	22.14	App	06h	37h	M	M
Get Channel Authentication Capabilities	22.13	App	06h	38h	M	M
Get Session Challenge	22.16	App	06h	39h	M	N/A
Activate Session	22.17	App	06h	3Ah	M	N/A
Set Session Privilege Level	22.18	App	06h	3Bh	M	N/A
Close Session	22.19	App	06h	3Ch	M	N/A
Get Message	22.6	App	06h	33h	M	N/A
Send Message	22.7	App	06h	34h	M	N/A
Get Chassis Status	28.2	Chassis	00h	01h	M	M
Chassis Control	28.3	Chassis	00h	02h	M	N/A
Chassis Identify	28.5	Chassis	00h	04h	M	N/A
Set Power Restore Policy	28.8	Chassis	00h	06h	M	N/A

Command name	Reference	Type	Fn	Cmd	Compute blade	JBOD blade
Set Power Cycle Interval	28.9	Chassis	00h	0Bh	M	N/A
Set System Boot Options	28.12	Chassis	00h	08h	M	N/A
Get System Boot Options	28.13	Chassis	00h	09h	M	N/A
Get Sensor Reading Factors	35.5	Sensor	04h	23h	M	N/A
Get Sensor Reading	35.14	Sensor	04h	2Dh	M	N/A
Get Sensor Type	35.16	Sensor	04h	2Fh	M	N/A
Read FRU Data	34.2	Storage	0Ah	11h	M	M
Write FRU Data	34.3	Storage	0Ah	12h	M	M
Reserve SDR Repository	33.11	Storage	0Ah	22h	M	N/A
Get SDR	33.12	Storage	0Ah	23h	M	N/A
Reserve SEL	31.4	Storage	0Ah	42h	M	N/A
Get SEL Entry	31.5	Storage	0Ah	43h	M	N/A
Add SEL Entry	31.6	Storage	0Ah	44h	M	N/A
Clear SEL	31.9	Storage	0Ah	47h	M	N/A
Set Serial/Modem Mux	25.3	Transport	0Ch	12h	M	N/A
Get Power Reading	N/A	DCMI	2Ch	02h	M	N/A
Get Power Limit	N/A	DCMI	2Ch	03h	M	N/A
Set Power Limit	N/A	DCMI	2Ch	04h	M	N/A
Activate Power Limit	N/A	DCMI	2Ch	05h	M	N/A
Get Processor Info	N/A	OEM	30h	1Bh	M	N/A
Get Memory Info	N/A	OEM	30h	1Dh	M	N/A
Get PCIe Info	N/A	OEM	30h	44h	M	N/A
Get Nic Info	N/A	OEM	30h	19h	M	N/A
BMC Debug	N/A	OEM	30h	F9h	M	N/A
Get BIOS Code	N/A	OEM	32h	73h	M	N/A
Get Disk Status	N/A	OEM Group	2Eh	C4h	N/A	M
Get Disk Info	N/A	OEM Group	2Eh	C5h	N/A	M

M = Mandatory, N/A = Not applicable to blade typ

5.1 Blade Implementation Requirements

The following sections describe the blade implementation requirements.

Only a small subset of commands of IPMI2.0 is required to work with the Chassis Manager. If a blade implements IPMI2.0 and conforms to the requirements described in the sections that follow, it is safe to assume that the blade is compatible with the Chassis Manager.

5.1.1 IPMI Interfaces

- Keyboard Controller Style (KCS) for SMS or BT Interface
- Serial Interface

5.1.2 IPMI Bus Support

- Intelligent Platform Management Bus (IPMB)
- System Bus (system Interface)
- SMBus (I2C)
- Serial - IPMI Basic Mode

5.1.3 Intel® Node Manager

Blades containing Intel processors with the Intel ME~~® Manageability Engine~~ should have Intel® Node Manager enabled in the configuration. The BMC IPMB Channel Number should be 6 to align with the reference code default. This deviates from the IPMI 2.0 specification whereby the Primary IPMB Channel Number is 0.

5.1.4 BMC Debug Port

The Blade Hardware specification defines a dedicated BMC UART for the purpose of BMC debug. This port should allow IPMI over serial commands be sent directly to the BMC and should support a BMC debug mode whereby the BMC diagnostic can directly be interrogated. The default serial port baud rate should be set to 115.2 kbps.

5.1.5 BMC Bypass Jumper

The BMC can be enabled/disabled using the baseboard jumper (see the Blade Hardware specification for jumper numbering and references). This jumper will suspend BMC operation during the boot process, disabling watch dog interrupts and BMC activity. The jumper is for debug only, when the BMC prevents system BIOS diagnostic or the BMC is suspected of causing a boot failure.

5.1.6 BMC Diagnostic Login

The BMC diagnostic console login user name and password should be set as follows:

- Username = sysadmin
- Password = superuser

5.1.7 IPMI Serial Session Login

The IPMI session authentication username and password should be set as follows:

- Username = admin
- Password = admin

5.1.8 Serial Port Timeout

The serial line has a receiving transmission timeout of 100 milliseconds (ms). The BMC must therefore respond to all serial IPMI requests within 100ms. When it receives a response from the baseboard management controller (BMC), the Chassis Manager might immediately send another request. It is therefore possible that the BMC will receive multiple requests within a 100ms timeframe, depending on its response time.

5.1.9 Session Timeout

IPMI basic mode over a serial interface allows only one IPMI session. The IPMI session termination and timeout must be able to be configured with the **Set Serial/Modem Configuration** command. The timeout period is set in 30 second increments. The termination command must allow inactivity timeout termination. The default timeout preconfigured in the IPMI firmware should be 180 seconds.

5.1.10 BMC Serial Port Baud Rate

The blade IPMI basic mode default serial port baud rate should be set to 115.2 kbps. The IPMI serial port baud rate must be able to be configured with the **Set Serial/Modem Configuration** command. 115.2Kbps should be the default set in the System BIOS.

5.1.11 Sensor Data Record

The pulse-width modulation (PWM) sensor is the first sensor data record; however, if an alternative temperature sensor is preferred, this sensor can be the first sensor stored in the sensor data record (SDR). When enumerating the SDR, the first record accessed (0000) must always be the PWM or the alternative temperature sensor, depending on the PWM implementation.

5.1.12 System Event Log

The System Event Log (SEL) should implement circular logging. The maximum number of records in the circular log should be 226 records or roughly 4KB in memory. When the SEL is full, the log should overwrite the oldest record with the newest record.

5.1.13 Event Throttling

Certain system events that are generated in quick succession may need to be throttled so they don't overwhelm the BMC.

For example, during temperature excursions, PROCHOT events may occur in quick succession causing the Intel ME to send -many "**Add SEL Entry** commands via the IPMB to the BMC for CPU Thermal Status. The system should be configured to perform the following to prevent the BMC from becoming overwhelmed:

1. Register the CPUx_PROCHOT interrupt configuration to the falling edge trigger.
    a. When interrupt occurs, the BMC will proceed with the following steps:
        1. Unregister interrupt configuration for CPUx_PROCHOT.
        2. Log "CPUx_PROCHOT asserted" SEL for this event.
        3. Wait for 1 minute.
2. After 1 minute, the BMC will proceed with the following steps:
    a. Register interrupt configuration for CPUx_PROCHOT again.
    b. If there are still interrupts occurring, unregister interrupt configuration again, but do not log SEL. Proceed with step a) again after 1 minute.
    c. If there is no new interrupt occur for 10 seconds, check the level of CPUx_PROCHOT. If CPUx_PROCHOT is HIGH, BMC should log "CPUx_PROCHOT deasserted" SEL.

The PROCHOT signal from each CPU will be handled independently. For example, if CPU0_PROCHOT asserts, the above procedure should be executed for CPU0_PROCHOT only. If CPU1_PROCHOT is asserted while CPU0_PROCHOT is unregistered, the BMC should log "CPU1_PROCHOT asserted" SEL.

5.1.14 Keyboard Controller Style Interface

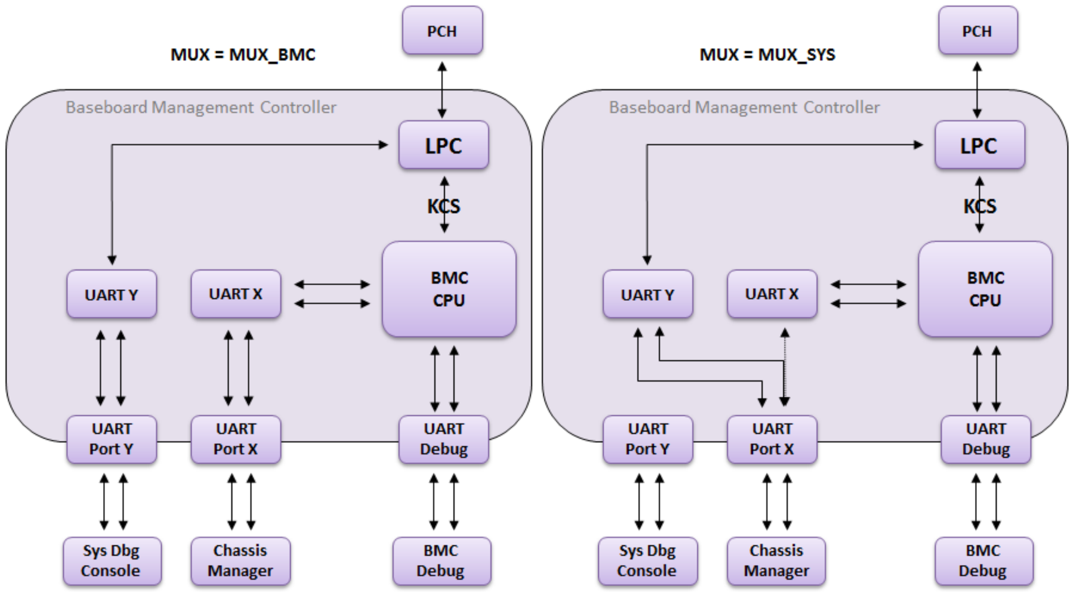
The BMC on compute blades should support the Native Microsoft Windows Server 2012 PMI driver for IPMI communication within the host operating system to the BMC.

5.1.15 Serial Port Sharing

The Chassis Manager to blade BMC communication will use the same serial connection to send IPMI commands and execute Serial Console Redirection to the blade. To fulfill this requirement the BMC should support Serial Port Sharing as described in the IPMI 2.0 specification (see Figure 8).

In the default condition, the mux setting will be MUX_BMC, which is standby to service IPMI command payloads. When the Chassis Manager requires console redirection from the blade, the MUX setting will change to MUX_SYS. While MUX_SYS is activated, the Chassis Manager can change back to MUX_BMC using the methods listed in Figure 8.

Figure 8: Serial port sharing mux diagram



5.1.16 Firmware Decompression

The BMC firmware must deterministically be fully decompressed (loaded) and ready to service IPMI request messages within 24 seconds. Any unused modules from the base firmware that cause decompression delays should be removed.

5.1.17 Inlet Sensor

The blade inlet sensor should be always be sensor number B0h in the Sensor Data Repository (SDR). If the inlet sensor requires an offset caused by hard disk preheat, the offset will be applied by the Chassis Manager after reading the sensor value. It is the responsibility of the blade manufacturer to supply Microsoft with the offset that needs to be applied. The Chassis Manager will identify the blade using the **Get Device Id** IPMI command.

The `[8:10] Manufacturer Id` and `[11:12] Production Id` are used to identify the blade manufacturer and product. The product ID for this specification should be 1030. The manufacturer Id should comply with the IPMI specification and IANA "Private Enterprise" ID. If the inlet temperature sensor requires an offset, the **Get Device Id** command bytes `[8:10] Manufacturer Id` should not be 000000h = unspecified or 0FFFFFh = reserved.

5.2 Request and Response Packet Formats

Every request message from the Chassis Manager software has the format shown in Figure 9.

1	2	3	4	5	6	N	N+1
rsAddress	Function	checksum	rqAddress	Seq	cmd	payload	checksum
1 byte	1 byte	1 byte	1 byte	1 byte	1 byte	n byte	1 byte

Figure 9: Request packet format

Every response from the Chassis Manager software to a request has the format shown in Figure 10.

1	2	3	4	5	6	7	N	N+1
rqAddress	Function	checksum	rsAddress	Seq	cmd	completion code	payload	checksum
1 byte	1 byte	1 byte	1 byte	1 byte	1 byte	1 byte	n byte	1 byte

Figure 10: Response packet format

Table 16 provides details of the request and response packets.

Table 16: Details for Request and Response Packets

Number	Packet	Description
1	rqAddress	Requester address, 1 byte Least-significant (LS) bit is 0 for slave addresses and 1 for software IDs Upper 7-bits hold slave address or software ID, respectively Byte is 20h when the BMC is the requester
2	Function	Function code Response function = request function +1 [debug identification]
3	Checksum	2's complement checksum of preceding bytes in the connection header 8-bit checksum algorithm: initialize checksum to 0 For each byte, checksum = (checksum + byte) modulo 256, then checksum = -checksum When the checksum and the bytes are added together, modulo 256, the result should be 0

Number	Packet	Description
4	rsAddress	Responder slave address, 1 byte LS bit is 0 for slave addresses and 1 for software IDs Upper 7-bits hold slave address or software ID, respectively This byte is 20h when the BMC is the responder; the rqAddress will be the software ID from the corresponding request packet
5	Seq	Sequence number, generated by the requester
6	Cmd	Command byte
7	Completion code	Completion code returned in the response to indicated success/failure status of the request
N	Payload	As required by the particular request or response for the command
N+1	Checksum	2's complement of bytes between the previous checksum 8-bit checksum algorithm: Initialize checksum to 0 For each byte, checksum = (checksum + byte) modulo 256, then checksum = -checksum When the checksum and the bytes are added together, modulo 256, the result should be 0

5.3 Packet Framing

Special characters are used to delimit the start and end of a command packet.

Table 17 lists the packet framing characters. Note that the framing and data escape characters are applied after the message fields have been formatted.

Table 17: Special characters used for packet framing

Description	Value
Start character	A0h
Stop character	A5h
Packet handshake	A6h
Data escape character	AAh

Message framing is similar to inter-integrated circuit (I2C) conditional framing, but replaced with start and stop characters and with the addition of a data byte escape character to ensure that the framing characters are not encountered within the body of the packet. The packet handshake character is used for implementing a level of software flow control with the remote application that is accessing the BMC.

The start, stop, and escape characters are not allowed within the body of the message to make sure that the beginning and end of a message is unambiguously delimited. If a byte matching one of the special characters is encountered in the data to be transmitted, it is encoded into a corresponding two-character sequence for transmission. Table 18 summarizes these encoding sequences.

Table 18: Encoding sequences for special characters

Data byte	Encoded sequence
A0h	AAh (ESC), B0h
A5h	AAh (ESC), B5h
AAh	AAh (ESC), BAh
A6h	AAh (ESC), B6h
1Bh <ESC>	AAh (ESC), 3Bh

The first character of the sequence is always the escape character. Only the special characters plus the ASCII escape <ESC> character, 1Bh, are escaped. (Note that the ASCII escape <ESC> character, 1Bh, is escaped to let the BMC snoop for certain escape sequences in the data stream, such as the <ESC>(and <ESC>Q patterns.) All other byte values in the message are transmitted without escaping.

When the packet is received, the process is reversed. If the two-byte escape sequence is detected in the packet, it is converted to the corresponding data-byte value. Note that the BMC will reject any messages that have illegal character combinations or that exceed message buffer length limits. The BMC may not send an error response for these conditions.

The handshake character is used to signal that the BMC has freed space in its input buffers for a new incoming message. The BMC typically returns a handshake character within one millisecond of being able to accept a new message, unless the controller has already initiated a message transmission or an operation such as firmware update. Note that the handshake character is used in the system for flow control and error detection. Even if unauthenticated IPMI messages are being rejected (or dropped) by the BMC, the BMC is expected to respond with a handshake to indicate its buffers are ready for a new incoming message.

Figure 11 shows the message payload encapsulated with the serial start and stop bytes.

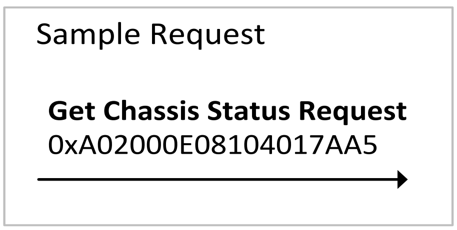
Request packet:

1	2	3	4	5	6	7	N	N+1	N+2
Start (A0h)	rsAddress	Function	checksum	rqAddress	Seq	cmd	payload	checksum	Stop (A5h)
1 byte	1 byte	1 byte	1 byte	1 byte	1 byte	1 byte	n byte	1 byte	1 byte

Response packet:

1	2	3	4	5	6	7	8	N	N+1	N+2
Start (A0h)	rqAddress	Function	checksum	rsAddress	Seq	cmd	completion code	payload	checksum	Stop (A5h)
1 byte	1 byte	1 byte	1 byte	1 byte	1 byte	1 byte	1 byte	n byte	1 byte	1 byte

Sample framed request packet



Sample framed response packet

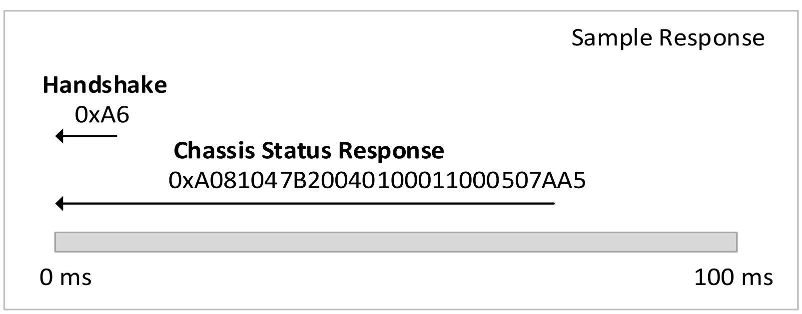


Figure 11: Message payload encapsulated with the serial start and stop bytes

5.4 Serial Console Redirection

To support serial console redirection, a blade should support serial port sharing.

Serial port sharing is a mechanism in which the BMC controls logic that lets a serial controller on the baseboard and a serial controller for the BMC share a single serial connector. To support serial console redirection, the blade BMC should be able to switch serial port control to and from the system baseboard.

Serial port sharing lets the IPMI basic mode messaging and system console redirection coexist on the same physical serial port. The BMC firmware should let the **Set Serial/Modem Mux** command switch the serial port from "`BMC`" to "`SYS`" for terminal console redirection. When the multiplexer (mux) is switched to the system (SYS), the firmware should snoop the in-bound traffic to detect IPMI message patterns for the IPMI **Get Channel Authentication Capabilities** and the **Set Serial/Modem Mux** (with switch to BMC parameters) commands. If it detects an IPMI message pattern matching **Get Channel Authentication Capabilities** or the **Set Serial Modem Mux** back to BMC, the BMC firmware should take control of the serial port from the system and respond to the IPMI message. The BMC firmware should retain control of the serial port until it receives a new request to switch control of the serial port back to the system.

When the BMC has control of the serial port, the messaging protocol should comply with the Chassis Manager API (see Open CloudServer Chassis Manager user interface specification). When in console redirection mode, the system should support VT100 console output.

Table 19 lists the conditions that determine the blade BMC behavior and the switching mechanism that enables the transitions between BMC messaging and console redirection.

Table 19: Conditions causing switching

Switching	Conditions
Switch to system	• IPMI **Set Serial/Modem Mux (to SYS)** is received over serial
Switch to BMC	• IPMI **Set Serial/Modem Mux (to BMC)** is received over serial • The IPMI message **Get Channel Authentication Capabilities** is received over serial

5.5 Chassis Manager Serial Port Session

The Chassis Manager lets a client establish a serial port session to a target device (for example, a top of rack [TOR] network switch) that is physically connected to the Chassis Manager board. The Chassis Manager API provides four serial port session methods: `StartSerialPortConsole`, `StopSerialPortConsole`, `SendSerialPortData`, and `ReceiveSerialPortData`.

Note that the serial port session API services devices attached to universal asynchronous receiver/transmitter (UART) 1, 2, 5, and 6. UART 4, which services blade devices populated in the chassis, uses a separate API. Note also that the ports allocated for the serial port session are COM 1, 2, 5, and 6.

Before attempting to communicate with the target device, the client should explicitly open a serial port session using `StartSerialPortConsole` with a parameter specifying the target serial port. The Chassis Manager will then attempt to open and initialize the target serial port. Currently, the serial baud rate supported for the serial port session is 9600 bps. When it receives the response for the `StartSerialPortConsole` request, the client must check the completion code to see if the serial port session has been successfully established. In addition, the client should use the session token stored in the response when issuing subsequent requests to the Chassis Manager throughout the session.

When a serial port session has been successfully established, the client can issue commands to the target device using `SendSerialPortData`. The client can also receive data packets from the target device using `ReceiveSerialPortData`. Note that `ReceiveSerialPortData` is designed to operate asynchronously with `SendSerialPortData`, which means that the client can invoke `ReceiveSerialPortData` even if there is no preceding or matching invocation `SendSerialPortData`. The asynchronous design ensures that the client can reliably receive the data packets sent by the target device even without an explicit command (for example, console output messages generated while the target device is booting up).

After all the packets of interest have been communicated, the client should explicitly close the session with `StopSerialPortConsole`. The Chassis Manager will then release the target serial port and make it available for other clients.

5.6 Command Completion Codes

Table 20 lists the completion codes, follows the IPMI 2.0 completion codes.

Table 20: Command-completion codes

Code	Description
Generic completion codes (00h, C0h-FFh)	
00h	• Command completed normally
C0h	• Node busy • Command could not be processed because command processing resources are temporarily unavailable
C1h	• Invalid command • Used to indicate an unrecognized or unsupported command
C2h	• Command invalid for given logical unit number (LUN)
C3h	• Timeout while processing command • Response unavailable

Code	Description
C4h	• Out of space • Command could not be completed because of a lack of storage space required to execute the given command operation
C5h	• Reservation canceled or invalid reservation ID
C6h	• Request data truncated
C7h	• Request data length invalid
C8h	• Request data field length limit exceeded
C9h	• Parameter out of range • One or more parameters in the data field of the request are out of range (different from the invalid data field [CCh] code in that it indicates that the erroneous field(s) has a contiguous range of possible values)
CAh	• Cannot return number of requested data bytes
CBh	• Requested sensor, data, or record not present
CCh	• Invalid data field in request
CDh	• Command illegal for specified sensor or record type
CEh	• Command response could not be provided
CFh	• Cannot execute duplicated request • This completion code is for devices which cannot return the response that was returned for the original instance of the request; such devices should provide separate commands that allow the completion status of the original request to be determined • An event receiver does not use this completion code, but returns the 00h completion code in the response to (valid) duplicated requests
D0h	• Command response could not be provided; SDR repository in update mode
D1h	• Command response could not be provided; device in firmware update mode
D2h	• Command response could not be provided; BMC initialization or initialization agent in progress
D3h	• Destination unavailable; cannot deliver request to selected destination (for example, this code can be returned if a request message is targeted to SMS but the receive message queue reception is disabled for the particular channel)
D4h	• Cannot execute command due to insufficient privilege level or other security-based restriction (for example, disabled for firmware firewall)

Code	Description
D5h	• Cannot execute command • Command, or request parameter(s), not supported in present state
D6h	• Cannot execute command • Parameter is illegal because command sub -function has been disabled or is unavailable (for example, disabled for firmware firewall)
FFh	• Unspecified error
Device-specific (OEM) codes (01h-7Eh)	
01h-7Eh	• Device-specific (OEM) completion codes • This range is used for command-specific codes that are also specific to a particular device and version (prior knowledge of the device command set is required for interpretation)
Command-specific codes (80h-BEh)	
80h-BEh	• Standard command-specific codes • This range is reserved for command-specific completion codes for commands specified in this document

5.7 Blade Command Payload

The Chassis Manager protocol payload carries the blade commands, as shown in Figure 12.

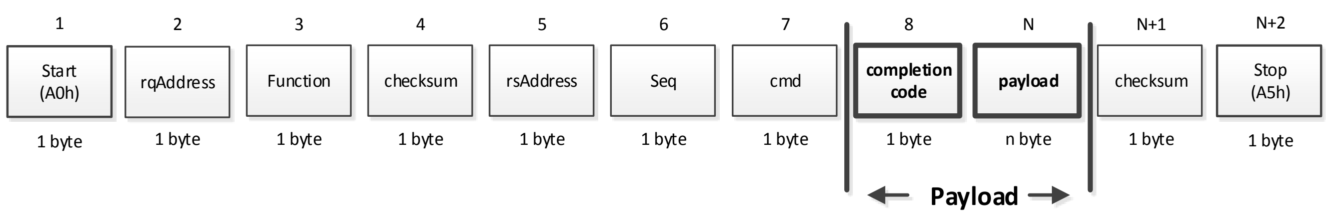


Figure 12: Chassis Manager payload

Certain support commands are required for blade identification and session establishment. The following commands should always be accepted by the blade, whether sent outside of an active session or within the context of an active session:

- Get System GUID
- Get Channel Authentication Capabilities
- Get Session Challenge
- Activate Session

5.7.1 Blade Identification Commands

The **Get Channel Authentication Capabilities** command is used to identify the blade type (compute or JBOD). A blade should respond to this command before and after a session has been established. The response message should use byte 9 (OEM auxiliary data) to advertise the blade type: 0x04 for compute or 0x05 for storage. When response message byte 9 is combined with the OEM ID (bytes 6:8), the Chassis Manager will know both the OEM and type of blade.

Compute blade implementations are required to support session-based authentication, while JBOD blades are not required to support-session based authentication. A blade advertises its authentication support through the **Get Channel Authentication Capabilities** command, byte 3 and byte 4. If a JBOD blade does not support authentication, then byte 3 and byte 4 should equal zero. If authentication is not supported on a JBOD blade, all mandatory (M) IPMI commands listed in this specification should be supported without a session initialization processes.

Note that user session-based authentication is required for all OCS system compute blades. Per-message-based authentication is not supported because session headers are not support over serial IPMI.

5.7.2 Session Establishment Commands

Before general messaging can occur, a session must be activated through a set of session setup commands: **Activate Session**, **Get Session Challenge**, and **Set Session Privilege Level**. These commands can be thought of as always being unauthenticated. Note that Activate Session is the first, and in some cases only, authenticated command for a session. Figure 13 shows the process.

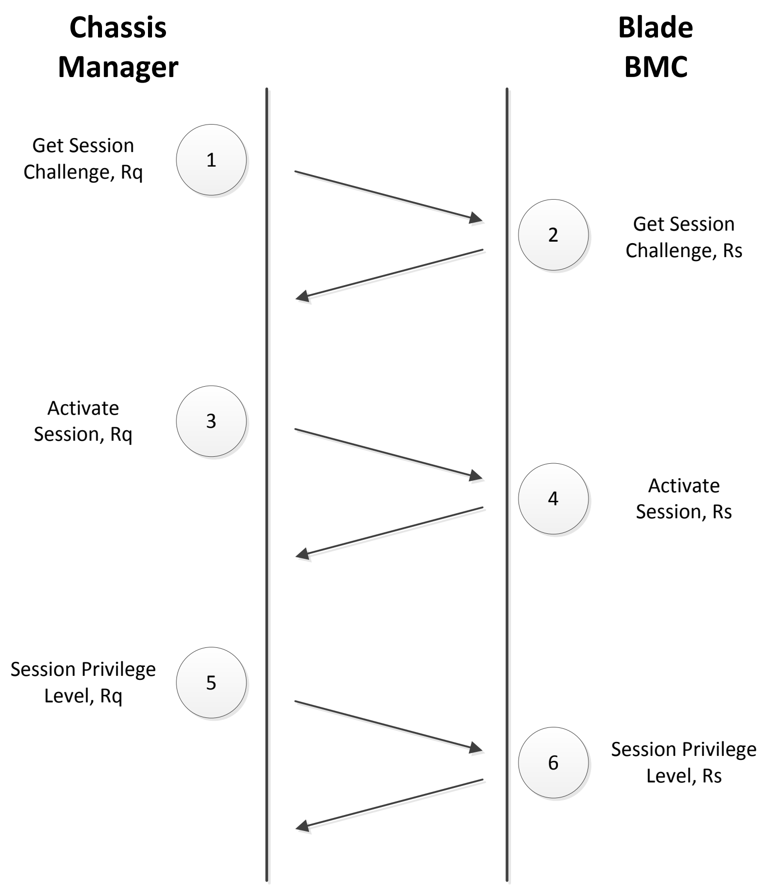


Figure 13: Session establishment process

The following steps are used to establish a session:

1. The Chassis Manager sends a **Get Channel Authentication Capabilities** to the blade to get information about the blade type and session authentication support.
   If sessions are supported, the Chassis Manager advances to step 2.
   If the blade is a session-less JBOD (some JBOD blades require no authentication or session establishment), the Chassis Manager skips steps 2-7 during the initialization process.

2. The Chassis Manager sends a **Get Session Challenge** request to the BMC with an authentication type of "none" and a username that selects which set of user information should be used for the session (UserId 1, if default account is UserId 0). This is the only place where the username is used during the process.

3. The BMC looks up the user information associated with the username. If the user is found and allowed access via the given channel, the BMC returns a **Get Session Challenge** response that includes a randomly generated challenge string and a temporary session ID. The BMC keeps track of the username associated with the session ID; the session ID is used to look up the user's information in step 4.

4. The Chassis Manager then issues an **Activate Session** request. The request contains the temporary session ID plus the authentication information (None). (For example, the serial/modem connection might only pass a simple clear-text password in the activate session data.) The authentication format for different authentication types is specified in the description of the **Activate Session** command. Note that the WCS system specification only supports authentication type None (0x00).

5. The BMC uses the temporary session ID to look up the information for the user identified in the **Get Session Challenge** request (for example, the user's password/key data and a stored copy of the earlier challenge string) and uses it to verify that the packet signature or password is correct.

6. The Chassis Manager then sends a **Set Session Privilege Level** command to the BMC. This command is sent in authenticated format. After the session is activated, the session is set to an initial privilege level. A session that is activated is initially set to USER level, regardless of the maximum privilege level requested in the **Activate Session** command. The Chassis Manager must raise the privilege level of the session using this command to execute commands that require a higher level of privilege.

7. The BMC looks up the privilege level set for the user. If the privilege level requested matches the level set for the user, the BMC responses to the Chassis Manager **Set Session Privilege Level** with a positive completion code. Note that **Set Session Privilege Level** cannot be used to set a privilege level higher than the lowest of the privilege level set for the user.

5.7.3 Session Termination

If the blade BMC supports IPMI single-session serial connections and a session is established, the BMC should accept a Close Session command and immediately terminate the session, freeing up resources for another session to be established.

If a session is not closed using Close Session, the session should time out and close itself after a specified time interval. This time interval should be configurable with the Set Serial/Modem Configuration command. (Note that the IPMI firmware should support all Set Serial/Modem Configuration command parameters defined in this specification.)

If the Chassis Manager follows the sequence described in 5.7.2 above to establish a new session without sending a Close Session, or before the time out interval for the current session has expired, the blade BMC should always terminate the previous session and allow a new session to be established. The Activate Session should always complete successfully to establish a new session.

5.8 Command Formats

The following sections describe the formats of the Chassis Manager protocol payload commands.

Note that the following sections copy command formats from the IPMI 2.0 specification. In addition to IPMI-defined command formats, this section details system-defined OEM commands and modifications to IPMI command payloads, such as the Get Channel Authentication Capabilities command.

5.8.1 Get Device ID

Refer to the IPMI 2.0 specification.

5.8.2 Get System GUID

Refer to the IPMI 2.0 specification.

5.8.3 Get Channel Authentication Capabilities

The blade should respond to the Get Channel Authentication Capabilities command outside of an active session. The command can also be executed within the context of an active session.

Note: Byte 9 is a purposeful deviation from the IPMI 2.0 specification.

Table 21 describes the **Get Channel Authentication Capabilities** command request.

Table 21: Get channel authentication capabilities command request

Byte	Description
1	Channel number [7] – 1b = get IPMI v2.0+ extended data If the given channel supports authentication but does not support RMCP+ (for example, a serial channel), then the response data should return with bit [5] of byte 4 = 0b, and byte 5 should return 01h 0b = Backward compatible with IPMI v1.5 Result response data only returns bytes 1:9, bit [7] of byte 3 (authentication type support) and bit [5] of byte 4 returns as 0b, bit [5] of byte byte 5 returns 00h. [6:4] – reserved [3:0] – channel number 0h-7hBh, Fh = channel numbers Eh = retrieve information for channel this request was issued on.

Byte	Description
2	Requested maximum privilege level [7:4] – reserved [3:0] – requested privilege level 0h = reserved 1h = Callback level 2h = User level 3h = Operator level 4h = Administrator level 5h = OEM proprietary level

Table 22 describes the **Get Channel Authentication Capabilities** command response.

Table 22: Get channel authentication capabilities command response

Byte	Description
1	Completion code
2	Channel number Channel number that the authentication capabilities are being returned for. If the channel number in the request was set to Eh, this will return the channel number on which the request was received
3	Authentication type support Returns the setting of the authentication type enable field from the configuration parameters for the given channel that corresponds to the requested maximum privilege level [7] - 1b = IPMI v2.0+ extended capabilities available (See extended capabilities field below) 0b = IPMI v1.5 support only [6] - reserved [5:0] - IPMI v1.5 authentication type(s) enabled for given requested maximum privilege level All bits: 1b = supported 0b = authentication type not available for use. [5] - OEM proprietary (per OEM identified by the IANAOEM ID in the RMCPPing Response) [4] - straight password / key [3] - reserved [2] - MD5

Byte	Description
	[1] - MD2 [0] - none
4	[7:6] – reserved [5] - KG status (two-key login status). Applies to v2.0/RMCP+ RAKP authentication only (otherwise, ignore as "reserved") 0b = KG is set to default (all 0's). User key KUID will be used in place of KG in RAKP (knowledge of KG not required for activating session) 1b = KG is set to non -zero value (knowledge of both KG and user password (if not anonymous login) required for activating session) The following bits apply to IPMI v1.5 and v2.0: [4] -Per-message authentication status 0b = Per-message authentication is enabled; packets to the BMC must be authenticated per authentication type used to activate the session and user level authentication setting 1b = Per-message authentication is disabled; Authentication Type is "none" accepted for packets to the BMC after the session has been activated. [3] - User Level Authentication status 0b = User Level Authentication is enabled. User Level commands must be authenticated per Authentication Type used to activate the session. 1b = User Level Authentication is disabled. Authentication Type "none" is accepted for User Level commands to the BMC. [2:0] -Anonymous Login status; this parameter returns values that tells the remote console whether there are users on the system that have "null" usernames. This can be used to guide the way the remote console presents login options to the user. (see IPMI v1.5 specification sections 6.9.1, "Anonymous Login" Convention and 6.9.2, Anonymous Login) [2] - 1b = Non-null usernames enabled. (One or more users are enabled that have non-null usernames). [1] - 1b = Null usernames enabled (One or more users that have a null username, but non-null password, are presently enabled) [0] - 1b = Anonymous Login enabled (A user that has a null username and null password is presently enabled)
5	For IPMI v1.5: -reserved For IPMI v2.0+: -extended capabilities [7:2] - reserved [1] - 1b = channel supports IPMI v2.0 connections. [0] - 1b = channel supports IPMI v1.5 connections

Byte	Description
6:8	OEM ID Identification bytes: IANA Enterprise number for OEM/organization. This field must return an OEM Id irrespective of authentication type available.
9	Compute blade = 0x04, JBOD blade = 0x05

5.8.4 Get Session Challenge

Refer to the IPMI 2.0 specification.

5.8.5 Activate Session

Refer to the IPMI 2.0 specification.

5.8.6 Set Session Privilege Level

Refer to the IPMI 2.0 specification.

5.8.7 Close Session

Refer to the IPMI 2.0 specification.

5.8.8 Get Message

Refer to the IPMI 2.0 specification.

5.8.9 Send Message

Refer to the IPMI 2.0 specification.

5.8.10 Get Chassis Status

Refer to the IPMI 2.0 specification.

5.8.11 Chassis Control

Refer to the IPMI 2.0 specification.

5.8.12 Chassis Identify

Refer to the IPMI 2.0 specification.

5.8.13 Set Power Restore Policy

Refer to the IPMI 2.0 specification.

5.8.14 Set Power Cycle Interval

Refer to the IPMI 2.0 specification.

5.8.15 Set System Boot Options

Refer to the IPMI 2.0 specification.

5.8.16 Get System Boot Options

Refer to the IPMI 2.0 specification.

5.8.17 Get Sensor Reading Factors

Refer to the IPMI 2.0 specification.

5.8.18 Get Sensor Reading

Refer to the IPMI 2.0 specification.

5.8.19 Get Sensor Type

Refer to the IPMI 2.0 specification.

5.8.20 Read FRU Data

Refer to the IPMI 2.0 specification. Offset should be in bytes no WORD. Fru Inventory Area Info command is not required.

5.8.21 Write FRU Data

Refer to the IPMI 2.0 specification. Offset should be in bytes no WORD. Fru Inventory Area Info command is not required.

5.8.22 Reserve SDR Repository

Refer to the IPMI 2.0 specification.

5.8.23 Get SDR

Refer to the IPMI 2.0 specification.

Note: Sensor types supported are Full (01h), Compact (02h), and Event Only (03h).

5.8.24 Reserve SEL

Refer to the IPMI 2.0 specification.

The reservation process provides a limited amount of protection when records are being deleted or incrementally read.

A Reservation ID value is returned in response to this command. This value is required in other requests, such as the **Clear SEL** command (commands will not execute unless the correct Reservation ID value is provided).

As an example, if the Chassis Manager wants to clear the SEL, it first reserves the repository by issuing a **Reserve SEL** command. The application checks to see if all SEL entries have been handled before issuing the **Clear SEL** command. If a new event had been placed in the SEL after the records were checked but before the **Clear SEL** command, it is possible for the event to be lost. However, the addition of a new event to the SEL causes the present Reservation ID to be canceled, preventing the **Clear SEL** command from executing. The Chassis Manager can then repeat the reserve-check-clear process until it succeeds.

5.8.25 Get SEL Entry

Refer to the IPMI 2.0 specification.

5.8.26 Add SEL Entry

Refer to the IPMI 2.0 specification.

5.8.27 Clear SEL

Refer to the IPMI 2.0 specification.

5.8.28 Set Serial/Modem Mux

The Set Serial/Modem Mux command switches control of the IPMI serial port to the system for console redirection. The command must response to the request message before transferring control of the serial port to the system for console redirection.

Refer to the IPMI 2.0 specification.

5.8.29 Get Power Reading

Refer to the DCMI 1.5 specification.

5.8.30 Get Power Limit

Refer to the DCMI 1.5 specification.

5.8.31 Set Power Limit

Refer to the DCMI 1.5 specification.

5.8.32 Activate Power Limit

Refer to the DCMI 1.5 specification.

5.8.33 Get Processor Info

The **Get Processor Info** command returns the processor type and status.

Table 23 describes the **Get Processor Info** request.

Table 23: Get Processor Info request

Byte	Description
1	Processor index (0-based index)

Table 24 describes the **Get Processor Info** response.

Table 24: Get Processor Info response

Bytes	Description
1	Completion code
2	Processor type (see Table 60)
3:4	Processor frequency (in MHz); LSB first
5	Processor status: 01h is present FFh is not present

Table 25 lists the processor types.

Table 25: Processor types

Code	Processor make and model	Code	Processor make and model
00h	Celeron (Intel)	0Eh	Lynnfield (Intel)
01h	Pentium III (Intel)	0Fh	Lisbon (AMD)
02h	Pentium 4 (Intel)	10h	Phenom II (AMD)
03h	Xeon (Intel)	11h	Athlon II (AMD)
04h	Prestonia (Intel)	12h	Operaton (AMD)
05h	Nocona (Intel)	13h	Suzuka (AMD)
06h	Opteron (AMD)	14h	Core i3 (Intel)
07h	Dempsey (Intel)	15h	Intel® Xeon™® Processor Family E5-2600v1 product family
08h	Clovertown (Intel)	16h	Intel® Xeon™ E5-2600 v2 product

			~~family(Intel)~~
09h	Tigerton (Intel)	17h	Intel® Atom™ Processor S1200 product family
0Ah	Dunnington (Intel)	18h	Intel® Xeon~~™~~® Processor E5-2600 v3 product family
0Bh	Harpertown (Intel)	FFh	No CPU present

5.8.34 Get Memory Info

The **Get Memory Info** command returns information about a given memory DIMM. The command uses 1-based indexing.

If zero is used as the DIMM index input parameter, the response message will follow Table 26.

Table 26. Get Memory Info response (if zero index as DIMM index)

Bytes	Description
1	Completion code
2	DIMM slot number
3	DIMM presence info in bit map for DIMM1 to DIMM8
4	DIMM presence info in bit map for DIMM9 to DIMM16

Table 27 describes the **Get Memory Info** request.

Table 27: Get Memory Info request

Byte	Description
1	DIMM index (1-based index)

Table 28 describes the **Get Memory Info** response. The BMC should return 0x3F in byte 2 and 0x03 in byte 7 when no valid information is available about the DIMM. After the BIOS has updated the BMC with the correct DIMM information, the information should be stored in persistent memory. BMC should return the information from the previous POST even when the blade is turned off.

During power up, BMC should return the stored memory information until the BIOS has updated the BMC with the new memory information.

Table 28: Get Memory Info response (1+ index as DIMM index)

Bytes	Description
1	Completion code
2	Bit[5]: Type 00h: SDRAM 01h: DDR-1 RAM 02h: Rambus 03h: DDR-2 RAM 04h: FBDIMM 05h: DDR-3 RAM 06h: DDR-4 RAM 0Ah: DDR-4 NVDIMM with Supercap 3Fh: No DIMM present
3:4	DIMM speed (in MHz); LSB first
5:6	DIMM size (in Megbytes); LSB first
7	DIMM status: 00h: Reserved 01h: Unknown DIMM type 02h: Ok 03h: Not present 05h: Single bit error 07h: Multi bit error

5.8.35 Get PCIe Info

The **Get PCIe Info** command returns the device type using 1-based indexing. If zero is used as the PCIe index input parameter, the response message will follow Table 29 for the slot mapping.

Table 29: PCIe slot mapping

Index	Location	Supported devices		
1	LAN_MEZZ	CX3 or other LAN mezzanine-card		
2	Tray backplane PCIe slot	PCIex16	PCIex8	PCIex4
3				PCIex4
4			PCIex8	PCIex4
5				PCIex4
6	PCIe_slot 1	PCIex8	PCIex4	
7		N/A	PCIex4	

8	PCIe_slot 2	PCIex8	PCIex4	
9		N/A	PCIex4	
10	PCIe_slot 3	PCIex8	PCIex4	
11		N/A	PCIex4	
12	PCIe_slot 4	PCIex8	PCIex4	
13		N/A	PCIex4	

Table 30 describes the **Get PCIe Info** request.

Table 30: Get PCIe Info request

Byte	Description
1	PCIe Slot Index (1-based index) for PCIe. An index of zero will return the slot mapping.

Table 31 describes the **Get PCIe Info** response to an index of zero.

Table 31: Get PCIe Info response

Bytes	Description
1	Completion code
2	PCIe Presence info in bit map for Slot 1 to Slot 8
3	[4:0] PCIe Presence info in bit map for Slot 9 to Slot 13 [7:5] Reserved – written as not present 0b.

Table 32 describes the **Get PCIe Info** response to non-zero based index. For non-existent devices, the command should complete successfully with Vendor ID, Device ID, Subsystem Vendor ID, and Subsystem ID all set to 0xFFFF.

Table 32: Get PCIe Info response

Bytes	Description
1	Completion code
2:3	Vendor ID; LSB
4:5	Device ID; LSB
6:7	Subsystem Vendor ID; LSB
8:9	Subsystem ID; LSB

5.8.36 Get NIC Info

The **Get NIC Info** command returns the device type and status.

Table 33 describes the **Get NIC Info** request.

Table 33: Get NIC Info request

Byte	Description
1	NIC index (0-based index)

Table 34 describes the **Get NIC Info** response. A completion code of 0xCC should be returned for a NIC index with no NIC installed.

Table 34: Get NIC Info response

Bytes	Description
1	Completion code
2:7	6-byte MAC address

5.8.37 BMC Debug

The BMC Debug command enables the output of KCS and serial command trace debug messages in the BMC diagnostic debug console.

Table 35 describes the **BMC Debug** request.

Table 35: BMC Debug request

Byte	Description
1	Process: 0xFD = All KCS and serial command trace debug messages 0xFC = Fan control function trace debug messages
2	Enable/Disable: 0 = Disable (default) 1 = Enable

Table 36 describes the **BMC Debug** response.

Table 36: BMC Debug response

Byte	Description
1	Completion code

5.8.38 Get BIOS Code

The **Get BIOS Code** command returns the BIOS post code history. The history will be stored in memory allocated by the SNOOP driver. It will assign a total of 512 bytes for the current and previous post code. When the system is reset the driver starts to record the current BIOS post code. Table 37 describes the **Get BIOS Code** request.

Table 37: Get BIOS Code request

Byte	Description
1	Command: 0 = Read current BIOS code 1 = Read previous BIOS code

Table 38 describes the **Get BIOS Code** response.

Table 38: Get BIOS Code response

Byte	Description
1	Completion code 00h = Success CCh = Invalid data field C1h = Invalid command 07h=File error
2:257	BIOS code Stream of BIOS code in hexadecimal value

5.8.39 Get Disk Status

The **Get Disk Status** command returns the status of each disk in the JBOD. Each disk will add 1 byte to the response (see byte 4+ in the response below). The byte that represents the disk will be split, with bits 7-6 representing the disk status and bits 5-0 representing the unique disk number.

The channel parameter in the request packet is used to select the target SAS controller/disk backplane for JBODs. The **Get Disk Status** command uses NetFn: 2Eh/2Fh and command identifier C4h.

Table 39 describes the **Get Disk Status** request.

Table 39: Get Disk Status request

Byte	Description
1	[7:0] – reserve for channel number 0x00

Table 40 describes the **Get Disk Status** response.

Table 40: Get Disk Status response

Bytes	Description
1	Completion code
2	Channel number Default for this specification is 0x00
3	Disk count (total number of disks)
4+N	7-6: disk status 0=normal, 1=failed, 2=error 5-0: disk number Disk number/location ID

5.8.40 Get Disk Info

The **Get Disk Info** command returns sensor information regarding disks in the JBOD such as temperature.

To conform with the communication protocol this command uses OEM-reserved IPMI commands:

- The **Get Disk Status** command uses NetFn: 2Eh/2Fh and command identifier C5h.
- The **Get Disk Temperature** command is expected to return the status of each disk in the JBOD if disk temperatures are available. If not available, the command should report the JBOD temperature. The multiplier byte in the response message will be multiplied against the MS byte of the reading to assist in storing large and negative numbers.

Table 41 describes the **Get Disk Info** request.

Table 41: Get Disk Info request

Byte	Description
1	Channel number 00h for single channel
2	Disk number (if per-disk temperatures are available, otherwise 00h got JBOD)

Table 42 describes the **Get Disk Info** response.

Table 42: Get Disk Info response

Bytes	Description
1	IPMI completion code
2	Reading unit (see IPMI table)

Bytes	Description
3	MS byte multiplier (byte 4); byte should represent unsigned int [7] 1b = negative multiplier 0b = positive multiplier [6-0] reading MS byte multiplier
4	Reading LS byte first
5	Reading MS byte

Following are examples of disk packets:

Request Packet:
0xA0202EB28104C50000B6A5

Example Response Packet for 32.00 degrees C:
A0812F502004C50001000020F6A5

Example Response Packet for -5.02 degrees C:
A0812F502004C500018102058EA5

5.8.41 Network Controller BIOS Integration

The IPMI command requires BIOS interaction to receive the NIC MAC address. BIOS is responsible for communicating the Network Interface Controller MAC address to the BMC in support of the Get NIC Info command.

The BIOS should provide the BMC the MAC address information for each network controller. The BIOS should retrieve the MAC address information from the network controller using the UEFI EFI_SIMPLE_NETWORK_PROTOCOL data structure.

5.8.42 Sensor Data Repository

The first record in the Sensor Data Repository should be the Fan duty cycle: ReqDutyCycle. The Sensor Data Record Formats supported by the Chassis Manager are: Full Sensor Record (0x01) Compact Sensor Record (0x02) and Event Only (0x03).

SDR Entity's and entity IDs should adhere to the IPMI 2.0 Specification and uniquely identify a physical entity and instance or logical group of entities and instances within the system:

IPMI 2.0 - 39: "An Entity Id is a standardized numeric code that is used in SDRs to identify the types of physical entities or FRUs in the system. The codes include values for entities such as Processor, Power Supply, Fan, etc. The Entity ID values are specified in IPMI 2.0 Specification Table 43-13, Entity Id Codes.

The Entity ID is associated with an Entity Instance value that is used to indicate the particular instance of an entity. For example, a system with four processors would use an Entity Instance value of '0' to identify the first processor, '1' for the second, and so on"

Entity Instance values are zero-based. For example, the first temperature sensor will have Entity Instance 0x00.

DIMM temperature sensors should be aggregated by channel. For example, Temp_DIMM_A0 and Temp_DIMM_A1 should be aggregated into one sensor, i.e. Temp_DIMM_A.

The mezzanine temperature sensor (Temp_Mezz) has a slave address of 0x98 and uses the Texas Instruments TMP411 temperature sensor. When the Tray Mezz FPGA control is present, this sensor should be included in the thermal control algorithm for calculating PWM. In addition to the TI TMP411 sensor, there is an additional sensor at address 0xEE (7'b1110_1110). This temperature module returns an 8-bit signed integer with a direct mapping to Celsius (i.e. values [-128, 127] implies [-128°C, 127°C]). This sensor should be included in the SDR, but not form part of the PWM calculation.

The following table shows a sample of a SDR table.

Table 43. Sample SDR:

Sensor name	Record ID	Sensor number	Sensor type	Sensor format	Event reading	Entity ID	Entity instance
ReqDutyCycle	0x01	0x01	0x04	0x01	Threshold	0x1D	0x01
Temp_Inlet	0x04	0xB0	0x01	0x01	Threshold	0x07	0x00
Temp_Outlet	0x05	0xB1	0x01	0x01	Threshold	0x07	0x01
Temp_Mezz	0x03	0x39	0x01	0x01	Threshold	0x07	0x09
Temp_Mezz_Ctrl	0x3E	0x3A	0x01	0x01	Threshold	0x07	0x0A
Temp_CPU0	0x06	0xB2	0x01	0x01	Threshold	0x03	0x00
Temp_CPU1	0x07	0xB3	0x01	0x01	Threshold	0x03	0x01
Temp_DIMM_A	0x08	0xB4	0x01	0x01	Threshold	0x20	0x80
Temp_DIMM_B	0x09	0xB5	0x01	0x01	Threshold	0x20	0x81
Temp_DIMM_C	0x0A	0xB6	0x01	0x01	Threshold	0x20	0x82
Temp_DIMM_D	0x0B	0xB7	0x01	0x01	Threshold	0x20	0x83

Sensor name	Record ID	Sensor number	Sensor type	Sensor format	Event reading	Entity ID	Entity instance
Temp_DIMM_E	0x0C	0xB8	0x01	0x01	Threshold	0x20	0x84
Temp_DIMM_F	0x0D	0xB9	0x01	0x01	Threshold	0x20	0x85
Temp_PCH	0x0E	0xBA	0x01	0x01	Threshold	0x2D	0x00
P3V3	0x0F	0xC4	0x02	0x01	Threshold	0x07	0x02
P5V	0x10	0xC5	0x02	0x01	Threshold	0x07	0x03
PVDDQ_ABC	0x11	0xC6	0x02	0x01	Threshold	0x20	0x86
PVDDQ_DEF	0x12	0xC7	0x02	0x01	Threshold	0x20	0x87
P1V2	0x13	0xC8	0x02	0x01	Threshold	0x07	0x04
P1V1_SSB	0x14	0xC9	0x02	0x01	Threshold	0x07	0x05
P12V	0x15	0xCA	0x02	0x01	Threshold	0x07	0x0A
PV_VCCP_CPU0	0x16	0xCB	0x02	0x01	Threshold	0x03	0x00
PV_VCCP_CPU1	0x17	0xCC	0x02	0x01	Threshold	0x03	0x01
P12V_AUX	0x18	0xCD	0x02	0x01	Threshold	0x07	0x07
HSC Input Power	0x19	0x28	0x0B	0x01	Threshold	0x0A	0x00
HSC Input Volt	0x1A	0xCF	0x02	0x01	Threshold	0x0A	0x01
HSC Status	0x1B	0x29	0x08	0x02	SensorSpecific	0x0A	0x02
CPU0_PROC_HOT	0x21	0xBB	0x01	0x02	Discrete	0x03	0x00
CPU1_PROC_HOT	0x22	0xBC	0x01	0x02	Discrete	0x03	0x01
PVCCP_CPU0_VRHOT	0x23	0xBD	0x01	0x02	Discrete	0x03	0x00
PVCCP_CPU1_VRHOT	0x24	0xBE	0x01	0x02	Discrete	0x03	0x01
PVDQ_ABC_VRHOT	0x25	0xBF	0x01	0x02	Discrete	0x08	0x80
PVDQ_DEF_VRHOT	0x26	0xC0	0x01	0x02	Discrete	0x08	0x83
CPU0_MEM01_HOT	0x27	0xC1	0x01	0x02	Discrete	0x20	0x88
CPU1_MEM01_HOT	0x28	0xC2	0x01	0x02	Discrete	0x20	0x89
SSB ThermalTrip	0x29	0xC3	0x01	0x02	Discrete	0x07	0x08
Watchdog	0x2A	0x93	0x23	0x02	SensorSpecific	0x06	0x00
SEL	0x2B	0x8A	0x10	0x02	SensorSpecific	0x06	0x01

Sensor name	Record ID	Sensor number	Sensor type	Sensor format	Event reading	Entity ID	Entity instance
System Event	0x2C	0x83	0x12	0x02	SensorSpecific	0x06	0x02
CPU0	0x2D	0xD3	0x07	0x02	SensorSpecific	0x03	0x00
CPU1	0x2E	0xD4	0x07	0x02	SensorSpecific	0x03	0x01
CPU_ERROR	0x2F	0xD5	0x07	0x02	SensorSpecific	0x03	0x82
Memory	0x30	0x87	0x0C	0x02	SensorSpecific	0x20	0x8A
POST Error	0x31	0x9E	0x0F	0x02	SensorSpecific	0x22	0x00
PCI-E BUS	0x32	0xA1	0x13	0x02	SensorSpecific	0x31	0x00
CPU IIO Err	0x33	0xA7	0x07	0x02	SensorSpecific	0x03	0x83
CPU QPI Error	0x34	0x9D	0x07	0x02	SensorSpecific	0x03	0x84
System Power	0x35	0x2A	0x09	0x02	SensorSpecific	0x13	0x00
ME Pwr Mode	0x36	0xFC	0x16	0x0A	Discrete	0x2E	0x06
SPS FW Health	0x37	0x17	0xDC	0x75	Discrete	0x2E	0x00
NM Exception	0x38	0x18	0xDC	0x72	Discrete	0x2E	0x02
NM Health	0x39	0x19	0xDC	0x73	Discrete	0x2E	0x01
NM Capabilities	0x3A	0x1A	0xDC	0x74	Discrete	0x2E	0x04
NM Threshold	0x3B	0x1B	0xDC	0x72	Discrete	0x2E	0x03

5.8.43 Hardcoded Sensor Numbers

The following sensor numbers are fixed for WCS. The BMC should use the sensor numbers specified in the following table.

Table 44: Hardcoded sensor numbers:

Sensor name	Sensor number	Sensor type
ReqDutyCycle	0x01	0x04 (Fan)
Temp_Inlet	0xB0	0x01 (Temperature)

6 Chassis Manager REST API

The sections that follow describe the Chassis Manager Rest API.

6.1 User Roles and API Access

The Chassis Manager web service provides a full set of security features, including encryption, integrity, authentication, and fine-grained API-level authorization.

6.2 Encryption and Service Credentials

All data communication between the Chassis Manager web service and clients (web browser or command-line interface) is encrypted using secure socket layers (SSL). The system uses signature-based checksum (signed packets) to prevent tampering and sends data secure HTTP (HTTPS) to ensure integrity. The Chassis Manager web service is also authenticated against the client using Microsoft Certificate Services.

6.3 Client Credentials/Authentication

Client authentication is based on either machine-local or domain-user Windows credentials. Client Windows credentials are automatically obtained from the client computer based on the context of the logged-in user.

6.4 Role-Based API-Level Authorization

Client authorization to the Chassis Manager web service is provided at the granularity of the service APIs. The Chassis Manager APIs are categorized into three security domains:

- U1—APIs that perform Chassis Manager management functions and manage devices that are connected to the Chassis Manager (for example, blades and power supply units).
- U2—APIs that manage devices (for example, blades and power supply units) that are connected to the Chassis Manager.
- U3—APIs that perform only read-only operations.

Note: U1 includes all Chassis Manager APIs, while U2 and U3 include only a subset of the Chassis Manager APIs. Note also that U3 is a subset of U2, which is a subset of U1.

Users authorized to perform Chassis Manager functions can be categorized into three Windows security domains or groups:

- AcsCmAdmin–Users in this group are authorized to perform functions in U1. They have access to all APIs, including APIs for Chassis Manager management functions like “add-user” and “set-NIC.”
- AcsCmOperator–Users in this group are authorized to perform functions in U2, and have access to all APIs except those for Chassis Manager management functions.
- AcsCmUser–Users in this group are authorized to perform functions in U3, and can only access read-only APIs.

Any user attempting to access the Chassis Manager APIs will be authorized based on their role or group. Users who do not belong to any of the three authorization roles or groups are denied access to Chassis Manager API functionalities.

The three Windows groups are available in each Chassis Manager, and local Chassis Manager users can be assigned to any of the three local roles (CM-1/AcsCmAdmin, CM-1/AcsCmOperator, and CM-1/AcsCmUser) using the “add user” and “change user” APIs.

The three authorization roles can also be created in each domain that wants to communicate with the Chassis Manager. For example, Domain-1/User-1 will be checked against the corresponding domain’s authorization roles (Domain-1/AcsCmAdmin, Domain-1/AcsCmOperator, or Domain-1/AcsCmUser). By default, a Chassis Manager administrator has privileges that are equal to the authorization role AcsCmAdmin.

Table 45 lists the APIs and the corresponding authorized user roles.

Table 45: Chassis manager APIs and corresponding authorized roles

APIs	AcsCmAdmin	AcsCmOperator	AcsCmUser
Health/status information APIs			
GetChassisInfo	X	X	X
GetBladeInfo	X	X	X
GetAllBladesInfo	X	X	X
GetVersion	X	X	X
Blade management APIs			
GetBladeHealth	X	X	X
SetBladeAttentionLEDOn	X	X	
SetAllBladesAttentionLEDOn	X	X	
SetBladeAttentionLEDOff	X	X	
SetAllBladesAttentionLEDOff	X	X	
SetBladeDefaultPowerStateOn	X	X	
SetAllBladesDefaultPowerStateOn	X	X	

APIs	AcsCmAdmin	AcsCmOperator	AcsCmUser
SetBladeDefaultPowerStateOff	X	X	
SetAllBladesDefaultPowerStateOff	X	X	
GetBladeDefaultPowerState	X	X	X
GetAllBladesDefaultPowerState	X	X	X
GetPowerState	X	X	X
GetAllPowerState	X	X	X
SetPowerOn	X		
SetAllPowerOn	X		
SetPowerOff	X		
SetAllPowerOff	X		
GetBladeState	X	X	X
GetAllBladesState	X	X	X
SetBladeOn	X		
SetAllBladesOn	X		
SetBladeOff	X		
SetAllBladesOff	X		
SetBladeActivePowerCycle	X		
SetAllBladesActivePowerCycle	X		
ReadBladeLog	X	X	X
ReadBladeLogWithTimeStamp	X	X	X
ClearBladelog	X	X	
GetBladePowerReading	X	X	X
GetAllBladesPowerReading	X	X	X
GetBladePowerLimit	X	X	X
GetAllBladesPowerLimit	X	X	X
SetBladePowerLimit	X	X	
SetAllBladesPowerLimit	X	X	
SetBladePowerLimitOn	X	X	
SetAllBladesPowerLimitOn	X	X	
SetBladePowerLimitOff	X	X	
SetAllBladesPowerLimitOff	X	X	

APIs	AcsCmAdmin	AcsCmOperator	AcsCmUser
GetNextBoot	X	X	X
SetNextBoot	X	X	X
StartBladeSerialSession	X		
SendBladeSerialData	X		
ReceiveBladeSerialData	X		
StopBladeSerialSession	X		
Asset Management APIs			
GetChassisManagerAssetInfo	X	X	X
GetPDBAssetInfo	X	X	X
GetBladeAssetInfo	X	X	X
SetChassisManagerAssetInfo	X	X	
SetPDBAssetInfo	X	X	
SetBladeAssetInfo	X	X	
Serial console device APIs			
StartSerialPortConsole	X	X	
StopSerialPortConsole	X	X	
SendSerialPortData	X	X	
ReceiveSerialPortData	X	X	
Chassis management APIs			
SetChassisAttentionLEDOn	X	X	
SetChassisAttentionLEDOff	X	X	
GetChassisAttentionLEDStatus	X	X	X
ReadChassisLog	X	X	X
ClearChassisLog	X	X	
ReadChassisLogWithTimeStamp	X	X	X
GetChassisHealth	X	X	X
SetACSocketPowerStateOn	X	X	
SetACSocketPowerStateOff	X	X	
GetACSocketPowerState	X	X	X
GetChassisNetworkProperties	X		X
AddChassisControllerUser	X		

APIs	AcsCmAdmin	AcsCmOperator	AcsCmUser
ChangeChassisControllerUserPassword	X		
ChangeChassisControllerUserRole	X		
RemoveChassisControllerUser	X		
PSU APIs			
ResetPsu	X	X	
UpdatePsuFirmware	X		
GetPSUFirmwareStatus	X	X	X

6.5 REST API: Response and Completion Codes

The Chassis Manager interface is based on REST to more easily interface with the machines. The sections that follow describe the Chassis Manager functionality and provide the corresponding APIs and descriptions.

Each REST API call has the following information encapsulated in its return packet to provide high-level response status information:

- `<byte>`
  Completion code
- `<string>`
  Status description (a textual description of the result) when completion code is anything other than success.
- `<int>`
  API version
  Version number of the REST API
  Currently, this has value '1'; it will be updated upon future API or response packet structure changes.

Note: The NoActiveSerialSession = 0xB2 // error is thrown for Stop/Send/Receive serial session commands for both the blade and the port console when there is no active serial session. If you see this error, use a startserialsession API to create an active session.

6.6 REST API: Descriptions, Usage Scenarios, and Sample Responses

The sections that follow provide information about the Chassis Manager REST APIs.

6.6.1 Gets Information about Chassis

ChassisInfoResponse GetChassisInfo(bool bladeInfo, bool psuInfo, bool chassisInfo)

https://localhost:8000/GetChassisInfo?bladeinfo=true&psuInfo=true&chassisInfo=true

Usage scenario:
This API is used to get the status of chassis components including blades (for example, GUID and power status), power supplies (for example, power draw, status, and serial number), batteries (charge level, power output, fault status) and Chassis Manager (for example, MAC/IP address of the network interfaces, and version information). The power supply status is not available during a PSU firmware upgrade process.

Input parameters: (If no parameters are specified, the command fetches result for all)

- bladeInfo (shows information about blades , optional)
- psuInfo (shows information about power supplies , optional)
- chassisInfo (shows Chassis Manager information , optional)

Sample response:


```
<ChassisInfoResponse
xmlns="http://schemas.datacontract.org/2004/07/Microsoft.GFS.WCS.Contracts
" xmlns:i="http://www.w3.org/2001/XMLSchema-instance">
  <CompletionCode>Success</CompletionCode>
  <statusDescription></statusDescription>
  <apiVersion>1</apiVersion>
<chassisController>
  <CompletionCode>Success</CompletionCode>
  <statusDescription></statusDescription>
  <apiVersion>1</apiVersion>
  <assetTag />
  <firmwareVersion />
  <hardwareVersion>0</hardwareVersion>
- <networkProperties>
- <ChassisNetworkPropertiesResponse>
  <completionCode>Success</completionCode>
  <statusDescription></statusDescription>
  <apiVersion>1</apiVersion>
  <ipAddress i:nil="true" />
  <macAddress>08:9E:01:18:0C:07</macAddress>
  <dhcpEnabled>false</dhcpEnabled>
  <dhcpServer i:nil="true" />
  <dnsAddress i:nil="true" />
  <dnsDomain i:nil="true" />
```

```
  <dnsHostName i:nil="true" />
  <gatewayAddress i:nil="true" />
  <subnetMask i:nil="true" />
  </ChassisNetworkPropertiesResponse>
- <ChassisNetworkPropertiesResponse>
  <completionCode>Success</completionCode>
  <statusDescription></statusDescription>
  <apiVersion>1</apiVersion>
  <ipAddress>192.168.100.59</ipAddress>
  <macAddress>08:9E:01:18:0C:08</macAddress>
  <dhcpEnabled>true</dhcpEnabled>
  <dhcpServer>192.168.100.8</dhcpServer>
  <dnsAddress i:nil="true" />
  <dnsDomain>wcs.lab</dnsDomain>
  <dnsHostName>DVTCM03</dnsHostName>
  <gatewayAddress i:nil="true" />
  <subnetMask>255.255.255.0</subnetMask>
  </ChassisNetworkPropertiesResponse>
  </networkProperties>
  <serialNumber />
  <systemUptime>00:00:41.4301650</systemUptime>
  </chassisController>
<psuCollections>
<PsuInfo>
  <CompletionCode>Success</CompletionCode>
  <statusDescription></statusDescription>
  <apiVersion>1</apiVersion>
  <id>1</id>
  <powerOut>1011</powerOut>
  <serialNumber>46-49-51-44-31-32-32-32-30-30-30-31-33-32</serialNumber>
  <state>ON</state>
  </PsuInfo>
<PsuInfo>
  <CompletionCode>Success</CompletionCode>
  <statusDescription></statusDescription>
  <apiVersion>1</apiVersion>
  <id>2</id>
  <powerOut>960</powerOut>
  <serialNumber>46-49-51-44-31-32-32-32-30-30-30-31-30-35</serialNumber>
```

```
  <state>ON</state>
  </PsuInfo>
...
<PsuInfo>
  <CompletionCode>Success</CompletionCode>
  <statusDescription></statusDescription>
  <apiVersion>1</apiVersion>
  <id>6</id>
  <powerOut>925</powerOut>
  <serialNumber>46-49-51-44-31-32-32-32-30-30-30-31-30-37</serialNumber>
  <state>ON</state>
  </PsuInfo>
  </psuCollections>

<bladeCollections>
<BladeInfo>
  <CompletionCode>Success</CompletionCode>
  <statusDescription></statusDescription>
  <apiVersion>1</apiVersion>
  <bladeGuid>ffffffff-ffff-ffff-ffff-ffffffffffff</bladeGuid>
  <bladeName>BLADE1</bladeName>
  <bladeMacAddress>Not Applicable</bladeMacAddress>
  <id>1</id>
  <powerState>ON</powerState>
  </BladeInfo>
<BladeInfo>
  <CompletionCode>Success</CompletionCode>
  <statusDescription></statusDescription>
  <apiVersion>1</apiVersion>
  <bladeGuid>ffffffff-ffff-ffff-ffff-ffffffffffff</bladeGuid>
  <bladeName>BLADE2</bladeName>
  <bladeMacAddress>Not Applicable</bladeMacAddress>
  <id>2</id>
  <powerState>ON</powerState>
  </BladeInfo>
...
<BladeInfo>
  <CompletionCode>Success</CompletionCode>
  <statusDescription></statusDescription>
  <apiVersion>1</apiVersion>
```

```
  <bladeGuid>ffffffff-ffff-ffff-ffff-ffffffffffff</bladeGuid>
  <bladeName>BLADE24</bladeName>
  <bladeMacAddress>Not Applicable</bladeMacAddress>
  <id>24</id>
  <powerState>ON</powerState>
  </BladeInfo>
  </bladeCollections>

  </ChassisInfoResponse>
```

6.6.2 Gets Information about Blade

BladeInfoResponse GetBladeInfo(int bladeId)

https://localhost:8000/GetBladeInfo?bladeid=1

Usage scenario:
This API is used to get information about the blade (for example serial number and version information).

Input parameters:

- bladeId (blade index 1-48)

Sample response:

```
<BladeInfoResponse
xmlns="http://schemas.datacontract.org/2004/07/Microsoft.GFS.WCS.Contracts
" xmlns:i="http://www.w3.org/2001/XMLSchema-instance">
 <bladeResponse>
  <CompletionCode>Success</CompletionCode>
  <statusDescription></statusDescription>
  <apiVersion>1</apiVersion>
  <bladeNumber>1</bladeNumber>
  </bladeResponse>
 <detailedBladeInfo>
  <alertEnabled>true</alertEnabled>
  <assetTag> Will be added </assetTag>
 <bladeBmc>
  <gateway>Not applicable</gateway>
  <guid>4647ff2b-cb75-4ad6-85de-c612c5abdf87</guid>
  <ipAddress>Not applicable</ipAddress>
  <ipmiVersion>Not applicable</ipmiVersion>
  <macAddress>Not applicable</macAddress>
  <netmask>Not applicable</netmask>
```

```
  <solEnabled>true</solEnabled>
  <vlanTag>1</vlanTag>
  </bladeBmc>
  <dhcp>false</dhcp>
  <firmwareVersion>01.03</firmwareVersion>
  <hardwareVersion>T6M</hardwareVersion>
  <id>1</id>
  <ipAddress>0.0.0.0</ipAddress>
  <ipmiEnabled>true</ipmiEnabled>
  <logEnabled>true</logEnabled>
  <macAddress>00-00-00-00-00-00</macAddress>
  <numberComputeNodes>1</numberComputeNodes>
  <serialNumber>MH822400349</serialNumber>
  </detailedBladeInfo>
  </BladeInfoResponse>
```

6.6.3 Gets Information about All Blades

GetAllBladesInfoResponse GetAllBladesInfo()

https://localhost:8000/GetAllBladesInfo?

Usage scenario:
This API is used to get information about all blades (for example serial numbers and version information).

Input parameters:

- bladeId (blade index 1-48)

Sample response:

```
<GetAllBladesInfoResponse
xmlns="http://schemas.datacontract.org/2004/07/Microsoft.GFS.WCS.Contracts
" xmlns:i="http://www.w3.org/2001/XMLSchema-instance">
 <BladeInfoResponse>
 <bladeResponse>
  <CompletionCode>Success</CompletionCode>
  <statusDescription></statusDescription>
  <apiVersion>1</apiVersion>
  <bladeNumber>1</bladeNumber>
  </bladeResponse>
 <detailedBladeInfo>
  <alertEnabled>true</alertEnabled>
  <assetTag>Will be added</assetTag>
```

```
 <bladeBmc>
  <gateway>Not applicable</gateway>
  <guid>b560b268-c9e0-46da-8029-5f8d2eeed61e</guid>
  <ipAddress>Not applicable</ipAddress>
  <ipmiVersion>Not applicable</ipmiVersion>
  <macAddress>Not applicable</macAddress>
  <netmask>Not applicable</netmask>
  <solEnabled>true</solEnabled>
  <vlanTag>1</vlanTag>
  </bladeBmc>
  <dhcp>false</dhcp>
  <firmwareVersion>01.03</firmwareVersion>
  <hardwareVersion>T6M</hardwareVersion>
  <id>1</id>
  <ipAddress>0.0.0.0</ipAddress>
  <ipmiEnabled>true</ipmiEnabled>
  <logEnabled>true</logEnabled>
  <macAddress>00-00-00-00-00-00</macAddress>
  <numberComputeNodes>1</numberComputeNodes>
  <serialNumber>MH822400349</serialNumber>
  </detailedBladeInfo>
  </BladeInfoResponse>
...
 <BladeInfoResponse>
 <bladeResponse>
  <CompletionCode>Success</CompletionCode>
  <statusDescription></statusDescription>
  <apiVersion>1</apiVersion>
  <bladeNumber>24</bladeNumber>
  </bladeResponse>
 <detailedBladeInfo>
  <alertEnabled>true</alertEnabled>
  <assetTag>Blank For Now, need to locate in FRU</assetTag>
 <bladeBmc>
  <gateway>Not applicable</gateway>
  <guid>c7954895-cb36-4a79-a5b3-8fa2fbb0795a</guid>
  <ipAddress>Not applicable</ipAddress>
  <ipmiVersion>Not applicable</ipmiVersion>
  <macAddress>Not applicable</macAddress>
```

```
<netmask>Not applicable</netmask>
<solEnabled>true</solEnabled>
<vlanTag>1</vlanTag>
</bladeBmc>
<dhcp>false</dhcp>
<firmwareVersion>01.03</firmwareVersion>
<hardwareVersion>T6M</hardwareVersion>
<id>24</id>
<ipAddress>0.0.0.0</ipAddress>
<ipmiEnabled>true</ipmiEnabled>
<logEnabled>true</logEnabled>
<macAddress>00-00-00-00-00-00</macAddress>
<numberComputeNodes>1</numberComputeNodes>
<serialNumber>MH822400334</serialNumber>
</detailedBladeInfo>
</BladeInfoResponse>
</GetAllBladesInfoResponse>
```


6.6.4 Turns Chassis Attention LED ON

ChassisResponse SetChassisAttentionLEDOn()

https://localhost:8000/SetChassisAttentionLEDOn?

Usage scenario:
This API is used to turn the chassis attention LED ON.

The attention LED indicates that the Chassis Manager needs attention. It directs service technicians to the correct chassis for repair. Chassis Manager logs are available through the management system to direct repair (through the ReadChassisLog() API). Users can also flag a service requirement by turning ON the attention LED. When possible, repairs will also be self-directed by the chassis management system. Operators/users must make sure that the chassis attention LED is turned OFF after service is complete.

Input parameters:

- None

Sample response:


```
<ChassisResponse
xmlns="http://schemas.datacontract.org/2004/07/Microsoft.GFS.WCS.Contracts
" xmlns:i="http://www.w3.org/2001/XMLSchema-instance">
<completionCode>Success</completionCode>
<statusDescription></statusDescription>
<apiVersion>1</apiVersion>
</ChassisResponse>
```

6.6.5 Turns Chassis Attention LED OFF

ChassisResponse SetChassisAttentionLEDOff()

https://localhost:8000/SetChassisAttentionLEDOff?

Usage scenario:
This API is used to turn the chassis attention LED OFF.

The attention LED indicates that the Chassis Manager needs attention. It directs service technicians to the correct chassis for repair. Chassis Manager logs are available through the management system to direct repair (through the ReadChassisLog() API). Users can also flag a service requirement by turning ON the attention LED. When possible, repairs will also be self-directed by the chassis management system. Operators/users must make sure that the chassis attention LED is turned OFF after service is complete.

Input parameters:

- None

Sample response:


```
<ChassisResponse
xmlns="http://schemas.datacontract.org/2004/07/Microsoft.GFS.WCS.Contracts
" xmlns:i="http://www.w3.org/2001/XMLSchema-instance">
  <completionCode>Success</completionCode>
  <statusDescription></statusDescription>
  <apiVersion>1</apiVersion>
  </ChassisResponse>
```


6.6.6 Gets Chassis Attention LED Status

LEDStatusResponse GetChassisAttentionLEDStatus()

https://localhost:8000/GetChassisAttentionLEDStatus?

Usage scenario:
This API gets the chassis attention LED status (whether ON or OFF).

The attention LED indicates that the Chassis Manager needs attention. It directs service technicians to the correct chassis for repair. Chassis Manager logs are available through the management system to direct repair (through the ReadChassisLog() API). Users can also flag a service requirement by turning ON the attention LED. When possible, repairs will also be self-directed by the chassis management system. Operators/users must make sure that the chassis attention LED is turned OFF after service is complete.

Input parameters:

- None

Sample response:


```
<LEDStatusResponse
xmlns="http://schemas.datacontract.org/2004/07/Microsoft.GFS.WCS.Contracts
```

```
" xmlns:i="http://www.w3.org/2001/XMLSchema-instance">
  <completionCode>Success</completionCode>
  <statusDescription></statusDescription>
  <apiVersion>1</apiVersion>
  <ledState>OFF</ledState>
  </LEDStatusResponse>
```


6.6.7 Turns Blade Attention LED ON

BladeResponse SetBladeAttentionLEDOn(int bladeId)

https://localhost:8000/SetBladeAttentionLEDOn?bladeId=1

Usage scenario:
This API turns the blade attention LED ON.

The attention LED indicates that the blade needs attention. It directs service technicians to the correct blade for repair. Blade logs are available through the management system to direct repair (through the ReadBladeLog() API). Users can also flag a service requirement by turning ON the attention LED. Operators/users must make sure that the blade attention LED is turned OFF after service is complete.

Input parameters:

- bladeId (blade index 1-48)

Sample response:


```
<BladeResponse
xmlns="http://schemas.datacontract.org/2004/07/Microsoft.GFS.WCS.Contracts
" xmlns:i="http://www.w3.org/2001/XMLSchema-instance">
  <CompletionCode>Success</CompletionCode>
  <statusDescription></statusDescription>
  <apiVersion>1</apiVersion>
  <bladeNumber>1</bladeNumber>
  </BladeResponse>
```


6.6.8 Turns All Blade Attention LEDs ON

AllBladesResponse SetAllBladesAttentionLEDOn()

https://localhost:8000/SetAllBladesAttentionLEDOn?

Usage scenario:
This API turns the attention LEDs on all blades ON.

The attention LED indicates that the blade needs attention. It directs service technicians to the correct blade for repair. Blade logs are available through the management system to direct repair (through the ReadBladeLog() API). Users can also flag a service requirement by turning ON the attention LED.

Operators/users must make sure that the blade attention LED is turned OFF after service is complete.

Note that when multiple users are actively trying to access/modify the same state of a single blade or of different blades, ordering is not guaranteed.

Input parameters:

- None

Sample response:


```
- <AllBladesResponse
xmlns="http://schemas.datacontract.org/2004/07/Microsoft.GFS.WCS.Contracts
" xmlns:i="http://www.w3.org/2001/XMLSchema-instance">
- <BladeResponse>
  <CompletionCode>Success</CompletionCode>
  <statusDescription></statusDescription>
  <apiVersion>1</apiVersion>
  <bladeNumber>1</bladeNumber>
  </BladeResponse>
- <BladeResponse>
  <CompletionCode>Success</CompletionCode>
  <statusDescription></statusDescription>
  <apiVersion>1</apiVersion>
  <bladeNumber>2</bladeNumber>
  </BladeResponse>
…
- <BladeResponse>
  <CompletionCode>Success</CompletionCode>
  <statusDescription></statusDescription>
  <apiVersion>1</apiVersion>
  <bladeNumber>24</bladeNumber>
  </BladeResponse>
  </AllBladesResponse>
```


6.6.9 Turns Blade Attention LED OFF

BladeResponse SetBladeAttentionLEDOff(int bladeId)

https://localhost:8000/SetBladeAttentionLEDOff?bladeId=1

Usage scenario:
This API turns the blade attention LED OFF.

The attention LED indicates that the blade needs attention. It directs service technicians to the correct blade for repair. Blade logs are available through the management system to direct repair (through the

ReadBladeLog() API). Users can also flag a service requirement by turning ON the attention LED. Operators/users must make sure that the blade attention LED is turned OFF after service is complete.

Input parameters:

- bladeId (blade index 1-48)

Sample response:


```
<BladeResponse
xmlns="http://schemas.datacontract.org/2004/07/Microsoft.GFS.WCS.Contracts
" xmlns:i="http://www.w3.org/2001/XMLSchema-instance">
  <CompletionCode>Success</CompletionCode>
  <statusDescription></statusDescription>
  <apiVersion>1</apiVersion>
  <bladeNumber>1</bladeNumber>
  </BladeResponse>
```


6.6.10 Turns All Blade Attention LEDs OFF

AllBladesResponse SetAllBladesAttentionLEDOff()

https://localhost:8000/SetAllBladesAttentionLEDOff?

Usage scenario:
This API is used to turn the attention LED on all blades OFF.

The attention LED indicates that the blade needs attention. It directs service technicians to the correct blade for repair. Blade logs are available through the management system to direct repair (through the ReadBladeLog() API). Users can also flag a service requirement by turning ON the attention LED. Operators/users must make sure that the blade attention LED is turned OFF after service is complete.

Note that when multiple users are actively trying to access/modify the same state of a single blade or of different blades, ordering is not guaranteed.

Input parameters:

- None

Sample response:


```
- <AllBladesResponse
xmlns="http://schemas.datacontract.org/2004/07/Microsoft.GFS.WCS.Contracts
" xmlns:i="http://www.w3.org/2001/XMLSchema-instance">
- <BladeResponse>
  <CompletionCode>Success</CompletionCode>
  <statusDescription></statusDescription>
  <apiVersion>1</apiVersion>
  <bladeNumber>1</bladeNumber>
  </BladeResponse>
```

```
- <BladeResponse>
  <CompletionCode>Success</CompletionCode>
  <statusDescription></statusDescription>
  <apiVersion>1</apiVersion>
  <bladeNumber>2</bladeNumber>
  </BladeResponse>
…
- <BladeResponse>
  <CompletionCode>Success</CompletionCode>
  <statusDescription></statusDescription>
  <apiVersion>1</apiVersion>
  <bladeNumber>24</bladeNumber>
  </BladeResponse>
  </AllBladesResponse>
```


6.6.11 Sets Default Blade Power State ON

BladeResponse SetBladeDefaultPowerStateOn(int bladeId)

https://localhost:8000/SetBladeDefaultPowerStateOn?bladeId=1

Usage scenario:
This API sets the default power state of a blade ON.

The default power state of the blade is the state of the blade after it receives AC power, either when a blade is initially inserted in the slot or when power returns after a utility failure. If the default state is set to OFF, the blade will not be powered ON after receiving AC input power, and an explicit SetBladeActivePowerOn() API will need to be sent to power ON the blade. Note that the blade default power state does not affect the active power state of the blade, only their behavior after a power recycle.

Input parameters:

- bladeId (blade index 1-48)

Sample response:


```
<BladeResponse
xmlns="http://schemas.datacontract.org/2004/07/Microsoft.GFS.WCS.Contracts
" xmlns:i="http://www.w3.org/2001/XMLSchema-instance">
  <CompletionCode>Success</CompletionCode>
  <statusDescription></statusDescription>
  <apiVersion>1</apiVersion>
  <bladeNumber>1</bladeNumber>
  </BladeResponse>
```

6.6.12 Sets Default Power State of All Blades ON

AllBladesResponse SetAllBladesDefaultPowerStateOn()

https://localhost:8000/SetAllBladesDefaultPowerStateOn?

Usage scenario:
This API sets the default power state of all blades ON.

The default power state of the blade is the state of the blade after it receives AC power, either when a blade is initially inserted in the slot or when power returns after a utility failure. If the default state is set to OFF, the blade will not be powered ON after receiving AC input power, and an explicit SetBladeActivePowerOn() API will need to be sent to power ON the blade. Note that the blade default power state does not affect the active power state of the blade; the default power state only affects their behavior after a power recycle.

Note that when multiple users are actively trying to access/modify the same state of a single blade or of different blades, ordering is not guaranteed.

Input parameters:

- None

Sample response:


```
- <AllBladesResponse
xmlns="http://schemas.datacontract.org/2004/07/Microsoft.GFS.WCS.Contracts
" xmlns:i="http://www.w3.org/2001/XMLSchema-instance">
- <BladeResponse>
  <CompletionCode>Success</CompletionCode>
  <statusDescription></statusDescription>
  <apiVersion>1</apiVersion>
  <bladeNumber>1</bladeNumber>
  </BladeResponse>
- <BladeResponse>
  <CompletionCode>Success</CompletionCode>
  <statusDescription></statusDescription>
  <apiVersion>1</apiVersion>
  <bladeNumber>2</bladeNumber>
  </BladeResponse>
...
- <BladeResponse>
  <CompletionCode>Success</CompletionCode>
  <statusDescription></statusDescription>
  <apiVersion>1</apiVersion>
  <bladeNumber>24</bladeNumber>
```

```
</BladeResponse>
</AllBladesResponse>
```

6.6.13 Sets Default Blade Power State OFF

BladeResponse SetBladeDefaultPowerStateOff(int bladeId)

http://localhost:8000/SetBladeDefaultPowerStateOff?bladeId=1

Usage scenario:
This API sets the default power state of a blade OFF.

The default power state of the blade is the state of the blade after it receives AC power, either when a blade is initially inserted in the slot or when power returns after a utility failure. If the default state is set to OFF, the blade will not be powered ON after receiving AC input power, and an explicit SetBladeActivePowerOn() API will need to be sent to power ON the blade. Note that the blade default power state does not affect the active power state of the blade; the default power state of a blade only affects the behavior after a power recycle.

Input parameters:

- bladeId (blade index 1-48)

Sample response:

```
<BladeResponse
xmlns="http://schemas.datacontract.org/2004/07/Microsoft.GFS.WCS.Contracts
" xmlns:i="http://www.w3.org/2001/XMLSchema-instance">
  <completionCode>Success</completionCode>
  <statusDescription></statusDescription>
  <apiVersion>1</apiVersion>
  <bladeNumber>1</bladeNumber>
  </BladeResponse>
```

6.6.14 Sets Default Power State of All Blades OFF

AllBladesResponse SetAllBladesDefaultPowerStateOff()

http://localhost:8000/SetAllBladesDefaultPowerStateOff?bladeId=1

Sets the default power state of all blades OFF

Usage scenario:
This API sets the default power state of all blades OFF.

The default power state of the blade is the state of the blade after it receives AC power, either when a blade is initially inserted in the slot or when power returns after a utility failure. If the default state is set to OFF, the blade will not be powered ON after receiving AC input power, and an explicit SetBladeActivePowerOn() API will need to be sent to power ON the blade. Note that the blade default power state does not affect the active power state of the blade; the default power state only affects the behavior after a power recycle.

Note also that when multiple users are actively trying to access/modify the same state of a single blade or of different blades, ordering is not guaranteed.

Input parameters:

- None

Sample response:


```
- <AllBladesResponse
xmlns="http://schemas.datacontract.org/2004/07/Microsoft.GFS.WCS.Contracts
" xmlns:i="http://www.w3.org/2001/XMLSchema-instance">
- <BladeResponse>
  <completionCode>Success</completionCode>
  <statusDescription></statusDescription>
  <apiVersion>1</apiVersion>
  <bladeNumber>1</bladeNumber>
  </BladeResponse>
- <BladeResponse>
  <completionCode>Success</completionCode>
  <statusDescription></statusDescription>
  <apiVersion>1</apiVersion>
  <bladeNumber>2</bladeNumber>
  </BladeResponse>
...
- <BladeResponse>
  <completionCode>Success</completionCode>
  <statusDescription></statusDescription>
  <apiVersion>1</apiVersion>
```

```
<bladeNumber>24</bladeNumber>
</BladeResponse>
</AllBladesResponse>
```

6.6.15 Gets Default Blade Power State

BladeStateResponse GetBladeDefaultPowerState(int bladeId)

https://localhost:8000/GetBladeDefaultPowerState?bladeId=1

Usage scenario:
This API gets the default power state of the blade.

The default power state of the blade is the state of the blade after it receives AC power, either when a blade is initially inserted in the slot or when power returns after a utility failure. If the default state is set to OFF, the blade will not be powered ON after receiving AC input power, and an explicit SetBladeActivePowerOn() API will need to be sent to power ON the blade. Note that the blade default power state does not affect the active power state of the blade; the default power state only affects the behavior after a power recycle.

Input parameters:

- bladeId (blade index 1-48)

Sample response:

```
BladeStateResponse
xmlns="http://schemas.datacontract.org/2004/07/Microsoft.GFS.WCS.Contracts
" xmlns:i="http://www.w3.org/2001/XMLSchema-instance">
- <bladeResponse>
  <completionCode>Success</completionCode>
  <statusDescription></statusDescription>
  <apiVersion>1</apiVersion>
  <bladeNumber<
>1</bladeNumber>
  </bladeResponse>
  <bladeState>ON</bladeState>
  </BladeStateResponse>
```

6.6.16 Gets the Default Power State of All Blades

GetAllBladesStateResponse GetAllBladesDefaultPowerState()

https://localhost:8000/GetAllBladesDefaultPowerState?

Usage scenario:
The default power state of the blade is the state of the blade after it receives AC power, either when a blade is initially inserted in the slot or when power returns after a utility failure. If the default state is set

to OFF, the blade will not be powered ON after receiving AC input power, and an explicit SetBladeActivePowerOn() API will need to be sent to power ON the blade. Note that the blade default power state does not affect the active power state of the blade; the default power state only affects the behavior after a power recycle.

Note also that when multiple users are actively trying to access/modify the same state of a single blade or of different blades, ordering is not guaranteed.

Input parameters:

- None

Sample response:


```
- <GetAllBladesStateResponse
xmlns="http://schemas.datacontract.org/2004/07/Microsoft.GFS.WCS.Contracts
" xmlns:i="http://www.w3.org/2001/XMLSchema-instance">
- <BladeStateResponse>
- <bladeResponse>
  <completionCode>Success</completionCode>
  <statusDescription></statusDescription>
  <apiVersion>1</apiVersion>
  <bladeNumber>1</bladeNumber>
  </bladeResponse>
  <bladeState>ON</bladeState>
  </BladeStateResponse>
- <BladeStateResponse>
- <bladeResponse>
  <completionCode>Success</completionCode>
  <statusDescription></statusDescription>
  <apiVersion>1</apiVersion>
  <bladeNumber>2</bladeNumber>
  </bladeResponse>
  <bladeState>ON</bladeState>
  </BladeStateResponse>
...
- <BladeStateResponse>
- <bladeResponse>
  <completionCode>Success</completionCode>
  <statusDescription></statusDescription>
  <apiVersion>1</apiVersion>
  <bladeNumber>24</bladeNumber>
  </bladeResponse>
  <bladeState>ON</bladeState>
```

```
</BladeStateResponse>
</GetAllBladesStateResponse>
```

6.6.17 Gets Outlet Power State of Blade

PowerStateResponse GetPowerState(int bladeId)

https://localhost:8000/GetPowerState?bladeId=1

Usage scenario:
This API gets the AC outlet power state of a blade (whether or not the blade is receiving AC power).

When ON, the blade is receiving AC power (hard-power state). When AC power is supplied to the blade and when the default power state of the blade is ON, the blade chipset will receive power and the boot process will be initiated. If the default power state of the blade is OFF when AC power is applied, the blade chipset will not receive power (and the boot process will not be initiated) unless an explicit "SetBladeOn" command is sent to the blade.

When OFF, the blade is not receiving AC power

Input parameters:

- bladeId (blade index 1-48)

Sample response:

```
<PowerStateResponse
xmlns="http://schemas.datacontract.org/2004/07/Microsoft.GFS.WCS.Contracts
" xmlns:i="http://www.w3.org/2001/XMLSchema-instance">
- <bladeResponse>
  <completionCode>Success</completionCode>
  <statusDescription></statusDescription>
  <apiVersion>1</apiVersion>
  <bladeNumber>1</bladeNumber>
  </bladeResponse>
  <powerState>ON</powerState>
  </PowerStateResponse>
```

6.6.18 Gets AC Outlet Power State of All Blades

GetAllPowerStateResponse GetAllPowerState()

https://localhost:8000/GetAllPowerState?

Usage scenario:
This API gets the AC outlet power state of all blades (whether or not the blades are receiving AC power).

When ON, the blade is receiving AC power (hard-power state). When AC power is supplied to the blade and when the default power state of the blade is ON, the blade chipset will receive power and the boot process will be initiated. If the default power state of the blade is OFF when AC power is applied, the blade chipset will not receive power (and the boot process will not be initiated) unless an explicit “SetBladeOn” command is sent to the blade.

When OFF, the blade is not receiving AC power.

Note that when multiple users are actively trying to access/modify the same state of a single blade or of different blades, ordering is not guaranteed.

Input parameters:

- None

Sample response:

```
- < GetAllPowerStateResponse
xmlns="http://schemas.datacontract.org/2004/07/Microsoft.GFS.WCS.Contracts
" xmlns:i="http://www.w3.org/2001/XMLSchema-instance">
- <PowerStateResponse>
- <bladeResponse>
  <completionCode>Success</completionCode>
  <statusDescription></statusDescription>
  <apiVersion>1</apiVersion>
  <bladeNumber>1</bladeNumber>
  </bladeResponse>
  <powerState>ON</powerState>
  </PowerStateResponse>
- <PowerStateResponse>
- <bladeResponse>
  <completionCode>Success</completionCode>
  <statusDescription></statusDescription>
  <apiVersion>1</apiVersion>
  <bladeNumber>2</bladeNumber>
  </bladeResponse>
  <powerState>ON</powerState>
  </PowerStateResponse>
...
- <PowerStateResponse>
- <bladeResponse>
  <completionCode>Success</completionCode>
  <statusDescription></statusDescription>
  <apiVersion>1</apiVersion>
  <bladeNumber>24</bladeNumber>
  </bladeResponse>
  <powerState>ON</powerState>
```

```
  </PowerStateResponse>
```

</ **GetAllPowerStateResponse**>

6.6.19 Turns AC Outlet Power ON for Blade

BladeResponse SetPowerOn(int bladeId)

https://localhost:8000/SetPowerOn?bladeId=1

Usage scenario:
This API turns the AC power outlet power ON for a blade.

When ON, the blade is receiving AC power (hard-power state). When AC power is supplied to the blade and when the default power state of the blade is ON, the blade chipset will receive power and the boot process will be initiated. If the default power state of the blade is OFF when AC power is applied, the blade chipset will not receive power (and the boot process will not be initiated) unless an explicit "SetBladeOn" command is sent to the blade.

When OFF, the blade is not receiving AC power.

Note: it would take up to 50 seconds for the BMC to start responding to commands after a setpoweron – AC cycling.

Input parameters:

- bladeId (blade index 1-48)

Sample response:


```
<BladeResponse
xmlns="http://schemas.datacontract.org/2004/07/Microsoft.GFS.WCS.Contracts
" xmlns:i="http://www.w3.org/2001/XMLSchema-instance">
  <completionCode>Success</completionCode>
  <statusDescription></statusDescription>
  <apiVersion>1</apiVersion>
  <bladeNumber>1</bladeNumber>
  </BladeResponse>
```


6.6.20 Turns the AC Outlet Power ON for All Blades

AllBladesResponse SetAllPowerOn()

https://localhost:8000/SetAllPowerOn?

Usage scenario:
This API turns the AC power outlet ON for all blades.

When ON, the blade is receiving AC power (hard-power state). When AC power is supplied to the blade and when the default power state of the blade is ON, the blade chipset will receive power and the boot process will be initiated. If the default power state of the blade is OFF when AC power is applied, the blade chipset will not receive power (and the boot process will not be initiated) unless an explicit "SetBladeOn" command is sent to the blade.

When OFF, the blade is not receiving AC power.

Note that when multiple users are actively trying to access/modify the same state of a single blade or of different blades, ordering is not guaranteed. It would take up to 50 seconds for the BMC to start responding to commands after a setpoweron – AC cycling.

Input parameters:

- None

Sample response:


```
- <AllBladesResponse
xmlns="http://schemas.datacontract.org/2004/07/Microsoft.GFS.WCS.Contracts
" xmlns:i="http://www.w3.org/2001/XMLSchema-instance">
- <BladeResponse>
  <completionCode>Success</completionCode>
  <statusDescription></statusDescription>
  <apiVersion>1</apiVersion>
  <bladeNumber>1</bladeNumber>
  </BladeResponse>
- <BladeResponse>
  <completionCode>Success</completionCode>
  <statusDescription></statusDescription>
  <apiVersion>1</apiVersion>
  <bladeNumber>2</bladeNumber>
  </BladeResponse>
...
- <BladeResponse>
  <completionCode>Success</completionCode>
  <statusDescription></statusDescription>
  <apiVersion>1</apiVersion>
  <bladeNumber>24</bladeNumber>
  </BladeResponse>
  </AllBladesResponse>
```


6.6.21 Turns AC Outlet Power OFF for Blade

BladeResponse SetPowerOff(int bladeId)

https://localhost:8000/SetPowerOff?bladeId=1

Usage scenario:

This API turns the AC power OFF for a blade.

When ON, the blade is receiving AC power (hard-power state). When AC power is supplied to the blade and when the default power state of the blade is ON, the blade chipset will receive power and the boot process will be initiated. If the default power state of the blade is OFF when AC power is applied, the blade chipset will not receive power (and the boot process will not be initiated) unless an explicit "SetBladeOn" command is sent to the blade.

When OFF, the blade is not receiving AC power.

Note: it would take up to 50 seconds for the BMC to start responding to commands after a setpoweroff – AC cycling.

Input parameters:

- bladeId (blade index 1-48)

Sample response:


```
<BladeResponse
xmlns="http://schemas.datacontract.org/2004/07/Microsoft.GFS.WCS.Contracts
" xmlns:i="http://www.w3.org/2001/XMLSchema-instance">
  <completionCode>Success</completionCode>
  <statusDescription></statusDescription>
  <apiVersion>1</apiVersion>
  <bladeNumber>1</bladeNumber>
  </BladeResponse>
```


6.6.22 Turns AC Outlet Power OFF for All Blades

AllBladesResponse SetAllPowerOff()

https://localhost:8000/SetAllPowerOff?

Usage scenario:
This API turns the AC power OFF for all blades.

When ON, the blade is receiving AC power (hard-power state). When AC power is supplied to the blade and when the default power state of the blade is ON, the blade chipset will receive power and the boot process will be initiated. If the default power state of the blade is OFF when AC power is applied, the blade chipset will not receive power (and the boot process will not be initiated) unless an explicit "SetBladeOn" command is sent to the blade.

When OFF, the blade is not receiving AC power.

Note that when multiple users are actively trying to access/modify the same state of a single blade or of different blades, ordering is not guaranteed. It would take up to 50 seconds for the BMC to start responding to commands after a setpoweroff – AC cycling.

Input parameters:

- None

Sample response:


```
- <AllBladesResponse
xmlns="http://schemas.datacontract.org/2004/07/Microsoft.GFS.WCS.Contracts
" xmlns:i="http://www.w3.org/2001/XMLSchema-instance">
  <completionCode>Success</completionCode>
  <statusDescription></statusDescription>
  <apiVersion>1</apiVersion>
- <BladeResponse>
  <completionCode>Success</completionCode>
  <statusDescription></statusDescription>
  <apiVersion>1</apiVersion>
  <bladeNumber>1</bladeNumber>
  </BladeResponse>
- <BladeResponse>
  <completionCode>Success</completionCode>
  <statusDescription></statusDescription>
  <apiVersion>1</apiVersion>
  <bladeNumber>2</bladeNumber>
  </BladeResponse>
...
- <BladeResponse>
  <completionCode>Success</completionCode>
  <statusDescription></statusDescription>
  <apiVersion>1</apiVersion>
  <bladeNumber>24</bladeNumber>
  </BladeResponse>
  </AllBladesResponse>
```


6.6.23 Gets the ON/OFF State of Blade

BladeState GetBladeState(int bladeId)

https://localhost:8000/GetBladeState?bladeId=1

Usage scenario:
This API is used to get the ON/OFF state of a blade (whether or not blade chipset is receiving power).

When ON, the blade is receiving AC power (hard-power state) and the chipset is receiving power (soft-power state).

When OFF, the blade chipset is not receiving power.

Input parameters:

- bladeId (blade index 1-48)

Sample response:

```
<BladeStateResponse
xmlns="http://schemas.datacontract.org/2004/07/Microsoft.GFS.WCS.Contracts
" xmlns:i="http://www.w3.org/2001/XMLSchema-instance">
- <bladeResponse>
  <completionCode>Success</completionCode>
  <statusDescription></statusDescription>
  <apiVersion>1</apiVersion>
  <bladeNumber>1</bladeNumber>
  </bladeResponse>
  <powerState>ON</powerState>
  </BladeStateResponse>
```

6.6.24 Gets the ON/OFF State of All Blades

`GetAllBladesStateResponse` **GetAllBladesState()**

https://localhost:8000/GetAllBladesState?

Usage scenario:
This API gets the ON/OFF state of all blades (whether or not blade chipsets are receiving power).

When ON, the blades are receiving AC power (hard-power state) and the chipsets are receiving power (soft-power state).

When OFF, the blade chipsets are not receiving power.

Note that when multiple users are actively trying to access/modify the same state of a single blade or of different blades, ordering is not guaranteed.

Input parameters:

- None

Sample response:

```
- <GetAllBladesStateResponse
xmlns="http://schemas.datacontract.org/2004/07/Microsoft.GFS.WCS.Contracts
" xmlns:i="http://www.w3.org/2001/XMLSchema-instance">
  <completionCode>Success</completionCode>
  <statusDescription></statusDescription>
  <apiVersion>1</apiVersion>
- <BladeStateResponse>
- <bladeResponse>
```

```
  <completionCode>Success</completionCode>
  <statusDescription></statusDescription>
  <apiVersion>1</apiVersion>
  <bladeNumber>1</bladeNumber>
  </bladeResponse>
  <powerState>ON</powerState>
  </BladeStateResponse>
- <BladeStateResponse>
- <bladeResponse>
  <completionCode>Success</completionCode>
  <statusDescription></statusDescription>
  <apiVersion>1</apiVersion>
  <bladeNumber>2</bladeNumber>
  </bladeResponse>
  <powerState>ON</powerState>
  </BladeStateResponse>
...
- <BladeStateResponse>
- <bladeResponse>
  <completionCode>Success</completionCode>
  <statusDescription></statusDescription>
  <apiVersion>1</apiVersion>
  <bladeNumber>24</bladeNumber>
  </bladeResponse>
  <powerState>ON</powerState>
  </BladeStateResponse>
  </GetAllBladesStateResponse>
```

6.6.25 Supplies Power to the Blade Chipset

BladeResponse SetBladeOn(int bladeId)

https://localhost:8000/SetBladeOn?bladeId=1

Usage scenario:
This API is used to supply power to the blade chipset (initialize the boot process).

When ON, the blade is receiving AC power (hard-power state) and the chipset is receiving power (soft-power state).

When OFF, the blade chipset is not receiving power.

Input parameters:

- bladeId (blade index 1-48)

Sample response:

```
<BladeResponse
xmlns="http://schemas.datacontract.org/2004/07/Microsoft.GFS.WCS.Contracts
" xmlns:i="http://www.w3.org/2001/XMLSchema-instance">
  <completionCode>Success</completionCode>
  <statusDescription></statusDescription>
  <apiVersion>1</apiVersion>
  <bladeNumber>1</bladeNumber>
  </BladeResponse>
```

6.6.26 Supplies Power to All Blade Chipsets

AllBladesResponse SetAllBladesOn()

https://localhost:8000/SetAllBladesOn?

Usage scenario:
This API is used to supply power to the chipsets (initialize the boot process).

When ON, the blades are receiving AC power (hard-power state) and the chipsets are receiving power (soft-power state).

When OFF, the blade chipsets are not receiving power.

Note that when multiple users are actively trying to access/modify the same state of a single blade or of different blades, ordering is not guaranteed.

Input parameters:

- None

Sample response:

```
- <AllBladesResponse
xmlns="http://schemas.datacontract.org/2004/07/Microsoft.GFS.WCS.Contracts
" xmlns:i="http://www.w3.org/2001/XMLSchema-instance">
  <completionCode>Success</completionCode>
  <statusDescription></statusDescription>
  <apiVersion>1</apiVersion>
- <BladeResponse>
  <completionCode>Success</completionCode>
  <statusDescription></statusDescription>
  <apiVersion>1</apiVersion>
  <bladeNumber>1</bladeNumber>
  </BladeResponse>
```

```
- <BladeResponse>
  <completionCode>Success</completionCode>
  <statusDescription></statusDescription>
  <apiVersion>1</apiVersion>
  <bladeNumber>2</bladeNumber>
  </BladeResponse>
...
- <BladeResponse>
  <completionCode>Success</completionCode>
  <statusDescription></statusDescription>
  <apiVersion>1</apiVersion>
  <bladeNumber>24</bladeNumber>
  </BladeResponse>
  </AllBladesResponse>
```


6.6.27 Stops Power to Blade Chipset

BladeResponse SetBladeOff(int bladeId)

https://localhost:8000/SetBladeOff?bladeId=1

Usage scenario:
This API is used to remove or stop power to the chipset.

When ON, the blade is receiving AC power (hard-power state) and the chipset is receiving power (soft-power state).

When OFF, the blade chipset is not receiving power.

Input parameters:

- bladeId (blade index 1-48)

Sample response:


```
<BladeResponse
xmlns="http://schemas.datacontract.org/2004/07/Microsoft.GFS.WCS.Contracts
" xmlns:i="http://www.w3.org/2001/XMLSchema-instance">
  <completionCode>Success</completionCode>
  <statusDescription></statusDescription>
  <apiVersion>1</apiVersion>
  <bladeNumber>1</bladeNumber>
  </BladeResponse>
```


6.6.28 Stops Power to All Blade Chipsets

AllBladesResponse SetAllBladesOff()

https://localhost:8000/SetAllBladesOff?

Usage scenario:
This API is used to remove or turn OFF power to the chipsets.

When ON, the blades are receiving AC power (hard-power state) and the chipsets are receiving power (soft-power state).

When OFF, the blade chipsets are not receiving power.

Note that when multiple users are actively trying to access/modify the same state of a single blade or of different blades, ordering is not guaranteed.

Input parameters:

- None

Sample response:


```
- <AllBladesResponse
xmlns="http://schemas.datacontract.org/2004/07/Microsoft.GFS.WCS.Contracts
" xmlns:i="http://www.w3.org/2001/XMLSchema-instance">
  <completionCode>Success</completionCode>
  <statusDescription></statusDescription>
  <apiVersion>1</apiVersion>
- <BladeResponse>
  <completionCode>Success</completionCode>
  <statusDescription></statusDescription>
  <apiVersion>1</apiVersion>
  <bladeNumber>1</bladeNumber>
  </BladeResponse>
- <BladeResponse>
  <completionCode>Success</completionCode>
  <statusDescription></statusDescription>
  <apiVersion>1</apiVersion>
  <bladeNumber>2</bladeNumber>
  </BladeResponse>
...
- <BladeResponse>
  <completionCode>Success</completionCode>
  <statusDescription></statusDescription>
  <apiVersion>1</apiVersion>
  <bladeNumber>24</bladeNumber>
  </BladeResponse>
```

```
</AllBladesResponse>
```

6.6.29 Power Cycle Blade

BladeResponse SetBladeActivePowerCycle(int bladeId, uint offTime)

https://localhost:8000/SetBladeActivePowerCycle?bladeId=1&offTime=0

Usage scenario:
This API is used to power cycle (or soft reset) a blade.

Power cycle resets the blade (causing a software reboot sequence). The blade AC power signal remains ON throughout the process. Any serial session active on that blade will continue to be active during this process.

Input parameters:

- bladeId (blade index 1-48)
- offTime (time interval [in seconds] when the blade is powered off; if not specified, the default interval is 0 [optional])

Sample response:


```
<BladeResponse
xmlns="http://schemas.datacontract.org/2004/07/Microsoft.GFS.WCS.Contracts
" xmlns:i="http://www.w3.org/2001/XMLSchema-instance">
  <completionCode>Success</completionCode>
  <statusDescription></statusDescription>
  <apiVersion>1</apiVersion>
  <bladeNumber>1</bladeNumber>
  </BladeResponse>
```


6.6.30 Power Cycle All Blades

AllBladesResponse SetAllBladesActivePowerCycle(uint offTime)

https://localhost:8000/SetAllBladesActivePowerCycle?offTime=0

Usage scenario:
This API power cycles (or soft resets) all blades.

Power cycle resets the blade (causing a software reboot sequence). The blade AC power signal remains ON throughout the process. Any serial session active on that blade will continue to be active during this process.

Note that when multiple users are actively trying to access/modify the same state of a single blade or of different blades, ordering is not guaranteed.

Input parameters:

- offTime (time interval [in seconds] when the blade is powered off; if not specified, the default interval is 0 [optional])

Sample response:


```
- <AllBladesResponse
xmlns="http://schemas.datacontract.org/2004/07/Microsoft.GFS.WCS.Contracts
" xmlns:i="http://www.w3.org/2001/XMLSchema-instance">
  <completionCode>Success</completionCode>
  <statusDescription></statusDescription>
  <apiVersion>1</apiVersion>
- <BladeResponse>
  <completionCode>Success</completionCode>
  <statusDescription></statusDescription>
  <apiVersion>1</apiVersion>
  <bladeNumber>1</bladeNumber>
  </BladeResponse>
- <BladeResponse>
  <completionCode>Success</completionCode>
  <statusDescription></statusDescription>
  <apiVersion>1</apiVersion>
  <bladeNumber>2</bladeNumber>
  </BladeResponse>
...
- <BladeResponse>
  <completionCode>Success</completionCode>
  <statusDescription></statusDescription>
  <apiVersion>1</apiVersion>
  <bladeNumber>24</bladeNumber>
  </BladeResponse>
  </AllBladesResponse>
```


6.6.31 Turns Chassis AC Sockets (TOR Switches) ON

ChassisResponse SetACSocketPowerStateOn(uint portNo)

https://localhost:8000/SetACSocketPowerStateOn?portNo=1

Usage scenario:
This API turns the chassis AC sockets (TOR switches) ON.

The AC socket power state refers to the active power state of the COM ports (and therefore to the power state of the device connected to the port) on the Chassis Manager. For example, the TOR switch is connected to COM1 (port number 1).

Power ON/OFF APIs for the AC sockets makes it possible to remotely power-reset the device The power ON/OFF APIs are also used for enabling/disabling.

Input parameters:

- portNo (port number of the AC socket)

Sample response:

```
<ChassisResponse
xmlns="http://schemas.datacontract.org/2004/07/Microsoft.GFS.WCS.Contracts
" xmlns:i="http://www.w3.org/2001/XMLSchema-instance">
  <completionCode>Success</completionCode>
  <statusDescription></statusDescription>
  <apiVersion>1</apiVersion>
  </ChassisResponse>
```

6.6.32 Turns Chassis AC Sockets (TOR Switches) OFF

ChassisResponse SetACSocketPowerStateOff(uint portNo)

https://localhost:8000/SetACSocketPowerStateOff?portNo=2

Usage scenario:
This API turns the chassis AC sockets (TOR switches) OFF.

The AC socket power state refers to the active power state of the COM ports (and therefore to the power state of the device connected to the port) on the Chassis Manager. For example, the TOR switch is connected to COM1 (port number 1).

Power ON/OFF APIs for the AC sockets makes it possible to remotely power-reset the device The power ON/OFF APIs are also used for enabling/disabling.

Input parameters:

- portNo (port number of the AC socket)

Sample response:

```
<ChassisResponse
xmlns="http://schemas.datacontract.org/2004/07/Microsoft.GFS.WCS.Contracts
" xmlns:i="http://www.w3.org/2001/XMLSchema-instance">
  <completionCode>Success</completionCode>
  <statusDescription></statusDescription>
  <apiVersion>1</apiVersion>
  </ChassisResponse>
```

6.6.33 Gets Status of Chassis AC Sockets (TOR Switches)

ACSocketStateResponse GetACSocketPowerState(uint portNo)

https://localhost:8000/GetACSocketPowerState?portNo=1

Usage scenario:
This API gets the status of the chassis AC sockets (TOR switches).

The AC socket power state refers to the active power state of the COM ports (and therefore to the power state of the device connected to the port) on the Chassis Manager. For example, the TOR switch is connected to COM1 (port number 1).

Power ON/OFF APIs for the AC sockets makes it possible to remotely power-reset the device The power ON/OFF APIs are also used for enabling/disabling.

Input parameters:

- portNo (port number of the AC socket)

Sample response:


```
<ACSocketStateResponse
xmlns="http://schemas.datacontract.org/2004/07/Microsoft.GFS.WCS.Contracts
" xmlns:i="http://www.w3.org/2001/XMLSchema-instance">
  <completionCode>Success</completionCode>
  <statusDescription></statusDescription>
  <apiVersion>1</apiVersion>
  <portNo>1</portNo>
  <powerState>ON</powerState>
  </ACSocketStateResponse>
```


6.6.34 Starts Serial Session to Blade

StartSerialResponse StartBladeSerialSession(int bladeId)

https://localhost:8000/StartBladeSerialSessions?bladeId=1

Usage scenario:
This API starts a serial session to a blade.

Users might want to open a serial session to a blade to debug, to view blade boot messages, or to execute BIOS commands. A provided VT100 console client continuously polls the blade for serial session data (using the ReceiveBladeSerialData API). Any user command entered using the VT100 console is sent to the blade (using the SendBladeSerialData API).

Note that this session might close unexpectedly if there are any simultaneous IPMI commands issued to any of the chassis blades or if the session is inactive (no SendBladeSerialData request from client) for more than five minutes.

Input parameters:

- bladeId (blade index 1-48)

Sample response:

```
StartSerialResponse.sessionToken: 11234
StartSerialResponse.CompletionCode: success
```

6.6.35 Stop Serial Session to Blade

ChassisResponse StopBladeSerialSession(int bladeId, string sessionToken, bool forceKill=false)

https://localhost:8000/StopBladeSerialSessions?bladeId=1

Usage scenario:
This API stops a serial session to a blade.

Users might want to open a serial session to a blade to debug, to view blade boot messages, or to execute BIOS commands. A provided VT100 console client continuously polls the blade for serial session data (using the ReceiveBladeSerialData API). Any user command entered using the VT100 console is sent to the blade (using the SendBladeSerialData API).

Note that this session might close unexpectedly if there are any simultaneous IPMI commands issued to any of the chassis blades (when the config. parameter forceKill is set to true) or if the session is inactive (no SendBladeSerialData request from client) for more than five minutes. However, when forceKill is set to false, an already existing blade serial console session will not be interrupted, and an incoming IPMI command will be ignored.

Input parameters:

- bladeId (blade index 1-48)
- sessionToken (generated as part of the StartBladeSerialSession API)
- forceKill (true or false with semantics explained above)

Sample response:

```
<completionCode>Success</completionCode>
<statusDescription></statusDescription>
```

6.6.36 Sends Data to Blade Serial Device

ChassisResponse SendBladeSerialData(int bladeId, string sessionToken, byte[] data)

https://localhost:8000/SendBladeSerialData?bladeId=1?sessionToken?

Usage scenario:
This API sends the data entered by user on the serial console to the blade serial device (internal API used by the serial-client-proxy)

Users might want to open a serial session to a blade to debug, to view blade boot messages, or to execute BIOS commands. A provided VT100 console client continuously polls the blade for serial session data (using the ReceiveBladeSerialData API). Any user command entered using the VT100 console is sent to the blade (using the SendBladeSerialData API).

Note that this session might close unexpectedly if there are any simultaneous IPMI commands issued to any of the chassis blades or if the session is inactive (no SendBladeSerialData request from client) for more than five minutes.

Input parameters:

- bladeId (blade index 1-48)
- sessionToken (generated as part of the StartBladeSerialSession API)
- data (data to be sent)

Sample response:

```
ChassisResponse .completionCode: Success
```

6.6.37 Receives Data from Blade

SerialDataResponse ReceiveBladeSerialData(int bladeId, string sessionToken)

https://localhost:8000/ReceiveBladeSerialData?bladeId=1?sessionToken?

Usage scenario:
This API receives data from the blade (internal API used by the serial-client-proxy).

Users might want to open a serial session to a blade to debug, to view blade boot messages, or to execute BIOS commands. A provided VT100 console client continuously polls the blade for serial session data (using the ReceiveBladeSerialData API). Any user command entered using the VT100 console is sent to the blade (using the SendBladeSerialData API).

Note that this session might close unexpectedly if there are any simultaneous IPMI commands issued to any of the chassis blades or if the session is inactive (no SendBladeSerialData request from client) for more than five minutes.

Input parameters:

- bladeId (blade index 1-48)
- sessionToken (token id generated as part of the StartBladeSerialSession API)

Sample response:

```
SerialDataResponse .bladeResponse.bladeCompletionCode: Success
SerialDataResponse .data
```

6.6.38 Starts Serial Port Console

StartSerialResponse StartSerialPortConsole(int portId, int sessionTimeoutInSecs, int deviceTimeoutInMsecs);

https://localhost:8000/StartSerialPortConsole?PorId=3

Usage scenario:
This API is used to open a serial-port console terminal to serial devices that are connected to the Chassis Manager (for example, TOR network switches). Note that the serial console might close if session is inactive (no SendSerialPortData request from client) for more than two

minutes.

A provided VT100 console client continuously polls the serial device for data (using the ReceiveSerialPortData API). Any user command entered using the VT100 console is sent to the device (using the SendSerialPortData API).

Input parameters:

- portId (com1-com2)
- SessionTimeoutInSecs(Optional)
- DeviceTimeoutInMsecs(Optional)

Sample response:

```
StartSerialResponse.sessionToken: 32656
StartSerialResponse.CompletionCode: success
```

6.6.39 Stops Serial Port Console

ChassisResponse StopSerialPortConsole(int portId, string sessionToken, bool forceKill);

https://localhost:8000/StopSerialPortConsole?PorId=1

Usage scenario:
This API is used to close a serial-port console terminal to serial devices that are connected to the Chassis Manager (for example, TOR network switches). Note that the serial console might close if session is inactive (no SendSerialPortData request from client) for more than two minutes.

A provided VT100 console client continuously polls the serial device for data (using the ReceiveSerialPortData API). Any user command entered using the VT100 console is sent to the device (using the SendSerialPortData API).

ForceKill option when set to true will kill all existing sessions to the port.

Input parameters:

- portId (com1-com2)
- sessionToken
- forceKill

Sample response:

```
ChassisResponse.CompletionCode: success
```

6.6.40 Sends Serial Port Data

ChassisResponse SendSerialPortData(string portId, string sessionToken, byte[] data)

https://localhost:8000/SendSerialPortData?portId=1?sessionToken?

Sends the serial data to the blade

Usage scenario:
This is an internal API used for sending data from the user serial-client terminal to serial devices connected to the Chassis Manager (for example, user commands executed on the TOR network switch serial console). Note that the serial console might close if session is inactive(no SendSerialPortData request from client) for more than two minutes.

A provided VT100 console client continuously polls the serial device for data (using the ReceiveSerialPortData API). Any user command entered using the VT100 console is sent to the device (using the SendSerialPortData API).

Input parameters:

- portId (com1-com2)
- sessionToken (token id)
- data (data to be sent)

Sample response:

```
ChassisResponse.CompletionCode: Success
```

6.6.41 Receives Serial Port Data

SerialDataResponse ReceiveSerialPortData(string portId, string sessionToken)

https://localhost:8000/ReceiveSerialPortData?portId=1?sessionToken?

Usage scenario:
This is an internal API used for receiving data on the terminal from serial devices connected to the Chassis Manager (for example, user commands executed on the TOR network switch serial console). Note that the serial console might close if session is inactive(no SendSerialPortData request from client) for more than two minutes.

A provided VT100 console client continuously polls the serial device for data (using the ReceiveSerialPortData API). Any user command entered using the VT100 console is sent to the device (using the SendSerialPortData API).

Input parameters:

- portId (com1-com2)
- sessionToken (token id)

Sample response:

```
ChassisResponse.CompletionCode: Success
SerialData.data
```

6.6.42 Reads the Chassis Log (with timestamp parameter)

LogResponse ReadChassisLog()

LogResponse ReadChassisLogWithTimestamp(Datetime startTimestamp, Datetime endTimestamp)

https://localhost:8000/ReadChassisLog?

Usage scenario:
This API reads the chassis log.

The chassis log contains information about the various alerts and warning messages associated with devices connected to the Chassis Manager (for example, blades overheating, and fan/PSU failure). The chassis log also contains user audit information, such as timestamp/activity performed by the user.

Input parameters:

- startTimestamp (read log from the start timestamp, optional)
- endTimestamp (read log till the given end timestamp, optional)

Sample response:


```
- <ChassisLogResponse
xmlns="http://schemas.datacontract.org/2004/07/Microsoft.GFS.WCS.Contracts
" xmlns:i="http://www.w3.org/2001/XMLSchema-instance">
  <completionCode>Success</completionCode>
  <statusDescription></statusDescription>
  <apiVersion>1</apiVersion>
- <logEntries>
- <LogEntry>
  <eventDescription>CM_B1B02_2\adminB1B02,Invoked </eventDescription>
  <eventTime>2012-09-11T21:04:33.024</eventTime>
  </LogEntry>
- <LogEntry>
  <eventDescription>CM_B1B02_2\adminB1B02,Invoked </eventDescription>
  <eventTime>2012-09-11T21:04:33.117</eventTime>
  </LogEntry>
...
- <LogEntry>
  <eventDescription>CM_B1B02_2\adminB1B02,Invoked
ReadChassisLog()</eventDescription>
  <eventTime>2012-09-11T21:20:03.366</eventTime>
  </LogEntry>
  </logEntries>
  </ChassisLogResponse>
```


6.6.43 Clears the Chassis Log

ClearResponse ClearChassisLog()

https://localhost:8000/ClearChassisLog?

Usage scenario:

This API clears the chassis log. Users must periodically clear the consumed log entries because there are size restrictions on the Chassis Manager storage space.

Input parameters:

- None

Sample response:


```
<ChassisResponse
xmlns="http://schemas.datacontract.org/2004/07/Microsoft.GFS.WCS.Contracts
" xmlns:i="http://www.w3.org/2001/XMLSchema-instance">
  <completionCode>Success</completionCode>
  <statusDescription></statusDescription>
  <apiVersion>1</apiVersion>
  </ChassisResponse>
```


6.6.44 Reads Log from Blade (with timestamp parameter)

LogResponse ReadBladeLog(int bladeId)

LogResponse ReadBladeLogWithTimestamp(int bladeId, uint logType, Datetime startTimestamp, Datetime endTimestamp)

https://localhost:8000/ReadBladeLog?bladeId=1

Usage scenario:
This API reads the log of a blade. Blade logs (system event logs) contain information about events, warning, and alerts pertaining to that blade (for example, thermal throttling of blades because of overheating).

Input parameters:

- bladeId (blade index, 1-48)
- startTimestamp (read log from this given start timestamp, optional)
- endTimestamp (read log till the given end timestamp, optional)

Sample response:


```
- <ChassisLogResponse
xmlns="http://schemas.datacontract.org/2004/07/Microsoft.GFS.WCS.Contracts
" xmlns:i="http://www.w3.org/2001/XMLSchema-instance">
  <completionCode>Failure</completionCode>
  <statusDescription>The CM did not respond</statusDescription>
  <apiVersion>1</apiVersion>
- <logEntries>
- <LogEntry>

<eventDescription>Sensor_SpecificDrive_Slot2433328Assertion111</eventDescr
iption>
```

```
  <eventTime>2012-08-23T14:35:21</eventTime>
  </LogEntry>
- <LogEntry>

<eventDescription>Sensor_SpecificDrive_Slot2443328Assertion111</eventDescr
iption>
  <eventTime>2012-08-23T14:35:21</eventTime>
  </LogEntry>
...
- <LogEntry>
  <eventDescription>DiscreteTemperature187257Desertion3</eventDescription>
  <eventTime>2012-08-23T16:11:08</eventTime>
  </LogEntry>
  </logEntries>
  </ChassisLogResponse>
```


6.6.45 Clears Log from Blade

BladeResponse ClearBladelog(int bladeId)

https://localhost:8000/ClearBladeLog?bladeId=1

Usage scenario:
This API clears the log from a blade.

Blade logs (system event logs) contain information about events, warning, and alerts pertaining to that blade (for example, thermal throttling of blades because of overheating).

Input parameters:

- bladeId (blade index, 1-48)

Sample response:


```
- <BladeResponse
xmlns="http://schemas.datacontract.org/2004/07/Microsoft.GFS.WCS.Contracts
" xmlns:i="http://www.w3.org/2001/XMLSchema-instance">
  <completionCode>Success</completionCode>
  <statusDescription></statusDescription>
  <apiVersion>1</apiVersion>
  <bladeNumber>1</bladeNumber>
  </BladeResponse>
```


6.6.46 Gets Power Reading from Blade

BladePowerReadingResponse GetBladePowerReading(int bladeId)

https://localhost:8000/GetBladePowerReading?bladeId=1

Usage scenario:
This API is used to get the power reading of a blade. It can be used for monitoring or for other power-control mechanisms (see the SetBladePowerLimit() API).

Input parameters:

- bladeId (blade index, 1-48)

Sample response:

```
- <BladePowerReadingResponse
xmlns="http://schemas.datacontract.org/2004/07/Microsoft.GFS.WCS.Contracts
" xmlns:i="http://www.w3.org/2001/XMLSchema-instance">
  <completionCode>Success</completionCode>
  <statusDescription></statusDescription>
  <apiVersion>1</apiVersion>
- <bladeResponse>
  <completionCode>Success</completionCode>
  <statusDescription></statusDescription>
  <apiVersion>1</apiVersion>
  <bladeNumber>1</bladeNumber>
  </bladeResponse>
  <powerReading>308</powerReading>
  </BladePowerReadingResponse>
```

6.6.47 Gets Power Readings from All Blades

`GetAllBladesPowerReadingResponse` GetAllBladesPowerReading()

https://localhost:8000/GetAllBladesPowerReading?

Usage scenario:
This API can be used for monitoring or for other power-control mechanisms (see the SetBladePowerLimit() API).

Note that when multiple users are actively trying to access/modify the same state of a single blade or of different blades, ordering is not guaranteed.

Input parameters:

- None

Sample response:

```
- <GetAllBladesPowerReadingResponse
xmlns="http://schemas.datacontract.org/2004/07/Microsoft.GFS.WCS.Contracts
" xmlns:i="http://www.w3.org/2001/XMLSchema-instance">
- <BladePowerReadingResponse>
  <completionCode>Success</completionCode>
  <statusDescription></statusDescription>
```

```
  <apiVersion>1</apiVersion>
- <bladeResponse>
  <completionCode>Success</completionCode>
  <statusDescription></statusDescription>
  <apiVersion>1</apiVersion>
  <bladeNumber>1</bladeNumber>
  </bladeResponse>
  <powerReading>308</powerReading>
  </BladePowerReadingResponse>
- <BladePowerReadingResponse>
  <completionCode>Success</completionCode>
  <statusDescription></statusDescription>
  <apiVersion>1</apiVersion>
- <bladeResponse>
  <completionCode>Success</completionCode>
  <statusDescription></statusDescription>
  <apiVersion>1</apiVersion>
  <bladeNumber>2</bladeNumber>
  </bladeResponse>
  <powerReading>311</powerReading>
  </BladePowerReadingResponse>
...
- <BladePowerReadingResponse>
  <completionCode>Success</completionCode>
  <statusDescription></statusDescription>
  <apiVersion>1</apiVersion>
- <bladeResponse>
  <completionCode>Success</completionCode>
  <statusDescription></statusDescription>
  <apiVersion>1</apiVersion>
  <bladeNumber>24</bladeNumber>
  </bladeResponse>
  <powerReading>310</powerReading>
  </BladePowerReadingResponse>
  </GetAllBladesPowerReadingResponse>
```


6.6.48 Gets Power Limit of Blade

GetBladeLimitResponse GetBladePowerLimit(int bladeId)

https://localhost:8000/GetBladePowerLimit?bladeId=1

Usage scenario:

This API can be used to get the power limit that is set on a particular blade.

Note: This API internally uses Intel ME for power limiting and reporting. Note that GetBladePowerLimit will fail with completion code 0x80 if there isn't an active SetBladePowerLimit. So use GetBladePowerLimit after SetBladePowerLimit.

Input parameters:

- bladeId (blade index, 1-48)

Sample response:


```
- <BladePowerLimitResponse
xmlns="http://schemas.datacontract.org/2004/07/Microsoft.GFS.WCS.Contracts
" xmlns:i="http://www.w3.org/2001/XMLSchema-instance">
  <completionCode>Success</completionCode>
  <statusDescription></statusDescription>
  <apiVersion>1</apiVersion>
- <bladeResponse>
  <completionCode>Success</completionCode>
  <statusDescription></statusDescription>
  <apiVersion>1</apiVersion>
  <bladeNumber>1</bladeNumber>
  </bladeResponse>
  <powerReading>750</powerReading>
  </BladePowerLimitResponse>
```


6.6.49 Gets Power Limit of All Blades

GetAllBladesPowerLimitResponse GetAllBladesPowerLinit()

https://localhost:8000/GetAllBladesPowerLimit?

Usage scenario:
This API can be used to get the power limit that is set on all blades in a chassis.

Note: When multiple users are actively trying to access/modify the same state of a single blade or of different blades, ordering is not guaranteed.

This API internally uses Intel ME for power limiting and reporting. Note that GetBladePowerLimit will fail with completion code 0x80 if there isn't an active SetBladePowerLimit. So use GetBladePowerLimit after SetBladePowerLimit.

Input parameters:

- None

Sample response:

```
- <GetAllBladesPowerLimitResponse
xmlns="http://schemas.datacontract.org/2004/07/Microsoft.GFS.WCS.Contracts
" xmlns:i="http://www.w3.org/2001/XMLSchema-instance">
- <BladePowerLimitResponse>
  <completionCode>Success</completionCode>
  <statusDescription></statusDescription>
  <apiVersion>1</apiVersion>
- <bladeResponse>
  <completionCode>Success</completionCode>
  <statusDescription></statusDescription>
  <apiVersion>1</apiVersion>
  <bladeNumber>1</bladeNumber>
  </bladeResponse>
  <powerReading>750</powerReading>
  </BladePowerLimitResponse>
- <BladePowerLimitResponse>
  <completionCode>Success</completionCode>
  <statusDescription></statusDescription>
  <apiVersion>1</apiVersion>
- <bladeResponse>
  <completionCode>Success</completionCode>
  <statusDescription></statusDescription>
  <apiVersion>1</apiVersion>
  <bladeNumber>2</bladeNumber>
  </bladeResponse>
  <powerReading>750</powerReading>
  </BladePowerLimitResponse>
...
- <BladePowerLimitResponse>
  <completionCode>Success</completionCode>
  <statusDescription></statusDescription>
  <apiVersion>1</apiVersion>
- <bladeResponse>
  <completionCode>Success</completionCode>
  <statusDescription></statusDescription>
  <apiVersion>1</apiVersion>
  <bladeNumber>24</bladeNumber>
  </bladeResponse>
  <powerReading>750</powerReading>
  </BladePowerLimitResponse>
  </GetAllBladesPowerLmitResponse>
```

6.6.50 Sets Power Limit on Blade

BladeResponse SetBladePowerLimit(int bladeId, double powerLimitInWatts)

https://localhost:8000/SetBladePowerLimit?bladeId=1&powerLimitInWatts=750

Usage scenario:
This API is used to set the power limit for a blade; if the user wants to set the same power limit for all the blades, the SetAllBladesPowerLimit() API can be used. SetBladePowerLimitOn API has to be executed to actually realize the set power limit in the device.
Power limits can be set for a variety of reasons, including energy savings and under-provisioning (consolidating more servers under a power hierarchy).

The min/max power limit is specified in app.config in the Chassis Manager.

Input parameters:

- bladeId (blade index 1-48
- powerLimitInWatts

Sample response:


```
- <BladeResponse
xmlns="http://schemas.datacontract.org/2004/07/Microsoft.GFS.WCS.Contracts
" xmlns:i="http://www.w3.org/2001/XMLSchema-instance">
  <completionCode>Success</completionCode>
  <statusDescription></statusDescription>
  <apiVersion>1</apiVersion>
  <bladeNumber>1</bladeNumber>
  </BladeResponse>
```


6.6.51 Sets Power Limit on All Blades

AllBladesResponse SetAllBladesPowerLimit(double powerLimitInWatts)

https://localhost:8000/SetAllBladesPowerLimit?powerLimitInWatts=750

Usage scenario:
This API is used to set the power limit for all blades in a chassis; if the user wants to set heterogeneous power limits, the SetBladePowerLimit() API can be used Power limits can be set for a variety of reasons, including energy savings and under-provisioning (consolidating more servers under a power hierarchy). SetBladePowerLimitOn API has to be executed to actually realize the set power limit in the device.
Note that when multiple users are actively trying to access/modify the same state of a single blade or of different blades, ordering is not guaranteed.

Input parameters:

- powerLimitInWatts

Sample response:


```
- <AllBladesResponse
xmlns="http://schemas.datacontract.org/2004/07/Microsoft.GFS.WCS.Contracts
" xmlns:i="http://www.w3.org/2001/XMLSchema-instance">
  <completionCode>Success</completionCode>
  <statusDescription></statusDescription>
  <apiVersion>1</apiVersion>
- <BladeResponse>
  <completionCode>Success</completionCode>
  <statusDescription></statusDescription>
  <apiVersion>1</apiVersion>
  <bladeNumber>1</bladeNumber>
  </BladeResponse>
- <BladeResponse>
  <completionCode>Success</completionCode>
  <statusDescription></statusDescription>
  <apiVersion>1</apiVersion>
  <bladeNumber>2</bladeNumber>
  </BladeResponse>
...
- <BladeResponse>
  <completionCode>Success</completionCode>
  <statusDescription></statusDescription>
  <apiVersion>1</apiVersion>
  <bladeNumber>24</bladeNumber>
  </BladeResponse>
  </AllBladesResponse>
```


6.6.52 Activates Blade Power Limit

BladeResponse SetBladePowerLimitOn(int bladeId)

https://localhost:8000/SetBladePowerLimitOn?bladeId=1

Usage scenario:
This API activates the power limit for a blade and enables power throttling.

Power limits can be set for a variety of reasons, including energy savings and under-provisioning (consolidating more servers under a power hierarchy).

Input parameters:

- bladeId (blade index 1-48

Sample response:


```
<BladeResponse
```

```
xmlns="http://schemas.datacontract.org/2004/07/Microsoft.GFS.WCS.Contracts
" xmlns:i="http://www.w3.org/2001/XMLSchema-instance">
  <completionCode>Success</completionCode>
  <statusDescription></statusDescription>
  <apiVersion>1</apiVersion>
  <bladeNumber>1</bladeNumber>
  </BladeResponse>
```


6.6.53 Activates Power Limit on All Blades

AllBladesResponse SetAllBladesPowerLimitOn()

https://localhost:8000/SetAllBladesPowerLimitOn?

Usage scenario:
This API activates the power limit for all blades in a chassis and enables power throttling.

Power limits can be set for a variety of reasons, including energy savings and under-provisioning (consolidating more servers under a power hierarchy).

Input parameters:

- None

Sample response:


```
- <AllBladesResponse
xmlns="http://schemas.datacontract.org/2004/07/Microsoft.GFS.WCS.Contracts
" xmlns:i="http://www.w3.org/2001/XMLSchema-instance">
  <completionCode>Success</completionCode>
  <statusDescription></statusDescription>
  <apiVersion>1</apiVersion>
- <BladeResponse>
  <completionCode>Success</completionCode>
  <statusDescription></statusDescription>
  <apiVersion>1</apiVersion>
  <bladeNumber>1</bladeNumber>
  </BladeResponse>
- <BladeResponse>
  <completionCode>Success</completionCode>
  <statusDescription></statusDescription>
  <apiVersion>1</apiVersion>
  <bladeNumber>2</bladeNumber>
  </BladeResponse>
...
- <BladeResponse>
```

```
  <completionCode>Success</completionCode>
  <statusDescription></statusDescription>
  <apiVersion>1</apiVersion>
  <bladeNumber>24</bladeNumber>
  </BladeResponse>
  </AllBladesResponse>
```


6.6.54 Deactivates Blade Power Limit

BladeResponse SetBladePowerLimitOff(int bladeId)

https://localhost:8000/SetBladePowerLimitOff?bladeId=1

Usage scenario:
This API deactivates the power limit for a blade and disables power throttling.

Power limits can be set for a variety of reasons, including energy savings and under-provisioning (consolidating more servers under a power hierarchy).

Input parameters:

- bladeId (blade index 1-48)

Sample response:


```
<BladeResponse
xmlns="http://schemas.datacontract.org/2004/07/Microsoft.GFS.WCS.Contracts
" xmlns:i="http://www.w3.org/2001/XMLSchema-instance">
  <completionCode>Success</completionCode>
  <statusDescription></statusDescription>
  <apiVersion>1</apiVersion>
  <bladeNumber>1</bladeNumber>
  </BladeResponse>
```


6.6.55 Deactivates Power Limit on All Blades

AllBladesResponse SetAllBladesPowerLimitOff()

https://localhost:8000/SetAllBladesPowerLimitOff?

Usage scenario:
This API deactivates the power limit for all blades in a chassis and disables power throttling.

Power limits can be set for a variety of reasons, including energy savings and under-provisioning (consolidating more servers under a power hierarchy).

Input parameters:

- None

Sample response:

```
- <AllBladesResponse
xmlns="http://schemas.datacontract.org/2004/07/Microsoft.GFS.WCS.Contracts
" xmlns:i="http://www.w3.org/2001/XMLSchema-instance">
  <completionCode>Success</completionCode>
  <statusDescription></statusDescription>
  <apiVersion>1</apiVersion>
- <BladeResponse>
  <completionCode>Success</completionCode>
  <statusDescription></statusDescription>
  <apiVersion>1</apiVersion>
  <bladeNumber>1</bladeNumber>
  </BladeResponse>
- <BladeResponse>
  <completionCode>Success</completionCode>
  <statusDescription></statusDescription>
  <apiVersion>1</apiVersion>
  <bladeNumber>2</bladeNumber>
  </BladeResponse>
...
- <BladeResponse>
  <completionCode>Success</completionCode>
  <statusDescription></statusDescription>
  <apiVersion>1</apiVersion>
  <bladeNumber>24</bladeNumber>
  </BladeResponse>
  </AllBladesResponse>
```


6.6.56 Gets Chassis Controller Network Properties

ChassisNetworkPropertiesResponse GetChassisNetworkProperties()

https://localhost:8000/GetChassisNetworkProperties?

Usage scenario:
This API gets the chassis controller network properties including MAC address, IP address, subnet mask, DHCP Enabled/Disabled.

Note that Microsoft does not support setting network properties of the Chassis Manager and encourages users to use standard Windows interface/APIs for this.

Input parameters:

- None

Sample response:

```
- <ChassisNetworkPropertiesResponse
xmlns="http://schemas.datacontract.org/2004/07/Microsoft.GFS.WCS.Contracts
" xmlns:i="http://www.w3.org/2001/XMLSchema-instance">
  <completionCode>Success</completionCode>
  <statusDescription></statusDescription>
  <apiVersion>1</apiVersion>
- <ChassisNetworkProperty>
<completionCode>Success</completionCode>
  <statusDescription></statusDescription>
  <apiVersion>1</apiVersion>
  <ipAddress>192.168.100.59</ipAddress>
  <macAddress>08:9E:01:18:0C:08</macAddress>
  <dhcpEnabled>true</dhcpEnabled>
  <dhcpServer>192.168.100.8</dhcpServer>
  <dnsAddress i:nil="true" />
  <dnsDomain>wcs.lab</dnsDomain>
  <dnsHostName>DVTCM03</dnsHostName>
  <gatewayAddress i:nil="true" />
  <subnetMask>255.255.255.0</subnetMask>
  </ChassisNetworkProperty>
  </ChassisNetworkPropertiesResponse>
```


6.6.57 Adds New Chassis Controller User

ChassisResponse AddChassisControllerUser(string userName, string passwordString, WCSSecurityRole role)

https://localhost:8000/AddChassisControllerUser?userName=xxx&passwordString=yyyy&role=1

Usage scenario:
This API is used to add a new chassis controller user with a specified password with privileges for accessing the Chassis Manager command line interface. The role parameter indicates the requested WCS user privilege level for this user (see below).

```
public enum WCSSecurityRole : int
{
    // WCS Roles
    WcsCmAdmin = 2,
    WcsCmOperator = 1,
    WcsCmUser = 0
}
```

Input parameters:

- userName – username associated with the user
- passwordString
  Password for this user.

Password that do not adhere to standard windows user complexity requirements will result in API failure with appropriate error message thrown.

- role
  Indicates the requested WCS user privilege level for this user.

Sample response:

```
<ChassisResponse
xmlns="http://schemas.datacontract.org/2004/07/Microsoft.GFS.WCS.Contracts
" xmlns:i="http://www.w3.org/2001/XMLSchema-instance">
  <completionCode>Success</completionCode>
  <statusDescription></statusDescription>
  <apiVersion>1</apiVersion>
  </ChassisResponse>
```

6.6.58 Changes Password for Existing Chassis Controller User

ChassisResponse ChangeChassisControllerUserPassword(string userName, string newPassword)

https://localhost:8000/ChangeChassisControllerUserPassword?userName=xxx &newPassword=yyyy

Usage scenario:
Changes the username and password for an existing user in the chassis controller.

Input parameters:

- userName
- newpassword

Sample response:

```
<ChassisResponse
xmlns="http://schemas.datacontract.org/2004/07/Microsoft.GFS.WCS.Contracts
" xmlns:i="http://www.w3.org/2001/XMLSchema-instance">
  <completionCode>Success</completionCode>
  <statusDescription></statusDescription>
  <apiVersion>1</apiVersion>
  </ChassisResponse>
```

6.6.59 Changes Role for Existing Chassis Controller User

ChassisResponse ChangeChassisControllerUserRole(string userName, `WCSSecurityRole role`)
https://localhost:8000/ChangeChassisControllerUserRole?userName=xxx &role=1

Usage scenario:
This API is used to change the role associated with the user (see below).

```
public enum WCSSecurityRole : int
{
    // WCS Roles
```

```
        WcsCmAdmin = 2,
        WcsCmOperator = 1,
        WcsCmUser = 0
    }
```

Input parameters:

- userName
- role

Sample response:


```
<ChassisResponse
xmlns="http://schemas.datacontract.org/2004/07/Microsoft.GFS.WCS.Contracts
" xmlns:i="http://www.w3.org/2001/XMLSchema-instance">
  <completionCode>Success</completionCode>
  <statusDescription></statusDescription>
  <apiVersion>1</apiVersion>
  </ChassisResponse>
```


6.6.60 Removes Existing Chassis Controller User

ChassisResponse RemoveChassisControllerUser(string userName)
https://localhost:8000/RemoveChassisControllerUser?userName=xxx

Usage scenario:
This command is used to remove an existing chassis controller user.

Input parameters:
userName

Sample response:


```
<ChassisResponse
xmlns="http://schemas.datacontract.org/2004/07/Microsoft.GFS.WCS.Contracts
" xmlns:i="http://www.w3.org/2001/XMLSchema-instance">
  <completionCode>Success</completionCode>
  <statusDescription></statusDescription>
  <apiVersion>1</apiVersion>
  </ChassisResponse>
```


6.6.61 Get Health of Chassis

ChassisHealthResponse GetChassisHealth(bool bladeHealth, bool psuHealth, bool fanHealth)

https://localhost:8000/GetChassisHealth

Usage scenario:
This API can be used for status/health monitoring of chassis devices such as blade, psu, and fan. The power supply status is not available during a PSU firmware upgrade process.

The BladeType attribute specified as part of the BladeShellResponse refers to the type of blade, server (compute server), JBod (storage server) or unknown (device not reachable or not populated) .

Input parameters:

- bladeHealth
- psuHealth
- fanHealth
- bladeHealth

Note: If none of the parameters are specified, the API will fetch result for all components.

Sample response:

- <ChassisHealthResponse xmlns="http://schemas.datacontract.org/2004/07/Microsoft.GFS.WCS.Contracts" xmlns:i="http://www.w3.org/2001/XMLSchema-instance">
<completionCode>Success</completionCode>
<statusDescription></statusDescription>
<apiVersion>1</apiVersion>
- <bladeShellCollection>
- <BladeShellResponse>
<completionCode>Success</completionCode>
<statusDescription></statusDescription>
<apiVersion>1</apiVersion>
<bladeId>1</bladeId>
<bladeState>ON</bladeState>
<bladeType>Server</bladeType>
</BladeShellResponse>
- <BladeShellResponse>
<completionCode>Success</completionCode>
<statusDescription></statusDescription>
<apiVersion>1</apiVersion>
<bladeId>2</bladeId>
<bladeState>Healthy</bladeState>
<bladeType>Server</bladeType>
</BladeShellResponse>
…
- <BladeShellResponse>
<completionCode>Success</completionCode>
<statusDescription></statusDescription>
<apiVersion>1</apiVersion>
<bladeId>24</bladeId>
<bladeState>Fail</bladeState>
<bladeType>Unknown</bladeType>

```
</BladeShellResponse>
</bladeShellCollection>
- <fanInfoCollection>
- <FanInfo>
  <completionCode>Success</completionCode>
  <statusDescription></statusDescription>
  <apiVersion>1</apiVersion>
<fanId>1</fanId>
<fanSpeed>4109</fanSpeed>
<isFanHealthy>true</ isFanHealthy >
</FanInfo>
- …
<FanInfo>
  <completionCode>Success</completionCode>
  <statusDescription></statusDescription>
  <apiVersion>1</apiVersion>
<fanId>6</fanId>
<fanSpeed>4094</fanSpeed>
< isFanHealthy >true</ isFanHealthy >
</FanInfo>
</fanInfoCollection>
- <psuInfoCollection>
- <PsuInfo>
  <completionCode>Success</completionCode>
  <statusDescription></statusDescription>
  <apiVersion>1</apiVersion>
<id>1</id>
<powerOut>0</powerOut>
<serialNumber>46-49-51-44-31-32-33-37-30-30-31-30-32-34</serialNumber>
<state>ON</state>
</PsuInfo>
…
- <PsuInfo>
  <completionCode>Success</completionCode>
  <statusDescription></statusDescription>
  <apiVersion>1</apiVersion>
<id>6</id>
<powerOut>0</powerOut>
<serialNumber>46-49-51-44-31-32-33-37-30-30-31-30-35-38</serialNumber>
<state>ON</state>
</PsuInfo>
</psuInfoCollection>

</ChassisHealthResponse>
```

3.62 Get Health of Blade

BladeHealthResponse GetBladeHealth(int bladeId, bool cpuInfo, bool memInfo, bool diskInfo, bool pcieInfo, bool sensorInfo, bool temp, bool fruInfo)

https://localhost:8000/GetBladeHealth?bladeid=2&cpuInfo=true&memInfo=true&diskInfo=true&pcieInfo=true&sensorInfo=true&temp=true&fruInfo=true

Usage scenario:
This API can be used for status/health monitoring of blade components such as cpu, memory, disk (JBOD only), pci, sensor, temperature, and FRU.

Input parameters:

- bladeId
- cpuInfo
- memInfo
- diskInfo
- pcieInfo
- sensorInfo
- temp
- fruInfo

Note: Except for bladeID, other parameters are optional. If none of the other parameters are specified, the API will fetch result for all components.

Sample response:


```
- <BladeHealthResponse
xmlns="http://schemas.datacontract.org/2004/07/Microsoft.GFS.WCS.Contracts" xmlns:i="http://www.w3.org/2001/XMLSchema-instance">
  <completionCode>Success</completionCode>
  <statusDescription></statusDescription>
  <apiVersion>1</apiVersion>
  <assetTag />
- <bladeShell>
  <completionCode>Success</completionCode>
  <statusDescription></statusDescription>
  <apiVersion>1</apiVersion>
  <bladeId>2</bladeId>
  <bladeState>ON</bladeState>
  <bladeType>Server</bladeType>
  </bladeShell>
  <hardwareVersion>T6M</hardwareVersion>
  <JbodDiskInfo i:nil="true" />
  <JbodInfo i:nil="true" />
- <memoryInfo>
- <memoryInfo>
```

```
  <completionCode>Success</completionCode>
  <apiVersion>1</apiVersion>
  <statusDescription></statusDescription>
  <dimm>1</dimm>
  <dimmType>DDR3</dimmType>
  <status>Ok</status>
  <speed>1333</speed>
  <size>8192</size>
  <memVoltage>1.35V</memVoltage>
  </memoryInfo>
- …
- <memoryInfo>
  <completionCode>Success</completionCode>
  <apiVersion>1</apiVersion>
  <statusDescription></statusDescription>
  <dimm>2</dimm>
  <dimmType>DDR3</dimmType>
  <status>Ok</status>
  <speed>1333</speed>
  <size>8192</size>
  <memVoltage>1.35V</memVoltage>
  </MemoryInfo>
…
- <PcieInfo>
- <PCIeInfo>
  <completionCode>Success</completionCode>
  <apiVersion>1</apiVersion>
  <statusDescription></statusDescription>
  <pcieNumber>1</pcieNumber>
  <vendorId>0x0000</vendorId>
  <deviceId>0x0000</deviceId>
  <subSystemId>0x00000000</subSystemId>
  <status>NotPresent</status>
  </PCIeInfo>
- <PCIeInfo>
  <completionCode>Success</completionCode>
  <apiVersion>1</apiVersion>
  <statusDescription></statusDescription>
  <pcieNumber>2</pcieNumber>
  <vendorId>0x15B3</vendorId>
  <deviceId>0x1003</deviceId>
  <subSystemId>0x8995152D</subSystemId>
  <status>Present</status>
  </PCIeInfo>
- <processorInfo>
- <processorInfo>
```

```
<completionCode>Failure</completionCode>
<statusDescription>Device did not return result</statusDescription>
<apiVersion>1</apiVersion>
<frequency>0</frequency>
<procId>0</procId>
<procType i:nil="true" />
<state i:nil="true" />
</processorInfo>
<productType />
<sensors />
<serialNumber>QTFCTM2350003</serialNumber>
</BladeHealthResponse>
```

6.6.62 Get Next Boot Device

BootResponse GetNextBoot(int bladeId)

https://localhost:8000/GetNextBoot?bladeid=2

Usage scenario:
This API gets the boot device of the blade after the next reboot.

This command does not reflect the BIOS boot order. The SetNextboot command acts as an interrupt before the BIOS boot order is initialized. If the SetNextBoot order is flagged with persistence it interrupts boot every time, until manual user intervention to change the BIOS manually (then the SetNextBoot is overridden and must be executed again). If it is not flagged with persistence the command will do a 1 time volatile override (if you hard power cycle volatile memory is lost, hence it needs to be a BMC power cycle).

Input parameters:
bladeId

Sample response:

```
<BootResponse
xmlns="http://schemas.datacontract.org/2004/07/Microsoft.GFS.WCS.Contr
acts" xmlns:i="http://www.w3.org/2001/XMLSchema-instance">
  <completionCode>Success</completionCode>
  <statusDescription></statusDescription>
  <apiVersion>1</apiVersion>
  <nextBootString>NoOverRide</nextBootString>
  </BootResponse>
```

6.6.63 Set Next Boot Device

BootResponse SetNextBoot(int bladeId, BladeBootType bootType, bool uefi, bool persistent, int bootInstance)

https://localhost:8000/SetNextBoot?bladeid=2&bootType=2&uefi=false&persistent=false

Usage scenario:
This API sets the boot device of the blade upon its next reboot. The boot device can be set using the bootType parameter from the list of devices specified in the enum below.

This command does not change the BIOS boot order; itd acts as an interrupt before the BIOS boot order is initialized. If the SetNextBoot order is flagged with persistence it interrupts boot every time, until manual user intervention to change the BIOS manually (then the SetNextBoot is overridden and must be executed again). If it is not flagged with persistence the command will do a one-time volatile override (if you hard power cycle volatile memory is lost, hence it needs to be a BMC power cycle).

```
/// <summary>
/// Boot type for blades.
/// The boot should follow soon (within one minute) after the boot type is set.
/// </summary>
public enum BladeBootType : int
{
    Unknown = 0,
    NoOverride = 1,
    ForcePxe = 2,
    ForceDefaultHdd = 3,
    ForceIntoBiosSetup = 4,
    ForceFloppyOrRemovable = 5
}
```

- **Input Parameters:**bladeId
- bootType: type of the boot device
- uefi: true sets UEFI BIOS, false sets legacy BIOS
- persistent: true sets it for all subsequent reboots, false sets it just for the next reboot

bootInstance: the instance of the boot device (for example, the second NIC card)

Sample response:


```
<BootResponse
xmlns="http://schemas.datacontract.org/2004/07/Microsoft.GFS.WCS.Contracts"
xmlns:i="http://www.w3.org/2001/XMLSchema-instance">

  <completionCode>Success</completionCode>

  <statusDescription></statusDescription>

  <apiVersion>1</apiVersion>

  <nextBootString>ForcePxe</nextBootString>

  </BootResponse>
```


6.6.64 Get Service Version

ServiceVersionResponse GetServiceVersion()

https://localhost:8000/GetServiceVersion?

Usage scenario:
This API gets the version information for Chassis Manager service.
Here the service version contains the following four values:

- Major Version
- Minor Version
- Build Number
- Revision

```
/// <summary>
/// Service version information
/// </summary>
[DataContract]
public class ServiceVersionResponse : ChassisResponse
{
    [DataMember]
    public string serviceVersion;

       }
```

Sample response:

```
<ServiceVersionResponse
xmlns="http://schemas.datacontract.org/2004/07/Microsoft.GFS.WCS.Contracts"
xmlns:i="http://www.w3.org/2001/XMLSchema-instance">
  <completionCode>Success</completionCode>
  <statusDescription></statusDescription>
  <apiVersion>1</apiVersion>
            <serviceVersion>1.2.0.0</serviceVersion>
  </ServiceVersionResponse>
```

6.6.65 Reset PSU

ChassisResponse ResetPsu(int pusId)

https://localhost:8000/ResetPsu?psuId=1

Usage scenario:
This API turns the power supply off, and then on again.

Input parameters:

- psuId

6.6.66 Get Chassis Manager Asset Info

ChassisAssetInfoResponse GetChassisManagerAssetInfo()

https://localhost:8000/GetChassisManagerAssetInfo

Usage scenario:

This API gets the FRU information of the Chassis Manager.

Input parameters:

- None

6.6.67 Get Power Distribution Board (PDB) Asset Info

ChassisAssetInfoResponse GetPDBAssetInfo ()

https://localhost:8000/GetPDBAssetInfo

Usage Scenario:

This API gets the FRU information of the power distribution board (PDB).

Input parameters:

- None

6.6.68 Get Blade Asset Info

BladeAssetInfoResponse GetBladeAssetInfo(int bladeId)

https://localhost:8000/ GetBladeAssetInfo?bladeId=1

Usage scenario:

This API gets the FRU information of the specified blade.

Input parameters:

- bladeId (blade index 1-48)

Sample response

<?xml version="1.0"?>

-<BladeAssetInfoResponse xmlns="http://schemas.datacontract.org/2004/07/Microsoft.GFS.WCS.Contracts" xmlns:i="http://www.w3.org/2001/XMLSchema-instance"><completionCode>Success</completionCode><apiVersion>1</apiVersion><statusDescription /><bladeNumber>2</bladeNumber><chassisAreaPartNumber>X873021-001</chassisAreaPartNumber><chassisAreaSerialNumber>QTFCTM2490136</chassisAreaSerialNumber><boardAreaManufacturerName>Microsoft</boardAreaManufacturerName><boardAreaManufacturerDate>12/6/2012 10:36:00 AM</boardAreaManufacturerDate><boardAreaProductName>WCS SB EN</boardAreaProductName><boardAreaSerialNumber>MH824800046</boardAreaSerialNumber><boardAreaPartNumber>X873096-001</boardAreaPartNumber><productAreaManufactureName>Microsoft</productAreaManufactureName><productAreaProductName>WCS Mt.Glacier</productAreaProductName><productAreaPartModelNumber>X873095-

001</productAreaPartModelNumber><productAreaProductVersion>1.0</productAreaProductVersion><productAreaSerialNumber>MH824800046</productAreaSerialNumber><productAreaAssetTag>N/A</productAreaAssetTag><manufacturer/><multiRecordFields xmlns:a="http://schemas.microsoft.com/2003/10/Serialization/Arrays"/></BladeAssetInfoResponse>

6.6.69 Set Chassis Manager Asset Info

MultiRecordResponse SetChassisManagerAssetInfo(string payLoad)

https://localhost:8000/ SetChassisManagerAssetInfo?payLoad=X

Usage scenario:
This API sets the FRU Multi Record Area information for the Chassis Manager with the user specified record data.

Input parameters:

- payLoad (user specified data).

Note: For payload, user can enter it as multiple fields (each representing a Multi Record FRU field), each separated by comma. As part of the specification, the user can only enter maximum 2 fields, where each field can be of maximum 56 bytes (characters) length.

6.6.70 Set PDB Asset Info

MultiRecordResponse SetPDBAssetInfo(string payLoad)

https://localhost:8000/ SetPDBAssetInfo?payLoad=X

Usage scenario:
This API sets the FRU Multi Record Area information for the power distribution board (PDB) with the user specified record data.

Input parameters:

- payLoad (user specified data)

Note: For payload, user can enter it as multiple fields (each representing a Multi Record FRU field), each separated by comma. As part of the specification, the user can only enter maximum two fields, where each field can be of maximum 56 bytes (characters) length.

6.6.71 Set Blade Asset Info

MultiRecordResponse SetBladeAssetInfo(int bladeId)

https://localhost:8000/ SetBladeAssetInfo?bladeId=1&payLoad=xxxxx

Usage scenario:
This API sets the FRU Multi Record Area information with the user specified data for the blade with the specified index.

Input parameters:

- bladeId (blade index 1-48)
- payLoad (user specified data)

Note: For payload, user can enter it as multiple fields (each representing a Multi Record FRU field), each separated by comma. As part of the specification, the user can only enter maximum 2 fields, where each field can be of maximum 56 bytes (characters) length.

6.6.72 Get Blade Post Code

BiosPostCode GetPostCode(int bladeId)

https://localhost:8000/GetPostCode?bladeId=1

Usage scenario:
This API gets the post code for the blade with specified blade id.

Input parameters:

- bladeId (blade index 1-48)

6.6.73 Update PSU Firmware

ChassisResponse UpdatePSUFirmware(int psuId, string fwFilepath, bool primaryImage)

https://localhost:8000/UpdatePSUFirmware?psuId=1&fwFilepath=c:\test\630_002279_0000

_MS1425W_SecWithBootloader_CSE76E_V021700.hex&primaryImage=false

Usage scenario:
This API updates the firmware of the primary or secondary controller of the power supply unit (PSU). When a firmware update is initiated, the PSU will shut off its output power. The Chassis Manager will also stop monitoring the PSU that is being updated.

Since firmware update takes approximately 11 minutes for the primary controller, and 9 minutes for the secondary controller, the update process should only be performed while AC power is supplied to the PSU.

Once firmware update is initiated on a primary or secondary controller, the firmware update must complete successfully before updating the other controller. For example, if firmware update is initiated but failed for any reason on the secondary controller, the firmware update must be executed again on the secondary controller until it completes successfully.

Use the GetPSUFirmwareStatus API to check the status of the firmware update.

Input parameters:

- psuId – the target PSU number to update.Typically 1-6
- fwFilepath – path to firmware image file
- primaryImage – true: firmware image file is for primary controller. False: firmware image file is for secondary controller

6.6.74 Get PSU Firmware Update Status

PsuFirmwareStatus GetPSUFirmwareStatus(int psuId)

https://localhost:8000/GetPSUFirmwareStatus?psuId=1

Usage scenario:
This API gets the firmware revision, overall firmware update status and the progress stage of the firmware update.

7 Command Line Interface

The CLI is intended for service technicians or testers who need quick access to the Chassis Manager services and capabilities without having to use a browser or write a REST client.

7.1 Install the Chassis Manager Service

Table 46 lists the commands to install, start, stop, and uninstall the Chassis Manager service and to launch the command-line interface.

Table 46: Commands to install Chassis Manager service and launch the CLI

Action	Command
Install service	CM-Binary-Directory> C:\Windows\Microsoft.NET\Framework\v4.0.30319\InstallUtil.exe Microsoft.GFS.WCS.ChassisManager.exe
Start service	net start chassismanager
Stop service	net stop chassismanager

Action	Command
Uninstall service	CM-Binary-Directory> C:\Windows\Microsoft.NET\Framework\v4.0.30319\InstallUtil.exe /u Microsoft.GFS.WCS.ChassisManager.exe
Launch CLI	WcsCli-Binary-Directory> wcscli –h <hostname> -p <port> -s<SSL encryption option> [[-u] <username> [-x] <password>] [-b <batch_file_name>] -h <hostname> Host name of computer on which the Chassis Manager service is running -p <port> Port on which Chassis Manager Service is listening (It is usually 8000) -s <SSL encryption option> • Select 0- To disable SSL encryption. • Select 1-To enabled SSL encryption. -u and -x Optional parameters, user credentials (username(-u) and password(-x)) to connect to CM service. If not specified default credentials are used. B<Batch file name> Optional argument, use to execute commands from a batch file. Note that host name, port, and SSL encryption options are mandatory and should be supplied to launch the CLI.
Get CLI Version	WcsCli-Binary-Directory> wcscli –v Returns the current CLI version.

7.2 State and Information Commands

The sections that follow describe the ACS system CLI commands, which provide information about the system state.

7.2.1 GetChassisInfo

Description:
This command gets status information about chassis components including the following:
Blades – for example, GUID and power status
Power supplies – for example, power draw status and serial number
Chassis Manager – version information, IP information, and system uptime
The power supply status is not available during a PSU firmware upgrade process.

Syntax:
`wcscli -getchassisinfo [-s] [-p] [-c] [-t] [-h]`
-s – Show information about blades
-p – Show information about power supplies
-c – Show Chassis Manager information
-h – Help, display the correct syntax

Sample usage:
`wcscli#  wcscli -s -p -c -t`

Sample output:
```
== Compute Nodes ==
#        | Name   | GUID      | State | BMC MAC        | Completion Code
1        | BLADE1         | 71cd4e40-a900-11e1-9856-089e013a37e8  | On      |
DeviceID: 0MAC Address: 08:9E:01:22:FB:42      | Success
2        | BLADE2         | 590fbcc0-a910-11e1-b117-089e013a37f8  | On      |
DeviceID: 0MAC Address: 08:9E:01:22:FB:32      | Success
3        | BLADE3         | b56945e0-a93d-11e1-be83-089e013a3809  | On      |
DeviceID: 0MAC Address: 08:9E:01:22:FB:3E      | Success
4        | BLADE4         | 9f7f0a40-a83d-11e1-a8ad-089e013a3798  | On      |
DeviceID: 0MAC Address:
08:9E:01:29:60:32      | Success
5        | BLADE5         | ae7ee0a0-d50c-11e1-b27d-089e015a2876  | On      |
DeviceID: 0MAC Address: 08:9E:01:5A:2C:16      | Success
6        | BLADE6         | 506d99c0-d4fd-11e1-b020-089e015a2872  | On      |
DeviceID: 0MAC Address: 08:9E:01:5A:2C:1C      | Success
7        | BLADE7         | 07c79a60-d505-11e1-a944-089e015a2874  | On      |
DeviceID: 0MAC Address: 08:9E:01:5A:2C:0A      | Success
8        | BLADE8         | 7b7398a0-d4e8-11e1-aa7a-089e015a286c  | On      |
DeviceID: 0MAC Address: 08:9E:01:5A:2C:18      | Success
9        | BLADE9         | 3d54f900-d4df-11e1-a52d-089e015a286a  | On      |
DeviceID: 0MAC Address: 08:9E:01:5A:2C:1E      | Success
10       | BLADE10        | dcfaf040-d4f3-11e1-8ce3-089e015a2870  | On      |
DeviceID: 0MAC Address: 08:9E:01:5A:2C:40      | Success
…......
== Power Supplies ==
#        | Serial Num     | State | Pout (W)       | Completion Code
1        | 46-49-51-44-31-32-33-37-30-30-31-31-32-33      | On      | 194     |
Success
2        | 46-49-51-44-31-32-33-37-30-30-31-31-32-38      | On      | 228     |
Success
3        | 46-49-51-44-31-32-33-37-30-30-31-30-36-30      | On      | 190     |
Success
4        | 46-49-51-44-31-32-33-37-30-30-31-30-38-33      | On      | 214     |
Success
5        | 46-49-51-44-31-32-33-37-30-30-31-30-33-30      | On      | 188     |
Success
6        | 46-49-51-44-31-32-33-37-30-30-31-30-32-39      | On      | 203     |
Success

== Chassis Controller ==
Firmware Version         : 02.02
```

```
Hardware Version          : 1
Serial Number             : 33333333
Asset Tag                 :
IP Address                : 192.168.100.23
IP Address Source         : 192.168.100.8
System Uptime             : 00:21:32.5127429
```

7.2.2 GetBladeInfo

Description:
This command gets information about the blades, including serial number and version information.

Syntax:
`wcscli -getbladeinfo [ -i <blade_index> | -a ] [-h]`

-i – Blade index (1-24)
-a – Get information for all blades
-h – Help, display the correct syntax

Sample usage:
To get information on blade 1, execute the following command:

```
WcsCli# wcscli -getbladeinfo -i 2
```

Sample output:

```
== Compute Node Info ==
Firmware Version          : 03.02
Hardware Version          : T6M
Serial Number             : QTFCTM2350005
Asset Tag                 :

== MAC Address ==
Device Id                         : 0
MAC Address               : 08:9E:01:22:FB:32
```

7.2.3 GetChassisHealth

Description:
This command gets health status for blades, power supplies, and fans. The power supply status is not available during a PSU firmware upgrade process.

Syntax:

```
wcscli -getchassishealth [-b] [-p] [-f] [-t] [-h]
```

-b – Show blade health
-p – Show PSU health
-f – Show fan health

–h – Help, display the correct syntax

Sample usage:

To get information about blade 1, execute the following command:

```
wcscli# wcscli -getchassishealth -b -p -f
```

Sample output:

```
== Blade Health ==
Blade Id          : 1
Blade State       : ON
Blade Type        : Server

Blade Id          : 2
Blade State       : ON
Blade Type        : Server

Blade Id          : 3
Blade State       : ON
Blade Type        : Server

Blade Id          : 4
Blade State       : OFF
Blade Type        : Server

Blade Id          : 5
Blade State       : ON
Blade Type        : Unknown

Blade Id          : 6
Blade State       : ON
Blade Type        : Unknown
……
……….
== PSU Health ==
Psu Id  : 1
Psu Serial Number        : 46-49-51-44-31-32-33-37-30-30-31-3
Psu State                : ON
PSU Power Out            : 0
Psu Completion code: Success
```

```
Psu Id  : 2
Psu Serial Number       : 46-49-51-44-31-32-33-37-30-30-31-3
Psu State               : ON
PSU Power Out           : 0
Psu Completion code: Success

Psu Id  : 3
Psu Serial Number       : 46-49-51-44-31-32-33-37-30-30-31-3
Psu State               : ON
PSU Power Out           : 0
Psu Completion code: Success

Psu Id  : 4
Psu Serial Number       : 46-49-51-44-31-32-33-37-30-30-31-3
Psu State               : ON
PSU Power Out           : 0
Psu Completion code: Success

Psu Id  : 5
Psu Serial Number       : 46-49-51-44-31-32-33-37-30-30-31-3
Psu State               : ON
PSU Power Out           : 0
Psu Completion code: Success

Psu Id  : 6
Psu Serial Number       : 46-49-51-44-31-32-33-37-30-30-31-3
Psu State               : ON
PSU Power Out           : 0
Psu Completion code: Success


== Fan Health ==
Fan Id  : 1
Fan Speed: 3670
Fan status: ON

Fan Id  : 2
Fan Speed: 3683
Fan status: ON
```

```
Fan Id  : 3
Fan Speed: 3474
Fan status: ON

Fan Id  : 4
Fan Speed: 3468
Fan status: ON

Fan Id  : 5
Fan Speed: 3633
Fan status: ON

Fan Id  : 6
Fan Speed: 3571
Fan status: ON
```

7.2.4 GetBladeHealth

Description:
This command gets health information about the blade, including CPU, memory, disk (JBOD only), PCIe, sensor, and FRU information. The information can be requested separately using specific command options.

Syntax:
```
wcscli -getbladehealth [-i <blade_index>] [-q] [-m] [-d] [-p] [-s] [-t] [-f] [-h]
```

−q – Show blade CPU information
−m – Show blade memory information
−d – Show JBOD disk information
−p – Show blade PCIe information
−s – Show blade sensor information
−t – Show temperature sensor information
−f – Show blade FRU information
−h – Help, display the correct syntax

Sample usage:

To get information on blade 1, execute the following command:

```
wcscli# wcscli -getbladehealth -i 1
```

Sample output:

```
== Blade 2 Health Information ==
Blade ID        : 2
Blade State     : ON
Blade Type      : Server
```

```
== Memory Information ==
Dimm               : 1
Dimm Type          : DDR3
Memory Voltage     : 1.35V
Size               : 16384
Speed              : 1333
Memory Status      : Ok
Memory Completion code: Success

Dimm               : 2
Dimm Type          : NotPresent
Memory Voltage     : NotPresent
Size               : 0
Speed              : 0
Memory Status      : NotPresent
Memory Completion code: Success

Dimm               : 3
Dimm Type          : DDR3
Memory Voltage     : 1.35V
Size               : 16384
Speed              : 1333
Memory Status      : Ok
Memory Completion code: Success

== PCIE Information ==
Dimm               : 1
Dimm Type          : DDR3
Memory Voltage     : 1.35V
Size               : 16384
Speed              : 1333
Memory Status      : Ok
Memory Completion code: Success

Dimm               : 2
Dimm Type          : NotPresent
Memory Voltage     : NotPresent
Size               : 0
Speed              : 0
```

```
Memory Status    : NotPresent
Memory Completion code: Success

Dimm             : 3
Dimm Type        : DDR3
Memory Voltage   : 1.35V
Size             : 16384
Speed            : 1333
Memory Status    : Ok
Memory Completion code: Success

== FRU Information ==
Blade Serial Number      : QTFCTM2370293
Blade Asset Tag :
Blade Product Type       :
Blade Hardware Version   : T6M
```

Note that some of the fields populated are blank as complete FRU data is not available for the on which the sample command was executed.

7.2.5 GetServiceVersion

Description:

This command gets Chassis Manager service assembly version.

Syntax:

`wcscli -getserviceversion [-h]`

-h – Help, display the correct syntax

Sample usage:

To get information on blade 1, execute the following command:

`wcscli# wcscli -getserviceversion`

Sample output:

```
Chassis Manager Service version: 1.2.0.0
```

7.2.6 Update PSU Firmware

Description:

This command updates the firmware of the primary or secondary controller of the power supply unit (PSU). When a firmware update is initiated, the PSU will shut off its output power. The Chassis Manager will also stop monitoring the PSU that is being updated.

Since firmware update takes approximately 11 minutes for the primary controller, and 9 minutes for the secondary controller, the update process should only be performed while AC power is supplied to the PSU.

Once firmware update is initiated on a primary or secondary controller, the firmware update must complete successfully before updating the other controller. For example, if firmware update is initiated but failed for any reason on the secondary controller, the firmware update must be executed again on the secondary controller until it completes successfully.

Use the getpsufwstatus command to check the status of the firmware update.

Syntax:

```
wcscli -updatepsufw [-i <psuId>] [-f <filePath>] [-
p <isPrimaryImage>]  [-h]
        psuId - the target PSU number to update. Typically 1-6
        -f - path to firmware image file
        -p - 1: Firmware image file is for primary controller.
             0: Firmware image file is for secondary controller.
        -h - help; display the correct syntax
```

Sample usage:

To update the secondary controller on PSU 1, execute the following command:

```
wcscli# wcscli -updatepsufw -i 1 -f c:\test\630_002279_0000
_MS1425W_SecWithBootloader_CSE76E_V021700.hex -p 0
```

Sample output:

OK. Use wcscli -getpsufwstatus to check the status of the firmware update.

7.2.7 Get PSU Firmware Update Status

Description:

This command gets the firmware revision, overall firmware update status and the progress stage of the firmware update.

Syntax:

```
wcscli -getpsufwstatus [-i <psuId>] [-h]
        psuId - the target PSU number. Typically 1-6
        -h - help; display the correct syntax
```

Sample usage:

To get the firmware update status on PSU 1, execute the following command:

```
wcscli# wcscli -getpsufwstatus -i 1
```

Sample output:

== PSU Firmware Status ==
Firmware Revision : D1.11.11
Firmware Update Status : NotStarted
Firmware Update Stage : NotStarted
Completion Code : Success

7.3 Blade Management Commands

The sections that follow describe the ACS system CLI blade management commands.

7.3.1 SetPowerOn

Description:
This command turns the AC outlet power ON for the blade.

When AC power gets supplied to the blade and the default blade power state is set to ON, the blade chipset will start to receive power and boot process will be initiated. If the default blade power state is set to OFF when AC power is applied, the blade chipset will not receive power (and the boot process will not be initiated). You can explicitly send a **SetBladeOn** command to power the blade on.

Syntax:
```
wcscli -setpoweron [-i <blade_index> | -a] [-h]
```
-i – Blade index (1-24)
-a – Get information for all blades
-h – Help, display the correct syntax

Sample usage:
To turn the power ON on blade 3, use the following command:
```
wcscli#  wcscli -setpoweron -i 3
```

Sample output:
```
OK
```

7.3.2 SetPowerOff

Description:
This command turns the AC outlet power OFF for the blades.

Syntax:
```
wcscli -setpoweroff [-i <blade_index> | -a] [-h]
```
-i – Blade index (1-24)
-a – Get information for all blades
-h – Help, display the correct syntax

Sample usage:
To turn the power OFF on blade 3, use the following command:

```
wcscli#  wcscli -setpoweroff -i 3
```

Sample output:

```
OK
```

7.3.3 GetPowerState

Description:
This command gets the AC outlet power ON/OFF state of blades (whether or not the blades are receiving AC power).

- When ON, blade is receiving AC power (hard power state).
- When OFF, blade is not receiving AC power.

Syntax:

```
wcscli -getpowerstate [-i <blade_index> | -a] [-h]
```

-i – Blade index (1-24)
-a – Get information for all blades
-h – Help, display the correct syntax

Sample usage:
To get the power state for blade 1, use the following command:

```
wcscli#  wcscli -getpowerstate -i 1
```

Sample output:

```
Blade 5: On
```

7.3.4 SetBladeOn

Description:
This command supplies the power to the blade chipset, initializing the boot process. This command is used to soft power the blade ON.

Syntax:

```
wcscli -setbladeon [ -i <blade_index> | -a] [-h]
```

-i – Blade index (1-24)
-a – All connected blades
-h – Help, display the correct syntax

Sample usage:
To soft power ON blade 1, use the following command:

```
wcscli#  wcscli -setbladeon -i 1
```

Sample output:

```
Blade 1: ON
```

7.3.5 SetBladeOff

Description:
This command removes the power from the blade chipset. This command is used to soft power the blade OFF.

Syntax:
```
wcscli -setbladeoff [[-i <blade_index>] |[-a]] [-h]
```
-i – Blade index (1-48)
-a – All connected blades
-h – Help, display the correct syntax

Sample usage:
To soft power OFF blade 1, use the following command:
```
wcscli#  wcscli -setbladeoff -i 1
```

Sample output:
```
Blade 1: OFF
```

7.3.6 GetBladeState

Description:
This command gets the ON/OFF state of the blade (whether the blade chipset is receiving power).

- When ON, blade is receiving AC power (hard power state) and the chipset is receiving power (soft power state).
- When OFF, blade chipset is not receiving power.

Syntax:
```
wcscli -getbladestate [[-i <blade_index>] | [-a]] [-h]
```
-i – Blade index (1-24)
-a – All connected blades
-h – Help, display the correct syntax

Sample usage:
To get the ON/OFF state of blade 1, use the following command:
```
wcscli#  wcscli -getbladestate -i 1
```

Sample output:
```
Blade State 1: ON
```

7.3.7 SetBladeDefaultPowerState

Description:
This command sets the default power state of the blade ON/OFF.

The default blade power state denotes the behavior of the blade after receiving AC power, either when a blade is initially inserted in to its slot or power returns after a utility failure. If the blade default power state is set to OFF, the blade won't be powered ON after receiving AC input power. An explicit **SetBladeOn** command needs to be sent to power ON the blade. If the blade default power state is set to ON, the blade will be powered ON after receiving AC input power.

Note that the blade default power state does not affect the active power state of the blade, only their behaviors after a hard power recycle.

Syntax:

```
wcscli -setbladedefaultpowerstate [[-i <blade_index>] | [-a] ] -s
<state>[-h]
```

-i – Blade index (1-24)
-a – All connected blades
-s – State, can be 0 (stay OFF) or 1 (power ON)
-h – Help, display the correct syntax

Sample usage:
To set the default power state of of blade 1 to ON, use the following command:

```
wcscli#  wcscli -setbladedefaultpowerstate -i 1 -s 1
```

Sample output:

```
Blade 1 Default Power State: ON
```

7.3.8 GetBladeDefaultPowerState

Description:
This command gets the default power state of the blade ON/OFF.

Syntax:

```
wcscli -getbladedefaultpowerstate [[-i <blade_index>] | [-a]][-h]
```

-i – Blade index, the number of blades (1-24)
-a – Gets information for all blades
-h – Help, display the correct syntax

Sample usage:
To set the default power state of of blade 1 to ON, use the following command:

```
wcscli#  wcscli -getbladedefaultpowerstate -i 1
```

Sample output:

```
Blade 1 Default Power State: ON
```

7.3.9 SetBladeActivePowerCycle

Description:
This command power cycles or soft resets the blade(s).

Power cycle resets the blade (causing a software reboot sequence). Blade AC power signal remains ON throughout this process. Any serial session active on that blade will continue to be active during this process.

Syntax:

```
wcscli -setbladeactivepowercycle [[-i <blade_index>]| [-a]] | [-t
<off_time>][-h]
```

-i – Blade index, the number of blades (1-24)
-a – Gets information for all blades
-t – Indicates the power off time, i.e., the time interval in seconds for how long the blade stays OFF before the power is again turned on. If not specified, the default interval is 0 seconds.
-h – Help, display the correct syntax

Sample usage:
To power cycle blade 3, use the following command:

```
wcscli#  wcscli -setbladeactivepowercycle -i 3
```

Sample output:

```
OK
```

7.3.10 SetBladeAttentionLEDOn

Description:
This command turns the blade attention LED On. The purpose of this attention LED is to indicate that this blade needs attention. The command can also be used to identify the blade.

The blade attention LED is used to help service technicians find the blade during repair. Users can also flag a service requirement by turning ON this LED. Operators/users must ensure that the blade attention LED is turned OFF after service is complete.

Syntax:

```
wcscli -setbladeattentionledon [[-i <blade_index>]| [-a]][-h]
```

-i – Blade index, the number of blades (1-24)
-a – Gets information for all blades
-h – Help, display the correct syntax

Sample usage:

To turn the blade attention LED for blade 1 ON, use the following command:

```
wcscli#  wcscli -setbladeattentionledon -i 1
```

Sample output:

```
OK
```

7.3.11 SetBladeAttentionLEDOff

Description:
This command turns the blade attention LED Off. The purpose of this attention LED is to indicate that this blade needs attention.

The blade attention LED is used to help service technicians find the blade during repair. Users can also flag a service requirement by turning ON this LED. Operators/users must ensure that the blade attention

LED is turned OFF after service is complete.

Syntax:
```
wcscli -setbladeattentionledoff [[-i <blade_index>]|[-a]][-h]
```
−i – Blade index, the number of blades (1-24)
−a – Gets information for all blades
−h – Help, display the correct syntax

Sample usage:
To turn the blade attention LED for blade 1 OFF, use the following command:
```
wcscli#  wcscli -setbladeattentionledoff -i 1
```

Sample output:
```
OK
```

7.3.12 ReadBladeLog

Description:
This command reads the log from a blade. The blade log (system event log) contains information about events/warnings/alerts pertaining to that blade like thermal throttling of blades due to overheating, etc.

Syntax:
```
wcscli -readbladelog [-i <blade_index>] [-n <entries_count>] [-h]
```
−i – Blade index, the number of blades (1-24)
−n – How many of the most recent entries to report. This is an optional parameter; if not specified the command will return all existing entries.
−h – Help, display the correct syntax

Sample usage:

To read the blade log for blade 1, use the following command:
```
wcscli#  wcscli -readbladelog -i 1
```

Sample output:
```
#       | EventTime           |    EventDescription

| 2012-08-23T14:35:21| Sensor_SpecificDrive_Slot2433328Assertion111
| 2012-08-23T16:11:08| DiscreteTemperature187257Desertion3
| 2012-08-23T18:35:21| Sensor_SpecificDrive_Slot2463328Assertion111
| 2012-08-23T19:35:21| Sensor_SpecificDrive_Slot2433328Assertion111
| 2012-08-23T20:35:21| Sensor_SpecificDrive_Slot2433328Assertion111
```

7.3.13 ClearBladeLog

Description:
This command clears the log from a blade.

Syntax:
```
wcscli -clearbladelog [-i <blade_index>] [-h]
```

–i – Blade index, the number of blades (1-24)
–h – Help, display the correct syntax

Sample usage:
To clear the blade log for blade 1, use the following command:

```
wcscli#  wcscli –clearbladelog –i 1
```

Sample output:

```
OK
```

7.3.14 SetBladePowerLimit

Description:
This command sets the power limit for a blade.

The power limit can be set for a variety of reasons including energy savings and under provisioning (consolidate more servers under a power hierarchy).

The minimum and maximum power limits are specified in the Chassis Manager app.config file.

Syntax:

```
wcscli -setbladepowerlimit [[-i <blade_index>] | [-a ]] -l <power_limit>
[-h]
```

–i – Blade index, the number of blades (1-24)
–a – Perform action for all blades
–l – Blade power limit, in W
–h – Help, display the correct syntax

Sample usage:

To set the power limit for blade 1 to 750 W, use the following command:

```
wcscli#  wcscli –setbladepowerlimit –i 1 –l 750
```

Sample output:

```
Blade 1 power limit: 750 Watts
```

7.3.15 SetBladePowerLimitOn

Description:
This command activates the power limit for a blade, and enables power throttling.

Syntax:

```
wcscli -setbladepowerlimiton [[-i <blade_index>]|[-a]] [-h]
```

–i – Blade index, the number of blades (1-24)
–a – Perform action for all blades
–h – Help, display the correct syntax

Sample usage:
To activates the power limit for blade 1, use the following command:

```
wcscli#  wcscli -setbladepowerlimiton -i 1
```

Sample output:

```
Blade 1: ON
```

7.3.16 SetBladePowerLimitOff

Description:
This command deactivates the power limit for a blade, and disables power throttling.

Syntax:

```
wcscli -setbladepowerlimitoff [[-i <blade_index>]|[-a]] [-h]
```

–i – Blade index, the number of blades (1-24)
–a – Perform action for all blades
–h – Help, display the correct syntax

Sample usage:
To deactivates the power limit for blade 1, use the following command:

```
wcscli#  wcscli -setbladepowerlimitoff -i 1
```

Sample output:

```
Blade 1: OFF
```

7.3.17 GetBladePowerLimit

Description:
This command gets the power limit for a blade.

Syntax:

```
wcscli -getbladepowerlimit [[-i <blade_index>]|[-a]] [-h]
```

–i – Blade index, the number of blades (1-24)
–a – Perform action for all blades
–h – Help, display the correct syntax

Sample usage:
To get the power limit for blade 1, use the following command:

```
wcscli#  wcscli -getbladepowerlimit -i 1
```

Sample output:

```
Blade 1 power limit 750 Watts
```

7.3.18 GetBladePowerReading

Description:
This command gets the power reading for a blade. The command can be used for monitoring and or other power control mechanism (refer to the **SetBladePowerLimit** command).

Syntax:

```
wcscli -getbladepowerreading [[-i <blade_index>]|[-a]] [-h]
```

–i – Blade index, the number of blades (1-24)
–a – Perform action for all blades
–h – Help, display the correct syntax

Sample usage:
To get the power reading for blade 1, use the following command:

```
wcscli#  wcscli -getbladepowerreading -i 1
```

Sample output:

```
Blade 1 power reading: 67 Watts
```

7.3.19 GetNextBoot

Description:
This command gets the pending boot order to be applied the next time the blade boots.

Once the boot order is applied, GetNextBoot will return the default value of NoOverRide.

Syntax:

```
wcscli -getnextboot [-i <blade_index>][-h]
```

–i – Blade index, the number of blades (1-24)
–h – Help, display the correct syntax

Sample usage:
To get the next boot device type for blade 1, use the following command:

```
wcscli#  wcscli -getnextboot -i 1
```

Sample output:

```
OK. Next boot is ForcePxe.
```

7.3.20 SetNextBoot

Description:
This command sets the device boot type of the start boot device during the subsequent reboot for a blade.

Syntax:

```
wcscli -setbootnext [-i <blade_index>] [-t <boot_type>] [-m <boot_mode>][-p <is_persistent>] [-n <boot_instance>] [-h]
```

–i – Blade index (1-24)
boot_type – One of the following:
1. NoOverRide
2. ForcePxe
3. ForceDefaultHdd,
4. ForceIntoBiosSetup
5. ForceFloppyOrRemovable

boot_mode - boolean (0 or 1) indicating whether the next boot is wanted to be in UEFI mode (or legacy mode)
is_persistent – Is this a persistent setting (set value 1) or a one-time setting (set value 0)
boot_instance – instance number of the boot device (for example, 0 or 1 for NIC if there are two NICs)
-h – Help, display the correct syntax

Sample usage:
To set the boot device for blade 1, use the following command:

```
wcscli#  wcscli -setnextboot -i 2 -p 1 -m 1 -n 1
```

Sample output:

```
OK. Next boot is ForceDefaultHdd
```

7.3.21 GetBladeAssetInfo

Description:

This command gets the FRU information of the specified blade.

Syntax:

```
wcscli -getbladeassetinfo [-i <blade_index>][-h]
```

-i – Blade index (1-24)
-h – Help, display the correct syntax

Sample usage:
To get the asset (FRU) information for blade 1, use the following command:

```
wcscli#  wcscli -getbladeassetinfo -i 1
```

Sample output:

```
--------------------------------------
Blade Chassis Info Area
--------------------------------------
Chassis Part Number = X898637-001
Chassis Serial Number = QTFCLJ4150005
--------------------------------------

Blade Board Info Area
--------------------------------------
Board Manufacturer Name = Microsoft
Board Manufacturer Date = 4/8/2014 8:18:00 AM
Board Product Name = C1030Q
Board Serial Number = ALJ41300232
Board Part Number = X898568-001
--------------------------------------

Blade Product Info Area
--------------------------------------
Product Manufacturer Name = Microsoft
Product Product Name = C1030Q
```

```
Product Part/Model Number = X892646-001
Product Version = 2.0
Product Serial Number = QTFCLJ4150005
PD Product Asset Tag =
--------------------------------------

Multi Record Info Area
--------------------------------------
Manufacturer =
Custom Fields
```

7.3.22 SetBladeAssetInfo

Description:

This command is used to edit the Multi Record Area FRU information of the specified blade.

Syntax:

`wcscli -setbladeassetinfo [-i <blade_index>] [-p <payload>][-h]`

-i – Blade index (1-24)
-p – data to be written to Chassis Manager FRU Multi Record Area
-h – Help, display the correct syntax

Note:

For payload, user can enter it as multiple fields (each representing a Multi Record FRU field), each separated by comma. As part of the specification, the user can only enter maximum 2 fields, where each field can be of maximum 56 bytes (characters) length.

Sample usage:
To set the asset (FRU) information for blade 1, use the following command:

`wcscli#  wcscli -setbladeassetinfo -i 1 -p TEST`

Sample output:

`Command Success: Blade 1 Blade's FRU has been written successfully`

7.3.23 GetPDBAssetInfo

Description:

This command gets the FRU information of the PDB.

Syntax:

`wcscli -getpdbassetinfo [-h]`

-h – Help, display the correct syntax

Sample usage:
To get the asset (FRU) information for the PDB, use the following command:

```
wcscli#  wcscli -getpdbassetinfo
```

Sample output:

```
--------------------------------------
Power Distribution Board (PDB) Chassis Info Area
--------------------------------------
Chassis Part Number =
Chassis Serial Number =
--------------------------------------

Power Distribution Board (PDB) Board Info Area
--------------------------------------
Board Manufacturer Name =
Board Manufacturer Date = 1/1/1996 12:00:00 AM
Board Product Name =
Board Serial Number =
Board Part Number =
--------------------------------------

Power Distribution Board (PDB) Product Info Area
--------------------------------------
Product Manufacturer Name =
Product Product Name =
Product Part/Model Number =
Product Version =
Product Serial Number =
PD Product Asset Tag =
--------------------------------------

Power Distribution Board (PDB) Multi Record Info Area
--------------------------------------
Manufacturer =
Custom Fields
```

7.3.24 SetPDBAssetInfo

Description:
This command is used to edit the Multi Record Area FRU information of the specified blade.

Syntax:

```
wcscli -setpdbassetinfo [-p <payload>][-h]
```

`-p` – data to be written to PDB FRU Multi Record Area

-h – Help, display the correct syntax

Sample usage:
To set the asset (FRU) information for PDB, use the following command:

```
wcscli#  wcscli -setpdbassetinfo -p TEST
```

Note:
For payload, user can enter it as multiple fields (each representing a Multi Record FRU field), each

separated by comma. As part of the specification, the user can only enter maximum 2 fields, where each field can be of maximum 56 bytes (characters) length.

Sample output:

```
PDB's FRU has been written successfully
```

7.3.25 GetBladeBIOSPostCode

Description:
This command gets the post code for the blade with specified blade id.

Syntax:

```
wcscli -getbladebiospostcode [-i <blade_index> [-h]
blade_index - the target blade number. Typically 1-24
-h - help; display the correct syntax
```

Sample usage:

```
Wcscli # wcscli -getbladebiospostcode -i 1
```

7.4 Chassis Manager Management Commands

The sections that follow describe the ACS system CLI Chassis Manager management commands.

7.4.1 SetChassisAttentionLEDOn

Description:
This command turns the Chassis attention LED On. The purpose of this attention LED is to indicate that this Chassis Manager needs attention.

The chassis attention LED is used to direct service technicians to the correct chassis during repair. Users can also flag a service requirement by turning ON this LED. When possible, repairs will also be self-directed by the chassis management system. Operators/users must ensure that the Chassis Attention LED is turned OFF after service is complete.

Syntax:

```
wcscli -setchassisattentionledon [-h]
```

-h – Help, display the correct syntax

Sample usage:
To turn the chassis attention LED ON, use the following command:

```
wcscli#  wcscli -setchassisattentionledon
```

Sample output:

```
OK
```

7.4.2 SetChassisAttentionLEDOff

Description:
This command turns the Chassis attention LED Off. The purpose of this attention LED is to indicate that this Chassis Manager needs attention.

The chassis attention LED is used to direct service technicians to the correct chassis during repair. Users can also flag a service requirement by turning ON this LED. When possible, repairs will also be self-directed by the chassis management system. Operators/users must ensure that the Chassis Attention LED is turned OFF after service is complete.

Syntax:
```
wcscli -setchassisattentionledoff [-h]
```
-h – Help, display the correct syntax

Sample usage:
To turn the chassis attention LED OFF, use the following command:
```
wcscli#  wcscli -setchassisattentionledoff
```

Sample output:
```
OK
```

7.4.3 GetChassisAttentionLEDStatus

Description:
This command gets the status of the chassis attention LED (whether ON or OFF).

The chassis attention LED is used to direct service technicians to the correct chassis during repair. Users can also flag a service requirement by turning ON this LED. When possible, repairs will also be self-directed by the chassis management system. Operators/users must ensure that the Chassis Attention LED is turned OFF after service is complete.

Syntax:
```
wcscli -getchassisattentionledstatus [-h]
```
-h – Help, display the correct syntax

Sample usage:
To get the status of the chassis attention LED, use the following command:
```
wcscli#  wcscli -getchassisattentionledstatus
```

Sample output:
```
Chassis LED: OFF
```

7.4.4 ReadChassisLog

Description:
This command reads the chassis log.

The chassis log contains information about various alerts/warning messages associated with devices connected to the Chassis Manager (for example, blade overheating or fan/PSU failure).

The chassis log also contains user audit information, such as timestamp/activity performed by that user.

Syntax:

```
wcscli -readchassislog [-s <startDate>] [-e <endDate>] [-h]
-s - Optional start date for the log entries (format: YYYY:MM:DD)
-e - Optional end date for the log entries (format: YYYY:MM:DD)

-h - help; display the correct syntax
```

Sample usage:

To get the chassis user log, use the following command:

```
wcscli#  wcscli -readchassislog
```

Sample output:

```
== Chassis Controller Log ==

Timestamp        | Entry

10/3/2012 3:22:56 PM    | WCS\v-ericku,Invoked
GetChassisInfo(True,True,True)

10/3/2012 3:22:57 PM    | WCS\v-ericku,Invoked GetBladeState(bladeid: 1)

10/3/2012 3:22:57 PM    | WCS\v-ericku,Invoked GetBladeState(bladeid: 2)

10/3/2012 3:22:57 PM    | WCS\v-ericku,Invoked GetBladeState(bladeid: 3)

10/3/2012 3:22:58 PM    | WCS\v-ericku,Invoked
GetChassisNetworkProperties()

10/3/2012 3:23:45 PM    | WCS\v-ericku,Invoked ReadBladelog(bladeId: 1)
```

7.4.5 ClearChassisLog

Description:

This command clears the chassis log.

To comply with size restrictions on Chassis Manager storage space, users are expected to periodically clear the consumed log entries.

Syntax:

```
wcscli -clearchassislog [-h]
```

-h – Help, display the correct syntax

Sample usage:

To clear the chassis user log, use the following command:

```
wcscli#  wcscli -clearchassislog
```

Sample output:
```
OK
```

7.4.6 GetACSocketPowerState

Description:
This command gets the status of chassis AC sockets (TOR switches).

The AC socket power state refers to active power state of the COM ports (and therefore to the power state of the device connected to the port) on the Chassis Manager. For example, the TOR switch is connected to COM1 (port number 1). The power ON/OFF commands for the AC sockets makes it possible to remotely power-reset the device and also for enabling/disabling purpose.

Syntax:
```
wcscli -getacsocketpowerstate -p <port_no> [-h]
```
-p – Port number of interest
-h – Help, display the correct syntax

Sample usage:
To get the status of the chassis AC sockets for port 2, use the following command:
```
wcscli#  wcscli -getacsocketpowerstate -p 2
```

Sample output:
```
ON
```

7.4.7 SetACSocketPowerStateOn

Description:
This command turns the chassis AC sockets (TOR switches) ON.

The AC socket power state refers to active power state of the COM ports (and therefore to the power state of the device connected to the port) on the Chassis Manager. For example, the TOR switch is connected to COM1 (port number 1). The power ON/OFF commands for the AC sockets makes it possible to remotely power-reset the device and also for enabling/disabling purpose.

Syntax:
```
wcscli -setacsocketpowerstateon -p <port_no> [-h]
```
-p – Port number of interest
-h – Help, display the correct syntax

Sample usage:
To turn ON the chassis AC sockets for port 12, use the following command:
```
wcscli#  wcscli -setacsocketpowerstateon -p 12
```

Sample output:
```
OK
```

7.4.8 SetACSocketPowerStateOff

Description:
This command turns the chassis AC sockets (TOR switches) OFF.

The AC socket power state refers to active power state of the COM ports (and therefore to the power state of the device connected to the port) on the Chassis Manager. For example, the TOR switch is connected to COM1 (port number 1). The power ON/OFF commands for the AC sockets makes it possible to remotely power-reset the device and also for enabling/disabling purpose.

Syntax:
```
wcscli -setacsocketpowerstateoff -p <port_no> [-h]
```
-p – Port number of interest
-h – Help, display the correct syntax

Sample usage:
To turn OFF the chassis AC sockets for port 12, use the following command:
```
wcscli#  wcscli -setacsocketpowerstateoff -p 12
```

Sample output:
```
OK
```

7.4.9 AddChassisControllerUser

Description:
This command adds a new chassis controller user with specified password and WCS security role for accessing the Chassis Manager command-line interface. The role should be Admin, Operator, or User. Each role has access to a specific set of APIs.

Syntax:
```
wcscli -adduser -u <username> -p <password> -a|-o|-r [-h]
```
-u username – Username for the new user
-p password – Password for the new user
Select one of the WCS security roles for the user (mandatory):
-a – Admin privilege
-o – Operator privilege
-r– User privilege
-h – Help, display the correct syntax

Sample usage:
To add a user with admin privileges, use the following command:
```
wcscli#  wcscli -adduser -u myname -p pass!123 -a
```

Sample output:
```
OK
```

7.4.10 ChangeChassisControllerUserPassword

Description:
This command changes the password for an existing user in the chassis controller.

Syntax:
```
wcscli -changeuserpwd -u <username> -p <newpassword> [-h]
```
-u username – Username for which the password will be changed
-p newpassword – new password for the user
-h – Help, display the correct syntax

Sample usage:
To add a user with admin privileges, use the following command:
```
wcscli#  wcscli -changeuserpwd -u myname  -p pa##!143
```

Sample output:
```
OK
```

7.4.11 ChangeChassisControllerUserRole

Description:
This command changes the WCS security role for an existing user in the chassis controller.

Note that the user will be removed from other WCS security roles and added to the new role specified. Users can belong to only one security role.

Syntax:
```
wcscli -changeuserrole -u <username> -a|-o|-r [-h]
```
-u username – Username for the new user
Select one of the WCS security roles for the user (mandatory):
-a – Admin privilege
-o – Operator privilege
-r – User privilege
-h – Help, display the correct syntax

Sample usage:
To change the user role to operator, use the following command:
```
wcscli#  wcscli -changeuserrole -u myname  -o
```

Sample output:
```
OK
```

7.4.12 RemoveChassisControllerUser

Description:
This command removes an existing user from the chassis controller.

Syntax:

```
wcscli -removeuser -u <username>[-h]
```

-u username – Username for the new user
-h – Help, display the correct syntax

Sample usage:
To remove a user, use the following command:

```
wcscli#  wcscli -removeuser -u myname
```

Sample output:

```
OK
```

7.4.13 GetChassisManagerAssetInfo

Description:

This command gets the FRU information of the Chassis Manager.

Syntax:

```
wcscli -getbladeassetinfo[-h]
```

-h – Help, display the correct syntax

Sample usage:
To get the asset (FRU) information for the Chassis Manager, use the following command:

```
wcscli#  wcscli -getchassismanagerassetinfo
```

Sample output:

```
 --------------------------------------

Chassis Manager Chassis Info Area

--------------------------------------

Chassis Part Number = X873021-001

Chassis Serial Number = 0000000000000000

--------------------------------------

Chassis Manager Board Info Area

--------------------------------------

Board Manufacturer Name = Microsoft

Board Manufacturer Date = 3/3/2014 12:52:00 AM

Board Product Name = E1000

Board Serial Number = NH840800124

Board Part Number = X873064-001
```

```
--------------------------------------

Chassis Manager Product Info Area

--------------------------------------

Product Manufacturer Name = Microsoft

Product Product Name = E1000

Product Part/Model Number = X873023-001

Product Version = A

Product Serial Number = NH840800124

PD Product Asset Tag = 5892018

--------------------------------------

Chassis Manager Multi Record Info Area

--------------------------------------

Manufacturer =

Custom Fields
```

7.4.14 SetChassisManagerAssetInfo

Description:
This command is used to edit the Multi Record Area FRU information of the Chassis Manager.

Syntax:
`wcscli -setbladeassetinfo [-p <payload>][-h]`

`-p` – data to be written to Chassis Manager FRU Multi Record Area

-h – Help, display the correct syntax

Note:
For payload, user can enter it as multiple fields (each representing a Multi Record FRU field), each separated by comma. As part of the specification, the user can only enter maximum 2 fields, where each field can be of maximum 56 bytes (characters) length.

Sample usage:
To set the asset (FRU) information for the Chassis Manager, use the following command:

`wcscli#  wcscli -setchassismanagerassetinfo -p TEST`

Sample output:
Chassis Manager's FRU has been written successfully

7.5 Blade and Serial Console Session Commands

The sections that follow describe the ACS system CLI blade and serial console session commands.

7.5.1 StartBladeSerialSession

Description:
This command is used to start serial session to a blade. The command will open a Serial-Client-terminal for the serial session.

Users might want to open a serial session to a blade for debugging purposes, to view blade boot messages, or for executing BIOS commands. A VT100 console will be provided that will continuously poll the blade for any serial session data. Any user command entered using the VT100 console will be sent to the blade.

Note that this session might close unexpectedly if there is a simultaneous IPMI command issued to any of the chassis blades or because to inactivity of more than five minutes.

In order to avoid the user screen from being overlapped with the incoming content from the serial session, it's recommended to clear the screen first before establishing a serial session with a blade.

Syntax:
```
wcscli -startbladeserialsession [-i <blade_index>] [-s
<session_timeout_in_secs>] [-h]
```
blade_index - the number of the blade. Typically 1-24
-s Session timeout in seconds
-h - help; display the correct syntax

Sample usage:
To start a serial session to the blade, use the following command:
```
wcscli# wcscli -startbladeserialsession -i blade_index
```

Sample Output:
This opens a VT100 session for that blade.

7.5.2 StopBladeSerialSession

Description:
This command is used to force the termination of any active serial session to any blade and for any 'KillSerialConsoleSession' App.config parameter value.

Chassis Manager service exposes a config. parameter called 'KillSerialConsoleSession' in its App.config file. The default value of this parameter is 1, which means that an incoming IPMI command will terminate an existing blade serial console session, allowing the incoming command to proceed (default behavior). A value of 0 implies that an incoming IPMI command will be ignored (fail) and an already existing blade serial session will not be interrupted. This is a configurable parameter which can be adjusted based on specific use case scenario. Note that in order to change this configuration value, the ChassisManager service must first be stopped. The configuration file is locked while the service is running.

Syntax:
```
wcscli -stopbladeserialsession [-h]
```
-h - help; display the correct `syntax`

Sample usage:
To stop a serial session to the blade, use the following command:
```
wcscli# wcscli –stopbladeserialsession
```

Sample Output:
The serial console session to the given blade will be terminated and control returns to WCSCLI prompt.

7.5.3 StartPortSerialSession

Description:
This command is used to open a serial port console terminal to serial devices that are connected to the Chassis Manager (for example, Top of Rack (TOR) network switches). Session timeout and device timeout can be specified as input, if not specified default timeout values (two minutes for session timeout and 100ms for device timeout) from service config are used for the session. Only ports 1 (COM1) and 2 (COM2) connections from the Chassis Manager are supported.

In order to avoid the user screen from being overlapped with the incoming content from the serial session, it is recommended to clear the screen first before establishing a serial session.

Syntax:
```
wcscli -startportserialsession [-i <Port_index>][-s
<session_timeout_in_secs>] [-d <device_timeout_in_millisecs>]
[-r <baud_rate> (75,110,300,1200,2400,4800,9600,19200,38400,57600,115200)][-h]
```
-i Port number. The number of the COM port the device is connected to. Enter 1 for COM1, 2 for COM2.
-s Session timeout in secs,
-d Device timeout in millisecs
-r Baud rate
-h - Help; display the correct syntax

Sample usage:
To to start serial port consolel, use the following command:
```
wcscli# wcscli –startportserialsession -i port_index –s 120 –d 100
```

Sample output:
This opens VT100 session for the serial port console.

7.5.4 StopPortSerialSession

Description:
Stop all existing sessions on given port.

Syntax:
```
wcscli -stopPortSerialSession [-i <Port_index> [-h]
```
-i Port number
-h - Help; display the correct syntax

Sample usage:
To to start serial port consolel, use the following command:

```
wcscli# wcscli –stopPortSerialSession -i port_number
```

Sample output:

```
All existing sessions ended.
```

7.5.5 EstablishCmConnection

Description:
Create a connection to the Chassis Manager service.

Make sure 'local echo' is enabled when connection to CM is established using this command. Otherwise, the commands typed by the user won't be visible on the terminal.

Syntax:

```
wcscli establishCmConnection -m <host_name> -p <port> -s <SSL_option> [-u]
<username> [-x] <password> [-b] <batchfileName> [-h]

-m host_name - Specify Host name for Chassis manager (for serial
connection, localhost is assumed)

-p port - Specify a valid Port to connect to for Chassis Manager (default
is 8000)
-s Select Chassis Manager (CM)'s SSL Encryption mode (enabled/disabled?)
Enter 0 if CM is not configured to use SSL encrytion (SSL disabled in CM)
Enter 1 if CM requires SSl Encryption (SSL enabled in CM)
-u & -x specify user credentials -- username and password -- to connect to
CM service (Optional.. will use default credentials)
-b Optional batch file option (not supported in serial mode).
-v Get CLI version information
-h help
```

Sample usage:
To establish a connection to the Chassis Manager, use the following command:

```
wcscli# wcscli –establishCmConnection -p 8000 -s 0 -u username -x password
```

Sample output:

```
Connection to CM succeeded..
```

7.5.6 TerminateCmConnection

Description:
Terminate a connection to the Chassis Manager service.

Syntax:
```
wcscli terminateCmConnection [-h]
```
```
-h help
```

Sample usage:
To terminate a connection to the Chassis Manager, use the following command:
```
wcscli# wcscli –terminateCmConnection
```

Sample output:
```
Connection to CM terminated successfully.
```

7.6 CLI Over Serial (WCSCLI+)

Note that these commands are available in WCSCLI Serial mode only. Otherwise, all these commands will fail with the following console message:

```
Command only supported in serial wcscli client mode.
```

7.6.1 StartChassisManager

Description:
This command is used to start serial Windows Chassis Manager service.

Syntax:

```
wcscli -startchassismanager [-h]
```

-h - help; display the correct `syntax`

Sample usage:
To start the Windows Chassis Manager service, use the following command:
```
wcscli# wcscli –startchassismanager
```

Sample Output:
Chassis Manager successfully started.

7.6.2 StopChassisManager

Description:
This command is used to stop an existing Windows Chassis Manager service.

Syntax:
```
wcscli -stopchassismanager [-h]
```
-h - Help; display the correct syntax

Sample usage:
To stop Windows Chassis Manager service, use the following command:

```
wcscli# wcscli –stopchassismanager
```

Sample output:
Chassis manager service successfully stopped.

7.6.3 GetChassisManagerStatus

Description:
Get the status of Windows Chassis Manager service.

Syntax:

```
wcscli -getchassismanagerstatus [-h]
```

-h - Help; display the correct syntax

Sample usage:
To get the status of Chassis Manager service, use the following command:

```
wcscli# wcscli –getchassismanagerstatus
```

Sample output:

chassismanager service status: Running

OK

7.6.4 EnableChassisManagerSsl

Description:
Enable SSL for the Chassis Manager service.

Syntax:

```
wcscli -enablechassismanagerssl [-h]
```

-h - Help; display the correct syntax

Sample usage:
To enable SSL for the Chassis Manager service, use the following command:

```
wcscli# wcscli –enablechassismanagerssl
```

Sample output:

Successfully enabled SSL in the chassismanager service.

You will need to establish connection to the CM again via establishCmConnection command to run any commands.

wcscli -establishCmConnection -m <host_name> -p <port> -s <SSL_option> [-u] <username> [-x] <password> [-b] <batchfileName>

-m host_name - Specify Host name for Chassis manager (for serial connection, localhost is assumed)

-p port - Specify a valid Port to connect to for Chassis Manager (default is 8000)

-s Select Chassis Manager (CM)'s SSL Encryption mode (enabled/disabled?)

Enter 0 if CM is not configured to use SSL encrytion (SSL disabled in CM)

Enter 1 if CM requires SSl Encryption (SSL enabled in CM)

-u & -x specify user credentials -- username and password -- to connect to CM service (Optional.. will use default credentials)

-b Optional batch file option (not supported in serial mode).

-v Get CLI version information

-h help

7.6.5 DisableChassisManagerSsl

Description:
Disable SSL for the Chassis Manager service.

Syntax:

```
wcscli -disablechassismanagerssl [-h]
```

-h - Help; display the correct syntax

Sample usage:
To disable SSL for the Chassis Manager service, use the following command:

```
wcscli# wcscli -disablechassismanagerssl
```

Sample output:

Successfully disabled SSL in the chassismanager service.

You will need to establish connection to the CM again via establishCmConnection command to run any commands.

wcscli -establishCmConnection -m <host_name> -p <port> -s <SSL_option> [-u] <username> [-x] <password> [-b] <batchfileName>

-m host_name - Specify Host name for Chassis manager (for serial connection, localhost is assumed)

-p port - Specify a valid Port to connect to for Chassis Manager (default is 8000)

```
-s Select Chassis Manager (CM)'s SSL Encryption mode (enabled/disabled?)
Enter 0 if CM is not configured to use SSL encrytion (SSL disabled in CM)
Enter 1 if CM requires SSl Encryption (SSL enabled in CM)
```

-u & -x specify user credentials -- username and password -- to connect to CM service (Optional.. will use default credentials)

-b Optional batch file option (not supported in serial mode).

-v Get CLI version information

-h help

7.6.6 GetNetworkProperties (getnic)

Description:
This command gets the chassis controller network properties.

Syntax:

```
wcscli -getnic[-h]
```

Note that when reading the NIC settings, in DHCP mode, all old NIC settings for the subnet mask, gateway IP and DNS servers will be returned by Windows through WMI.

-h – Help, display the correct syntax

Sample usage:
To get the network configuration details for the chassis, use the following command:

```
wcscli#  wcscli -getnic
```

Sample output:

```
N/W Interface 0:
        Index                   : 10
        Description             : Intel(R) 82574L Gigabit Network
Connection
        ServiceName             : e1iexpress
        MAC Address             : 08:9E:01:6C:24:1B
        IP Enabled              : True
        DHCP Enabled            : False
        DHCP Server             :
        IP Address              : {192.168.150.56,
fe80::8533:7d05:79b4:8895}
        Subnet Mask             : {255.255.0.0, 64}
        Gateway Address         : {124.55.22.33}
        DNS Servers             :
        DNS Hostname            : CMPVT10
        DNS Domain              :
        WINS Primary            :
        WINS Secondary          :
        Completion Code         : Success
```

7.6.7 SetNetworkProperties (setnic)

Description:
This command sets the chassis controller network properties (only available over serial wcscli client).

Syntax:

```
wcscli -setnic [-a] <IP addr source DHCP/STATIC -Required!> [-i] <IP
address (Required for Static IP)> [-m] <subnetmask (Required for Static
IP)> [-g] <gateway> [-p] <primary DNS> [-d] <secondary DNS> [-t] <network
interface number> [-h]
```

-a - IP addr source DHCP/Static

-i - IP address of the chassis controller (Required for Static IP. Not used for DHCP.)

-m - subnet mask of the chassis controller (Required for Static IP. Not used for DHCP.)

-g - gateway of the chassis controller (Optional for Static IP. Not used for DHCP.). Note that the gateway IP can be cleared by switching to DHCP and then back to static IP.

-p - primary DNS server address for the chassis controller (Optional)

-d - secondary DNS server address for the chassis controller (Optional. Only valid if primary DNS is also specified.)

-t - network interface controller number to configure (0-index. Optional). Default to 0 if not specified
-h – help; display the correct syntax

Sample usage:
To set the network configuration for the chassis, use the following command:

```
wcscli#  wcscli -setnic -m 255.255.0.0 -i 10.160.148.220 -p 10.160.148.220
-d 127.0.0.1 -a static -t 0
```

Sample output:
The command will execute successfully, and there will be intermittent connection loss to Chassis Manager because of the network config. update.

7.6.8 Clear

Description:
This command allows user to clear previously entered commands from history. Also clears the PuTTY client screen.

Syntax:

```
wcscli -clear [-h]
```

-h - Help; display the correct syntax

Sample usage:
To clear command history, use the following command:

```
wcscli# wcscli –clear
```

Sample output:
Clear command history cache and clears the display screen on client window, displaying the CLI prompt.

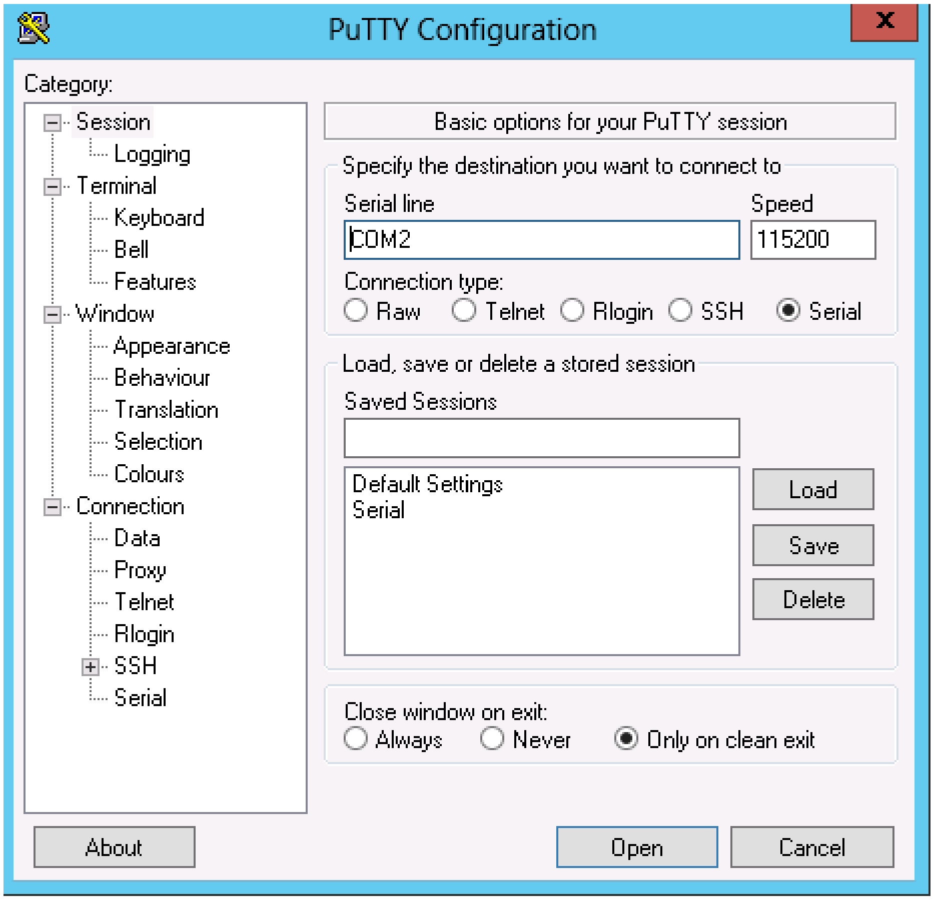
7.7 PuTTY Settings for serial connection

The recommended and supported tool for the CLI via serial connection is PuTTY.

This section describes the settings required for PuTTY.

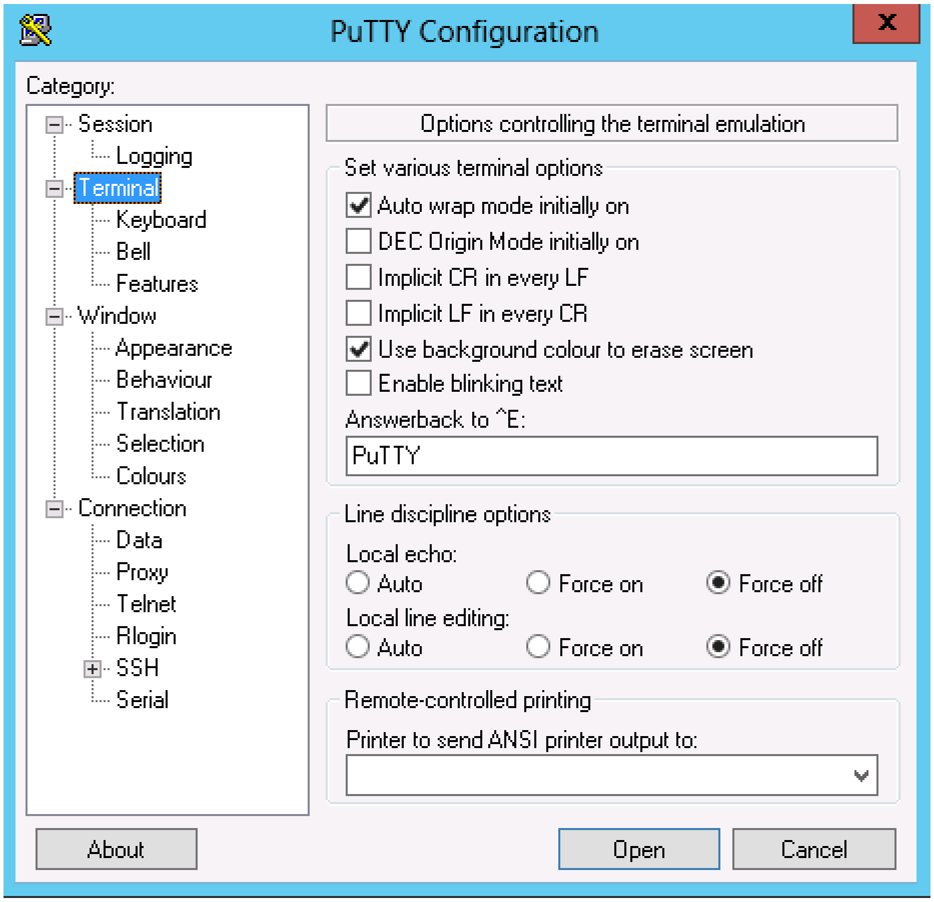
1.) Set the assigned COM port baud rate to (115200), as shown in Figure 14.
Note: COM port designations on your configuration may differ from this.

Figure 14: Set assigned baud rate



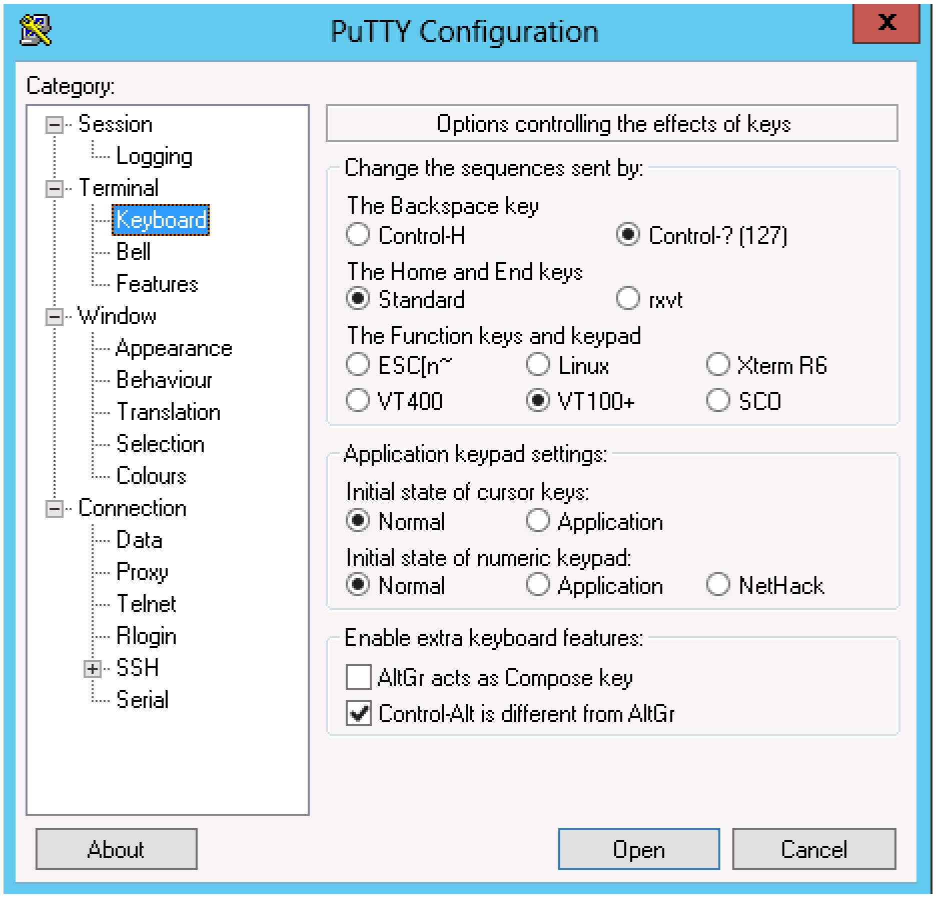
2.) Set terminal settings to local echo off and local line editing off, as shown in Figure 15. This allows the application to perform echo and line editing.

Figure 15: Set terminal settings



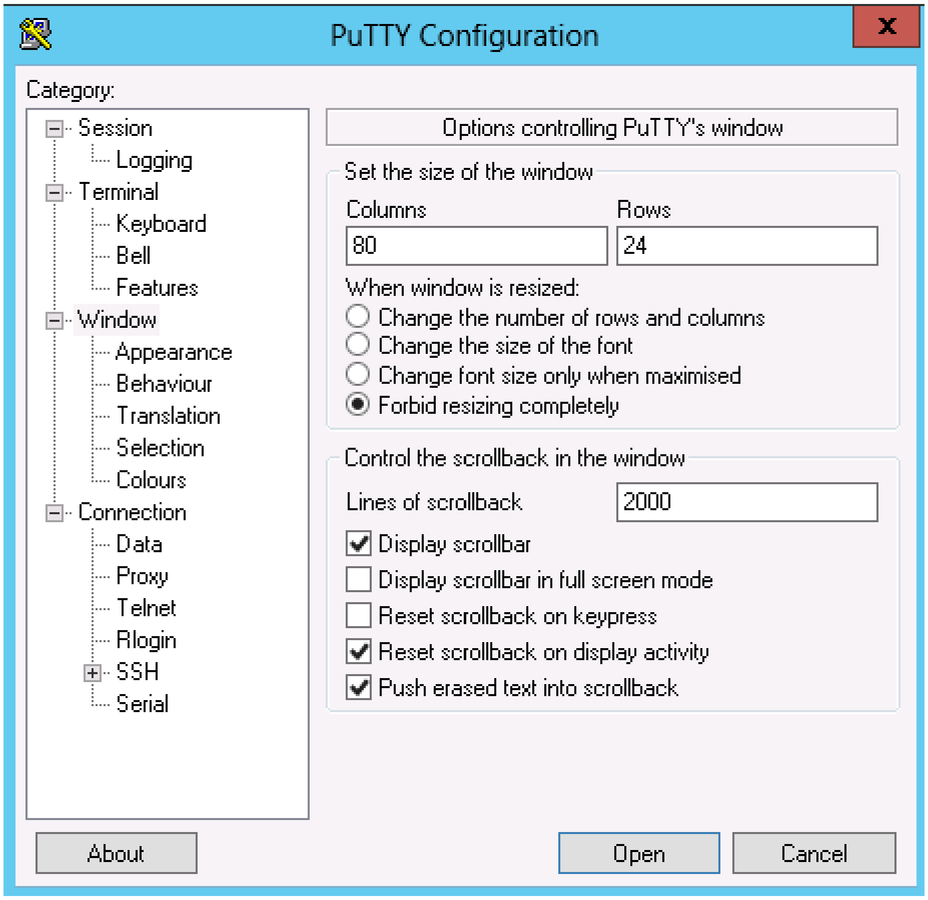
3.) Set the Keyboard to VT100+ and Backspace to be 'Control-?(127)', as shown in Figure 16.

Figure 16: Set keyboard and backspace settings



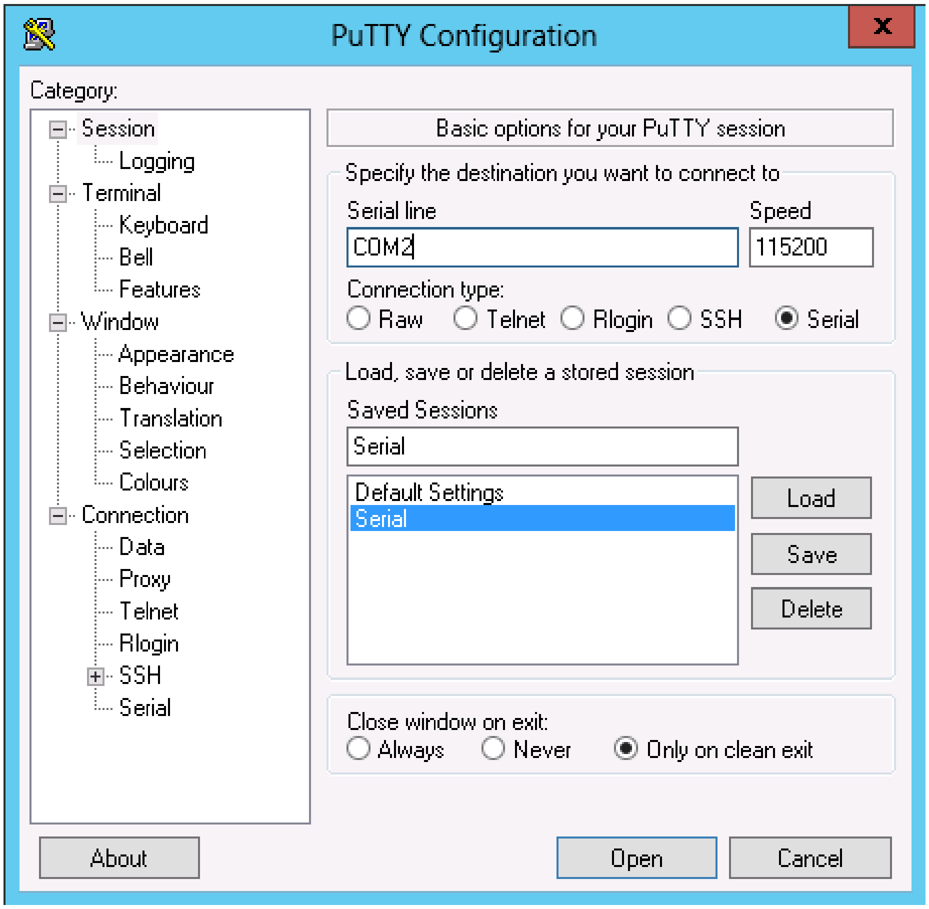
4.) Set the window size to default 80 characters and select forbid resizing option, as shown in Figure 17. This allows the PuTTY session output to match the command console output.

Figure 17: Set window size



5.) Save these settings to be loaded for future use and open the connection, as shown in Figure 18.

Figure 18: Save settings



7.8 Service Install

This section describes the various steps (see below) required to install the WcsCli service:

1. Create two folders under the C: drive. The folders can be named something like "WcsCliCOM5" and "WcsCliCOM6".
2. Copy the new WcsCli binaries under each of the two created folders in step 1.
    - Inside each of these directories, there is a file of type Config called "WcsCli.exe". Make sure you change the "COMPortName" key's value to the desired COM port to which the service attaches in that config. file. For example, for WcsCliCOM5 folder, update the WcsCli.exe config. file by changing the COMPortName to COM5.
3. Open a CMD prompt as "Administrator".
4. Navigate to each of the the two newly created folders.
5. Run"“C:\Windows\Microsoft.NET\Framework\v4.0.30319\InstallUtil.exe WcsCli.exe”.
6. Run “Net start WCSCLI*<COMPortName>.”*

7. Run “Services.msc” which will show the list of services and their statuses. You should see WcsCliCOM5 and WcsCliCOM6 listed as services. Also, you can refer to an installation log “WcsCli.InstallLog” that will get created under the same folder where the binaries reside.
8. Verify that the services are correctly installed and running. You can do that by checking for an existing service by running the following command in a CMD prompt:
    - “sc qc *<Service Name>*”; where Service Name are ‘WcsCliCOM5’ and ‘WcsCliCOM6’
    - For example, “sc qc WcsCliCOM5”