Server Management Command Line Protocol Specification (SM CLP)

Version 1.0

Status: Preliminary Standard

Publication Date: June 7th, 2005

DSP*0214*

Server Management Command Line Protocol Specification (SM CLP)

Version 1.0

Publication Date: June 7th, 2005

DSP0214

Status: Preliminary Standard

Abstract

The DMTF SM Command Line Protocol (SM CLP) specifies a common command line syntax and message protocol semantics for managing compute resources in Internet, enterprise and service provider environments.

Acknowledgments

The following persons were instrumental in the development of this specification:
Bob Blair, Newisys; Greg Dake, IBM; Jon Hass, Dell; Jeff Hilland, HP; Steffen Hulegaard, OSA Technologies; Arvind Kumar, Intel; Jeff Lynch, IBM; Aaron Merkin, IBM (co-editor); Christina Shaw, HP; Enoch Suen, Dell; Michael Tehranian, Sun; Perry Vincent, Intel (co-editor).

Table of Contents

List of Figures

List of Tables

1 Introduction

This document lays out the general framework for the SM Command Line Protocol (SM CLP). This document assumes that the reader is familiar with and has a working knowledge of the CIM Specification and Schema [1]. It is also assumed that the reader is familiar and has read the Server Management (SM) Architectural Specification [3], SM Managed Element Addressing Specification [4] and SM Profiles Specification [5]. In addition, it is assumed the reader is familiar with the conventions discussed in RFC2119 [6] for interpretation of keywords as well as the formation of UML strings [2].

1.1 Target Audience

This specification is intended to guide developers of implementations of the SM CLP and may be used as a reference by system administrators and other users of SM CLP implementations.

1.2 Conventions

The key words MUST, MUST NOT, REQUIRED, SHALL, SHALL NOT, SHOULD, SHOULD NOT, RECOMMENDED, NOT RECOMMENDED, MAY, and OPTIONAL, when they appear in this document, are to be interpreted as described in RFC 2119 [6].

The terms “default” and “implicit” are used in this document to describe aspects of the protocol as follows:

- Functions or behaviors are defined to be “implicit” if those functions are an integral part of the protocol definition and cannot be overridden by command or command options.
- Functions or behaviors are defined to be “default” if those functions are assumed to be in effect unless overridden or specified by the user via a command or command option.

1.2.1 Notation

Regular Expression (regex) and Augmented Backus-Naur Form (ABNF) will be used in this document to describe various aspects of the SM CLP specification. A complete SM CLP grammar in ABNF can be found in Section 4.14.

When command option names have multiple, supported forms, each form will be listed explicitly, separated by a comma. For example, the command option “level” has two acceptable forms, “`-l`” and “`-level`”. The specification text lists these alternatives as “`-l, -level`”.

The following fonts will be used to indicate specification elements:

`courier new`	Used to indicate literal characters used in the syntax expression.
italicized	Used to indicate the type or description of data to be inserted.
< >	Used to indicate terms in an expression
Capitalization	Used to indicate defined terms

Examples are given for information and are shown using Courier New font. In each example, the CLP command line is emphasized using bold text. The command output is shown in regular text font

1.3 Related Documents

[1] Common Information Model (CIM) Schema, V2.9, June 14, 2004 - Downloadable from http://www.dmtf.org/standards/cim/cim_schema_v29_prelim

[2] Unified Modeling Language (UML) from the Open Management Group (OMG) - Downloadable from http://www.omg.org/uml/

[3] “SM Architecture White Paper”, V1.0, DSP2001, 2005, DMTF Server Management Working Group – Downloadable from http://www.dmtf.org/apps/org/workgroup/svrmgmt/documents.php

[4] “SM Instance Addressing Specification”, V1.0, DSP0215, 2005, DMTF Server Management Working Group – Downloadable from http://www.dmtf.org/apps/org/workgroup/svrmgmt/documents.php

[5] "SM Profiles Specifications", V1.0, DSP0217, 2005, DMTF Server Management Working Group – Downloadable from http://www.dmtf.org/apps/org/workgroup/svrmgmt/documents.php

[6] RFC 2119, Bradner, S., "Key words for use in RFC s to Indicate Requirement Levels", BCP 14, RFC 2119, March 19

[7] “POSIX Utility Conventions”, The Open Group Base Specifications Issue 6, IEE Std 1003.1, 2004 Edition. Downloadable from http://www.opengroup.org/onlinepubs/009695399/basedefs/xbd_chap12.html

[8] “WBEM URL Syntax”, V1.0, August 18, 2003, DMTF, Downloadable from http://www.dmtf.org/standards/wbem/DSP0207.pdf

[9] “Uniform Resource Identifiers (URI): Generic Syntax”, IETF RFC 2396, August 1998 (http://www.ietf.org/rfc/rfc2396.txt)

[10] “Guidelines for new URL Schemes”, IETF RFC 2718, November 1999 (http://www.ietf.org/rfc/rfc2718.txt)

[11] “Registration Procedures for URL Scheme Names”, IETF RFC 2717, November 1999 (http://www.ietf.org/rfc/rfc2717.txt)

[12] “Extensible Markup Language (XML) 1.0 (Third Edition)”, World Wide Web Consortium, February 2004, (http://www.w3.org/tr/rec-xml)

[13] “CLP-to-CIM Mapping Specification”, V1.0, DSP0216, 2005, DMTF Server Management Working Group – Downloadable from http://www.dmtf.org/apps/org/workgroup/svrmgmt/documents.php

[14] “SM_CLP_Discovery_using_SLP_v1.0a.1.pdf”, V1.0, DSP0218, 2005, DMTF Server Management Working Group – Downloadable from http://www.dmtf.org/apps/org/workgroup/svrmgmt/documents.php

[15] “SSH Protocol Architecture”, Draft 17, IETF, October 2004 (http://www.ietf.org/internet-drafts/draft-ietf-secsh-architecture-17.txt)

[16] “SSH Transport Layer Protocol”, Draft 19, IETF, October 2004 (http://www.ietf.org/internet-drafts/draft-ietf-secsh-transport-19.txt)

[17] “SSH Authentication Protocol”, Draft 22, IETF, October 2004 http://www.ietf.org/internet-drafts/draft-ietf-secsh-userauth-22.txt)

[18] “SSH Connection Protocol”, Draft 19, IETF, October 2004 (http://www.ietf.org/internet-drafts/incoming/fixed/draft-ietf-secsh-connect-19.txt)

[19] “SSH Assigned Numbers”, Draft 07, IETF, October 2004 (http://www.ietf.org/internet-drafts/draft-ietf-secsh-assignednumbers-07.txt)

[20] “Telnet Protocol Specification”, IETF RFC0854, May 1983 (http://www.ietf.org/rfc/rfc854.txt)

[21] “Telnet Option Specifications”, IETF RFC0855, May 1983 (http://www.ietf.org/rfc/rfc855.txt)

[22] “Augmented BNF for Syntax Specifications: ABNF”, IETF RFC2234, November 1997 (http://www.ietf.org/rfc/rfc2234.txt)

[23] “Simple Mail Transfer Protocol”, IEFT RFC2821, April 2001 (http://www.ietf.org/rfc/rfc2821.txt)

[24] “Codes for the Representation of Names of Languages Part 2: Alpha-3 Code”, ISO 639-2, March 2002 (http://www.loc.gov/standards/iso639-2/langhome.html)

1.4 Terminology

Term	**Definition**
Absolute Target Address	A designation of target address which begins at the root of the containment hierarchy.
Addressing Associations	An association instance which is used by the MAP to construct the UFiP to an instance referenced by the association instance.
Admin Domain	The set of Managed Elements, Logical Devices, and Services for which the MAP has management responsibilities.
Association Class	Identifies the CIM Class of an association.
Client	Any system that acts in the role of a client to a MAP.
CLP Target	A Managed Element whose properties, behavior, UFcT, etc. are wholly defined by the profiles approved for use with the CLP.
Command Line Protocol (CLP)	The human-oriented command line protocol defined by the System Management Architecture for Server Hardware, used for managing systems.
Command or	A text message containing the complete

Term	Definition
Command Line	expression of a management action, including a command verb, and if specified, a command target, options and option arguments, and properties and value.
Command Processor	The logical entity within a MAP responsible for parsing, interpreting, and executing incoming commands and returning responses.
Command Response	Response returned by the CLP service to a Client when a Command is issued. This consists of Command Status and Command Results.
Command Results	The actual results of a successful command returned as part of the Command Response.
Command Status	Information returned by the CLP Service to a Client describing the overall status of a Command.
Command Status Data	Detailed information returned by the CLP service to a Client describing the status of the Command as part of the Command Response.
Core Properties	The profile which owns the definition of the class does not stipulate whether these properties should be included in instances of the class.
Current Default Target	The CLP session environment setting that establishes a default base address for all command targets that are expressed as a relative target address and is used as the command target if no command target is specified in a command entered.
Implicit Command Target	The target acted upon is inherent to the command being executed. The command does not act upon either the CDT or a target specified as part of the command. The `cd` command is an example of a command that acts upon an Implicit Command Target.
Keyword	A text string token that is recognized and reserved by the CLP to have a specified meaning when used in command output and input.
Local Addressing Service	The entity responsible for discovering, enumerating and determining the addresses of

Term	Definition
	Managed elements within the Local Domain.
Local Domain	The set of Managed Elements, Logical Devices, and Services for which the MAP has management responsibilities.
Manageability Access Point (MAP)	A service of a system that provides management in accordance to specifications of the DMTF System Management Architecture for Server Hardware.
Managed Element	The finest granularity of addressing which can be the target of commands or messages, or a collection thereof.
Managed Element Access Method	The method by which a Managed Element performs a unit of work.
Managed System	A collection of Managed Elements that comprise a Computer System for which the MAP has management responsibilities.
Management Service Core	The logical entity that contains the core services set of the MAP.
Non-addressing Association	An association instance which is not used by the MAP in constructing the UFiP to any instances the association instance references.
OEM Properties	Properties added to instances of a class by an OEM vendor.
OEM Target	A Managed Element whose properties, behavior, UFcT, etc are outside the scope of this specification and are vender dependent.
OEM Verbs	Verbs defined by an OEM vendor which are outside the SM CLP specification.
Operation	An identifiable activity of a MAP.
Option	A term of the command line that selects a particular behavior of a command verb.
Option argument	Input data passed to the command verb in relation to an option that selects a particular value for that option.
Property	An attribute of the command target.
Recommended Properties	The profile which owns the definition of the class recommends that instances of the class have values for these properties.

Term	Definition
Relative Target Address	A designation of target address in relation to the Current Default Target as opposed to an Absolute Target Address.
Required Properties	The profile which owns the definition of the class requires that instances of the class have values for these properties.
Reserved string	A text string token that is recognized and reserved by the CLP to have a specified meaning when used as an option argument, argument value, or property value.
Reserved Target	A valid value for a command target term which has a special meaning defined by this specification. The meaning of a Reserved Target can not be changed by a user nor can additional Reserved Targets be created.
Resultant Address	Target address after applying the rules of target address precedence.
Resultant Target	The effective target for a command after applying the Target Address Precedence rules.
Service Access Point	The representation of endpoint or interface into a service, such as the CLP.
SM CLP Verb	Verbs defined by the SM CLP specification.
Target	The string tag of a manageable element of a system, used in a command line to indicate which element of the system the command is applied.
Target Address	A string value used in a Command Line to identify the target for a command.
Target Class Addressing	A method of selecting Managed Elements within a container based on the UFcT of the element.
Text session or text-based session	An active connection to a service whereby the user can enter text-based messages and receive text-based data. Examples of commonly-used text sessions include Telnet and Secure Shell.
Transport	The layers of the communication stack responsible for reliable transportation of commands and message between the Client

Term	Definition
	and the MAP.
User Friendly instance Tag	User friendly identifier for a specific instance of a CIM class. A User Friendly instance Tag is constructed by concatenating an integer suffix to the UFcT for the CIM class.
User Friendly selection Tag	Short-hand notation for selecting all instances of a given class. A User Friendly selection Tag is constructed by concatenating the UFcT for a class with the character *
User-Friendly class Tag	A short human friendly alias for a CIM class name. It has the same properties and methods as the CIM class it represents.
User-Friendly instance Path	The unique path to an instance formed by concatenating the UFiT s of each instance from the root instance to the terminating instance.
Verb	The string name of a command, used as the first term of a command line.

1.5 Acronyms and Abbreviations

Term	Definition
CDT	Current Default Target
CIM	Common Information Model
CLP	Command Line Protocol
MAP	Manageability Access Point
ME	Managed Element
NIC	Network Interface Card
OEM	Original Equipment Manufacturer
SMASH	System Management Architecture for Server Hardware
SSHv2	Secure Shell Version 2
UFcT	User-Friendly class Tag
UFiT	User-Friendly instance Tag
UFiP	User-Friendly instance Path
WBEM	Web Based Enterprise Management

2 SM CLP Overview

This section contains an overview of the Command Line Protocol (CLP). This includes the goals behind creating the CLP and the specific problems which it attempts to resolve. In addition, this section will lay the groundwork for the sections that follow by detailing the background and assumptions of the CLP. This includes the architecture assumed in the design of the CLP and the components within that architecture.

2.1 Problem Statement

The fundamental problem which is the impetus behind this specification is the growing need to rely on multi-vendor, out-of-band hardware and software management solutions as core components of interoperable, heterogeneous enterprise-wide management solution. By the extending the DMTF Specifications to include a CIM-based command line protocol for managing out-of-band and out-of-service devices, the DMTF comes closer to realizing it's vision of enabling end-to-end, multi-vendor interoperability in management systems.

2.2 Principal Goals

The principal goal of this specification is to define a light-weight, human-oriented command line protocol which is also suitable for scripting environments. This includes a direct mapping to a subset of the CIM Schema. The command line protocol will specify the syntax and semantics used to allow the manipulation of the Managed Elements within servers, as collections or individually.

2.3 Solution

The solution proposed in this document is a command line protocol (CLP), which is transmitted and received over a text message-based transport protocol. The CLP is defined as a character-based message protocol and not as an interface, in a fashion similar to SMTP RFC 2821 [23].

The CLP is a command/response protocol, which means that a text command message is transmitted from the Client over the transport protocol to the Manageability Access Point (MAP). The MAP receives the command and processes it. A text response message is then transmitted from the MAP back to the Client.

The CLP is designed to work over existing character oriented transports. The specification contains mappings to Telnet and SSHv2, but any transport capable of carrying command/response message data of the type specified herein may be suitable for use as a transport.

The CLP enables internationalization by providing a mechanism for the Client to indicate to the MAP the language desired by the Client. Provided the MAP supports the requested language, output data will be presented to the user with the appropriate translations. This version of the CLP does not support specific internationalization of user account names and passwords as they are NOT REQUIRED to be in any specific language. In addition, the CLP input (commands & syntax) is not translated since CLP syntax is itself its own language.

The CLP allows for extensibility through four different mechanisms: verbs, targets, target properties, and option names and option arguments. The conventions contained herein allow for implementers to extend the interface in a non-conflicting mechanism that allows for differentiation and experimentation without encroaching upon the standard CLP syntax and semantics.

2.3.1 General Syntax

The general syntax for the CLP is of the form:

```
<verb> [<options>] [<target>] [<properties>]
```

The `verb` term refers to the specific command or action taking place. These are covered in detail in Section 4. The list of verbs includes those that establish and retrieve data (“`set`” and “`show`”), create and remove records or instances, (“`create`” and “`delete`”), change the state of a given target (“`reset`”, “`start`”, “`stop`”), manage the current session (“`cd`”, “`version`”, and “`exit`”) and provide command information (“`help`”).

The `target` term indicates the address or path of the target of the command. The format of this term is discussed in the SM Managed Element Addressing Specification [4]. This term can be an individual Managed Element, such as a disk, a NIC, the MAP itself, or even a service, such as the transport. This term can also be a collection of Managed Elements supported by the MAP, such as a system. There can be only one target term specified per command.

Command options always modify the action or behavior of the verb. Options MAY appear immediately after the verb on the Command Line and MUST be preceded by a hyphen (“-“). They provide features such as changing the output format, allowing the command to apply to nested levels, requesting the version of the command be displayed, and requesting help. Note that there MAY be zero or more option terms per command. For more information, see Section 5.

Command target properties are attributes which may contain values associated with a target that are needed to process the command. Command target properties identify properties of the target’s class which are to be retrieved or modified by the command. Valid command target property names are documented in the MOF file which defines the class. Implementations MUST use the property name defined in the MOF file to identify the property of the class.

The properties themselves are manipulated with the commands in Section 4. If a value is to be assigned to a property, the syntax MUST be of the form “<property name>=<value>”. There MAY be zero or more property terms per command.

2.3.2 Architectural Assumptions

There is an underlying assumption that the architecture the CLP is built upon, or is an interface into, is a CIM Server implementation. The CLP is organized around management tasks mapped to operations on CIM instances. It does this by retrieving or changing properties and invoking methods established in instances of the CIM Schema. The mapping of CLP commands to CIM elements is documented in the CLP-to-CIM Mapping Specification [13] to aid implementers and consumers of this specification.

The CLP consists of a set of specific functions intended for operational control of the server hardware and rudimentary control of the operating system. It is not intended to be a complete interface into managing the operating system. Therefore, the CLP contains the commands necessary to operate on a proper subset of the CIM Schema as defined in the CLP-to-CIM Mapping Specification [13].

The CLP is also architected to work over existing transports. It is assumed that the transports will provide the authentication and encryption necessary for the protocol. Role-based command use authorization is included in the CLP, but the architecture assumes that the CLP relies on the underlying transport for any access security and authentication. The CLP architecture is documented in SM Architecture White Paper [3] .

2.4 Architectural Concepts of the SM CLP

The following sections describe some of the key concepts of the SM CLP. A detailed statement of the architecture of the SM CLP is available in the SM Architecture White Paper [3] .

2.4.1 Physical Connection

The physical connection and media between the Client and the MAP are outside the scope of this specification. Any physical connection which is capable of running one of the supported transports should be able to support the CLP. Since the supported transports themselves can run over IP, it is safe to assume that the CLP can be transmitted and received over any media or physical connection that supports IP. However, this does not limit support for the CLP to physical media or connections which support IP. In fact, it is reasonable for an implementation to be created where the protocol never leaves the managed server.

2.4.2 Transport Management

This specification includes sections detailing the mapping of the CLP over two transport protocols: Telnet and SSHv2. The CLP is designed to be transport independent, but mappings to transports other than these two are outside of this specification.

The Client is the terminus of one end of the connection; the CLP Service implementation is the other. CLP Service implementations MUST manage the transport and the sessions which occur over the transport.

2.4.3 Authentication, Authorization & Encryption

The CLP Service does not perform any authentication or encryption. It relies entirely upon the transport to perform these functions. Session transport requirements are documented in Section 6.3.

To accommodate a single basis for user authorization, the user account database required by the transport is expected to share the user information with the CLP Service once the user is logged in. For more information, see Section 6.3.

The CLP Service authorizes commands through the use of authorization groups. Each user account is a member of a group, which MUST be assigned when the account is created. For more information, see Section 6.1.

The CLP contains commands for the creation, removal and modification of user accounts, including authorization and access rights. For more information, see Section 4.

2.4.4 Sessions

Sessions between a Client and a CLP Service are established over a transport protocol. Once the session has been authenticated, the Client can begin to submit commands using the CLP Service. Each session has a unique context within the MAP. Within this context, the CLP Service keeps track of session characteristics. Implementations supporting the CLP MUST maintain a session context to track session characteristics. Examples of these characteristics include the current default command target, currently selected output mode, current output language, and the current user and session identifier. There are commands for manipulating the session characteristics included in the CLP. For more information, see Section 6.2.1.

2.4.5 Input Editing

The CLP is a command / response protocol. CLP implementations MUST receive and parse an entire Command Line, complete with verb, target, options and properties. The CLP Service MUST NOT allow any interactive input or data editing. This does not preclude a Vendor from providing such capability associated with the Client implementation, but any such capability is outside of the scope of the specification.

2.4.6 Command Line Protocol Service

The CLP Service is responsible for providing and enforcing the syntax and semantics of the CLP. Implementations of the CLP service MUST support being configured using the CLP and MAY support configuration through other interfaces This includes starting, stopping and changing the attributes of the service.

2.4.6.1 CLP Service Access Point

The transport session is established to the Command Line Protocol Service Access Point. The access point represents the physical and logical communication mechanism through which the CLP Service receives incoming connection requests. The CLP provides the mechanisms necessary to enable, disable and configure the Access Point. If the CLP Service Access Point is instantiated using an Ethernet NIC, the SSHv2 protocol and the supporting protocol stacks, the CLP implementation MUST provide mechanisms for configuring the NIC, TCP, IP and SSHv2.

2.4.6.2 Operation Management

All commands submitted through the CLP Service create jobs within the MAP. There is one and only one global job queue within the MAP. Implementations MUST track all jobs using this single job queue.

Operations follow the CIM_Job schema, as defined in later sections. The CLP supports commands to query jobs, retrieve the interim status of jobs, retrieve the final status of jobs, and delete jobs. Operations are covered further in Section 3.1.6.

2.5 Use Cases

This section describes the intended features, functions and uses of the CLP. Note that the CLP is not limited to these functions, but these are the specific uses for which the CLP was intended.

The CLP is designed to apply to a number of server topologies. This includes, but is not limited to, stand alone servers, rack mounted servers, blades, partitioned servers & Telco servers. It is also suitable to manage any necessary enterprise components, enclosures, chassis, racks and power supplies necessary to utilize servers.

The CLP provides the ability to enumerate and configure server hardware. This includes discovery of the current hardware configuration and properties, system settings and local IO devices. The CLP provides some amount of configuration for local disk drives, including local arrays. The intention of providing this support is to allow initial logical unit creation for installation and/or provisioning. It is not intended that the CLP Service be the primary interface for managing mass storage, since these standards and access points exist in the industry.

The CLP also includes the ability to select, control and initiate the transfer of images. This is provided through the `CIM_SoftwareInstallationService` as well as the ability to control the boot configuration of any supported server. In addition, support for heartbeat and operating system status information is included.

Server state control is included in the CLP. This includes power control, intervention capability (to halt, reset or shutdown a server) and mechanisms to initiate a dump of the operating system.

Access to some system resources is included in the CLP as well. This includes access and manipulation of any accessible logs, the ability to view and set remote status displays, LED s and alarms, the ability to configure alert destinations and the ability to initiate a session with a remote text-based console device.

The CLP also supports normal expected user session functions such as help, version information and the ability to exit or terminate a session.

2.6 Known Limitations

First and foremost, while CLP commands are mapped to CIM methods and operations, the CLP is not intended to be a complete mapping to every CIM method, property and/or operation. The CLP supports a sufficient subset of CIM Server features to enable the CLP to be the primary locus of interaction for server management, regardless of server type, physical connection or operating system state.

Another known limitation pertains to the intended Client. The CLP is primarily focused on an interactive experience with a human user or simple script. It is not intended to be the primary interface for advanced server management software to utilize in order to manage their hardware. Consequently, the format of the commands and their responses as well as the CIM methods, properties and operations supported may not be sufficient for the CLP Service Access Point to be the primary interface for advanced server management software.

2.6.1 Scope

The following subjects are within the scope of this document:

- Command Line Protocol syntax and semantics
- input format and output format
- accessing and traversing the target address space
- error handling and semantics

– session management including mapping to supported transports
– session characteristics
– operation processing and reporting

The following subjects are outside the scope of this document:

– control command verbs - such as loop control, conditionals or prompting
– regular expressions - such as mathematical or logical expressions
– command editor environment
– Client's shell environment
– physical interconnects
– complex data, data types or objects
– operation error precedence

3 SM CLP Protocol

3.1 Semantics

The Command Line Protocol (CLP) defines the form and content of messages transmitted from and responses received by a Client within the context of a text-based session between that Client and the CLP Service for a Manageability Access Point (MAP).

The CLP consists of a set of command verbs that manipulate command targets representing Managed Elements (ME) that are within the scope of access by a MAP.

Each CLP interaction consists of a command line transmitted to the CLP Service and a subsequent response transmitted back to the Client. Each command transmitted generates one and only one response data transmission to the Client.

3.1.1 Command Verb

A CLP command verb retrieves information about a target or initiates a state change of the target. A command verb MUST always generate a Command Response. A CLP interaction MUST consist of one and only one command verb.

3.1.2 Command Options

CLP command options control the behavior of the command verb. All CLP option names are standard across the CLP command verb set. Implementations of the CLP MUST NOT redefine the usage of a CLP option name across different CLP command verbs.

3.1.3 Command Target

The command target address identifies the specific Managed Element or association that is to be affected by the command verb. All CLP commands have a command target, whether explicitly or implicitly identified. An explicitly-identified target is a target address path that is included in the command line entered. An implicitly-identified target is a target that is not identified in the command line entered but is either dictated by the command verb itself or is referenced from the session environment variable “Current Default Target”. Implementations MUST interpret command verbs submitted to the CLP only for the identified target (or targets see Section 3.1.3.6 Target Grouping by Class). The CLP also defines Reserved Targets. Reserved Targets are strings whose interpretation is defined by this specification. Reserved Targets can be used to construct the command target term. Implementations MUST NOT define Reserved Targets beyond the ones defined in this specification. Implementations MUST interpret Reserved Targets in accordance with the meaning assigned to them by this specification.

This version of the SM CLP Specification supports the SM ME Addressing Specification [4]

3.1.3.1 Current Default Target

A default target address MUST always be in effect during a CLP session. This target, called the Current Default Target, is used by the Command Processor to determine the Resultant Target for the command according to the rules of target address precedence defined in Section 3.1.3.2 below.

A session's Current Default Target is only modified if set explicitly by a user. The rules for establishing and maintaining a session's Current Default Target are as follows:

- Implementations MUST set the Current Default Target to the instance address of the root of the administrative domain of the MAP upon CLP session activation (i.e. defaulted to the object that represents the top of the hierarchy managed by the MAP). Implementations MUST use the same initial value for the CDT for all users and MUST NOT allow the initial value to be configurable by a user.
    - Therefore, Current Default Target is never <null> at CLP session start.
    - If a user sets the CDT to another target address and then ends the session, on the next login by the user to a CLP session of the same MAP, the CDT MUST be set to the MAP default.
- The Command Processor MUST NOT allow the user to explicitly set the Current Default Target to an invalid target address during the session.
- If the user attempts to set Current Default Target to a target address for a Managed Element that is not responding or is not recognized as being in the scope of the MAP, then the attempt fails, the implementation MUST NOT change the Current Default Target and MUST return a Command Status of COMMAND EXECUTION FAILED and a CIM Status CIM_ERR_NOT_FOUND.
- If a Managed Element becomes non-responding at some point after it has been set as the Current Default Target, then the implementation MUST return an appropriate error code and MUST keep the Current Default Target address set to its current value until the user explicitly changes it to a different, valid target address. This prevents spurious drops in communication with the Current Default Target from causing an automatic change in the Current Default Target. For example, a user sets the CDT to "`/system1/disk3`". Some time later, `/system1/disk3` becomes unresponsive. As long as the user does not target the CDT, there is no impact on the user's current session. If the user decides to target `/system1/disk3` by omitting the <target> term of the current command, the CLP implementation would discover that the target ME, `/system1/disk3`, is unresponsive and return an error code.
    - Since unpredictable, undesired results would occur if the Current Default Target is automatically changed, the Command Processor MUST NOT automatically change the value of the Current Default Target, for any reason.

3.1.3.2 Target Address Precedence

The implementation MUST determine the Resultant Target of a command in the order that follows:

1) If the command verb has an Implicit Command Target, then the Implicit Command Target MUST be selected as the Resultant Target.

2) If a target term is specified in the Command Line, the implementation MUST apply the Target Address Evaluation Rules to derive the Resultant Target. These rules detailed in Section 3.2.1.3.5.
3) If the Command Line did not include a target term, the Current DefaultTarget (set in the session environment) MUST be selected as the Resultant Target.

If the Resultant Target is determined by the implementation to be invalid, then the implementation MUST NOT execute the command and MUST return a Command Status of COMMAND EXECUTION FAILED and a CIM Status of CIM_ERR_NOT_FOUND in the Command Response data. CLP commands that have an Implicit Command Target may still accept a command target term (ie. the `cd` command) or may not accept a command target term (ie. the `exit` command). Each commands' usage of the command target term is documented in the sub-section of Section 4 devoted to the command.

3.1.3.3 Target Managed Element Object Model and Semantics

The CLP is designed for administrators and scripts that manage systems. At the same time, the CLP conforms to the object model described by the Common Information Model (CIM) Specification v2.9 [1].

The CLP defines a general command verb set used to manipulate Managed Elements. In many cases, CLP verbs relate directly to typical object interactions, such as "set property value", "read property value", "put into a particular state", etc. In other cases, CLP verbs are interpreted in the context of the Managed Element and map to particular methods of that Managed Element's class.

The CLP verb definitions in Section 4 describe each CLP command verb in detail.

The CLP-to-CIM Mapping Specification [13] describes the full mapping of the CLP to the Common Information Model. For each CIM class, the CLP-to-CIM Mapping Specification [13] describes the behavior of commands applied to a target instance of the class. The specification also describes the property names of those targets that are referenced or manipulated by the command.

In the CLP, Managed Elements have the following aspects:

- Properties – These are properties of the Managed Element itself and are described in more detail in Section 3.1.4
- Contained Targets – This is the set of Managed Elements immediately contained in the Managed Element according to the rules of instance containment described in Section 3.1.3.4.
- Associations – This is the set of associations which reference the Managed Element. They are described in more detail in Section 3.1.5.
- Verbs – This is the set of commands which are applicable to the Managed Element. The SM CLP verbs are described in Section 4.

3.1.3.4 Target Addressing

CLP target addressing is defined by the SM Managed Element Addressing Specification [4]. CLP implementations MUST only operate on targets that adhere to the SM Managed Element Addressing Specification [4] or to the rules for identifying OEM targets described in Section 3.2.6.

The specific arrangements of Managed Elements that a MAP may expose are documented in the SM Managed Element Addressing Specification [4] and the SM Profile Specifications [5]. The SM CLP separates Managed Elements into two categories of targets; CLP Targets and OEM Targets. CLP Targets are Managed Elements whose properties, behavior, UFcT, etc. are wholly defined by the profiles approved for use with the CLP. OEM Targets are Managed Elements whose properties, behavior, UFcT, etc are outside the scope of this specification and are vendor dependent.

3.1.3.5 Aggregated Targets

Command targets may be an aggregation of underlying components. These underlying components may or may not be visible in the address space of the MAP. When the command target is comprised of aggregated parts, the Command Processor MUST interpret the command for the aggregated target as a single job and return a Command Response accordingly. The implementation MUST ensure that a command initiates the appropriate action with respect to an aggregated Managed Element.

The implementation MAY rely on the target Managed Element to implement the aggregated command function. One example of an aggregated target is an operating system. When a user issues a `stop` to an operating system instance, a single job is spawned. The operating system may attempt to shutdown applications running within it. This action taken by the operating system is not modeled with jobs and the results for individual applications are not displayed in the Command Results.

3.1.3.6 Target Grouping by Class

“Grouping” describes the ability of a user to explicitly select more than one target for a command at the time the command is issued. The only method defined by the CLP for addressing multiple target addresses with a single command is Target Class Addressing.

If the final term of the target address is a UFsT, the Command Processor MUST interpret the target address as a selector for all Managed Elements of the class specified that are in the immediate container.

For example, the target address “`/system3/disk*`” instructs the Command Processor to issue the command for all targets with an UFcT of “`disk`” in the container “`/system3`”.

Implementations MUST support Target Class Addressing for the `show` command. Implementations MAY support Target Class Addressing for the `create` and `delete` commands. Implementations MUST NOT support Target Class Addressing for CLP commands other than the `show, create,` and `delete` commands.

3.1.4 Command Target Properties

Target properties are identifying and descriptive information related to and defined by the target. Target properties are identified by property names. Each class of target defines a set of valid property names. Valid property names are found in the CIM Schema Managed Object Files (MOFs). Vendors MAY support vendor-specific property names according to the rules defined in Section 3.2.6 of this specification. The SM CLP recognizes four categories of properties. Required Properties are properties which the profile defining the class of the target deems required for compliance with the profile. These properties will always be present for the instance across implementations. Recommended Properties are properties which the profile defining the class of the target deems recommended. These properties should be present for instances of the class across implementations. Core properties are properties which are defined for the class of the target in the CIM schema. The profile which defines the class does not require these properties, however they may be used because they are defined in the MOF. Note this includes any deprecated properties which are still defined in the MOF. They may be present across implementations. OEM Properties are properties defined by an OEM vendor for a target. These will not be consistent across different vendors' implementations.

3.1.5 Associations

The SM Managed Element Addressing Specification [4] specifies the association classes that may be used to construct paths to address any Managed Element appearing within the scope of the MAP. The SM Managed Element Addressing Specification [4] identifies these as Addressing Associations. Additional associations that are not used for addressing may exist and express relationships between Managed Elements. The SM Managed Element Addressing Specification [4] identifies these as Non-addressing Associations. Associations represent a special type of target. Association instances are not assigned UFiTs. For a given association class, instances are uniquely identified by the Managed Element instances they reference. They can be addressed using an extension of the target addressing syntax. Section 3.2.1.3.4 describes how to use the association separator "=>" to address association instances. Sections 4.9 and 4.10 illustrate the usage of the `set` and `show` commands respectively.

3.1.6 Command Processing

Implementations of the CLP MUST return a Command Status of COMMAND PROCESSING FAILED and a Processing Error of COMMAND SYNTAX ERROR if a completely formed command is not contained in a single text message transmission of the underlying transport protocol. Implementations MUST validate every Command Line against the grammar specified in Section 4.14. If a Command Line does not comply with the grammar defined in Section 4.14, the implementation MUST return a Command Status of COMMAND PROCESSING ERROR and a Processing Error of COMMAND SYNTAX ERROR. A command is processed by the Command Processor component of the SM architecture. The Command Processor returns a response and control to the Client for each command received. Commands and the subordinate activities generated when processing commands are tracked by the implementation. See Section 3.1.6.1 for more information.

3.1.6.1 Job Visibility

A job is defined as an identifiable activity of a MAP. The implementation spawns and manages jobs for a command and any subsequent actions that are taken to carry out the command request. When an implementation receives a command, the command becomes a job. The command job may generate additional subordinate Managed Element jobs in order to complete the task requested by the command verb but the implementation MUST NOT expose these jobs through the CLP. The command job MUST continue to exist until any and all jobs spawned by the command have completed, a Command Response has been returned to the Client, and the TimeBeforeRemoval has not expired.

By default, the implementation MUST return a Command Response and session control to the Client within a reasonable period of time, regardless of the status of command execution. . Returning a response and control prevents the CLP session from blocking indefinitely should a command take an unreasonable amount of time. The definition of "a reasonable period of time" is implementation specific. Any mechanisms for modifying this value are outside the scope of this specification and implementation specific.

When the implementation returns a Command Response synchronous with completion of the command job, the Command Response MUST contain the Command Status and the complete Command Results.

When the implementation returns a Command Response before the command job completes, the Command Response MUST contain the Command Status of COMMAND SPAWNED and the Job Identifier for the continuing command job.

The implementation MUST manage the command job until it completes and persist the Command Status when complete, identified by the Job Identifier. The implementation is NOT REQUIRED to maintain the Command Results for the command. Implementations MUST recognize the Job Identifier as an identifier used to obtain status information about the continuing command and to retrieve the Command Status when the command job is complete. The Job Identifier MUST be the Instance Suffix for the UFiT of the instance of CIM_ConcreteJob used to represent the job in the job queue. Once the Command Status holding time has expired, the corresponding job is deleted and the Job Identifier released. Implementations MUST also implement a user-controlled holding time for Command Status, controlled by a per-command option. Usage of this option is documented in Section 5.8.

Jobs are themselves Managed Elements of the system; therefore, the implementation MUST support display of information about a job and the ability to request a job to stop before completion using CLP commands. If the implementation is unable to start a job to execute a command, the implementation MUST return Command Status of COMMAND PROCESSING FAILED and a Processing Error of QUEUE FULL.

3.1.6.2 Error Handling

The CLP Service checks CLP commands for syntax and semantic errors. When a command is formed incorrectly or the command cannot be executed for the specified target because the target is not in an appropriate state, the implementation MUST return an error/exception status in the Command Status data and no Command Results.

When a command is syntactically correct and semantically appropriate, the implementation MUST attempt to perform the appropriate operations for the command target. If one or more operations fail, the implementation MUST return a Command Response containing both a Command Status and Command Results, including any error/exception information generated by the command target.

Implementations MUST include all detected syntax errors first in the Command Status data. Implementations MUST include all detected semantic errors in the Command Status data after all syntax errors have been included.

If the Command Processor detects a syntax error, the implementation MUST report an error and the implementation MUST NOT alter the state of the target Managed Element. If the Command Processor detects a semantic error, the implementation MUST report an error and SHOULD NOT alter the state of the target Managed Element.

Command Status output is defined by the CLP and is documented in Section 3.2.2 below.

Command Results output is defined for each CLP command and is documented in Section 4 below.

3.1.7 SESSION Reserved Target

Sessions with the CLP service are represented as Managed Elements within the address space of the MAP. This enables users of the CLP to manage attributes of the session using standard CLP commands. Users will frequently wish to manage attributes of their own session with the CLP service. In order to simplify accessing the Managed Element which represents the user's session, the CLP defines a reserved keyword "SESSION" ("env"). Implementations MUST interpret the keyword "SESSION" as the fully qualified path to the Managed Element that represents the session of the user issuing the command.

3.1.8 UFiT Assignment

Individual Managed Elements within the address space of the map are uniquely identified by an UFiT. The UFiT is constructed by concatenating an integer suffix to the UFcT for the class of the Managed Element. The rules for assigning and maintaining UFiTs are defined in the SM Managed Element Addressing Specification [4].

3.1.9 Input Data

Implementations of the CLP MUST NOT allow inclusion of input data embedded in the text session (sometimes referred to as "streaming"). A data file or stream can be selected for input to a CLP command by reference only using a command-specific option. For example, to input a firmware image for reloading firmware, the option `–source <URI>` is used.

3.1.9.1 Data passed in on command line

Data MAY be provided as input to a command via an option argument or a property value. Each command verb defines the options and option arguments that it accepts. The class of command target defines the properties that are accepted by the command. The length of the data provided should be less than or equal to 255 characters. Implementations MUST enforce a maximum length of 255 characters for any single term in a command. If a single term exceeds a maximum length of 255 characters, the implementation MUST NOT execute the command and MUST return a Command Status of COMMAND PROCESSING FAILED and a Processing Error of COMMAND SYNTAX ERROR.

3.1.10 Output Data

The semantics of CLP command output data are defined by the CLP output data schema. The CLP output data schema defines the attributes and organization of the data elements returned in response to a CLP command. When a command is issued, in the absence of transport errors, the implementation MUST return a Command Response data element as the response before any other text that may be appended. Implementations MUST include a Command Status data element in the Command Response data element and MAY include a Command Results data element. Implementations MAY return Command Results data. Implementations MUST always return Command Status data first in a Command Response.

The CLP syntax defines the keywords and value specifications (data types and ranges or domains) that are used to document CLP command output.

CLP output data is rendered in character form according to the rules set forth in Section 3.2.4 and Section 3.2.1.1. By default, a CLP session renders all output data in an output format called "text". Text output format is defined to be human-readable text that is not suitable for machine-parsing and will vary across implementations. In order to parse output data according to the CLP output data schema, one of the CLP's structured output formats should be selected. These formats are "clpcsv", "keyword", and "clpxml". Implementations MUST use the attribute keywords consistently to identify output data elements in each of the output formats. For example, the keyword "status" is rendered as a column heading keyword in "clpcsv" format, as a keyword in a "keyword=value" expression in "keyword" format, and as an XML tag in "clpxml" format.

Text mode output is optimized for human readability and, as such, may vary from implementation to implementation and in situation to situation. For example, when in text output mode, Command Status Data may not display the numeric value of the status code when the command is successful but instead simply describe the resulting condition of the Managed Element once the command has completed. For example, when the command "stop system1" completes successfully, the text mode output may simply state "system1 stopped.". By contrast, when a structured output mode is selected, the implementation MUST include all of the required and supported optional elements of the Command Status Data in the output.

The CLP Output Data Schema is documented in Section 3.2.2

The following sections contain data definitions for the output data elements in the CLP Output Data Schema.

3.1.10.1 Command Status Data Elements

Command Status data elements communicate the status of the command to the Client. Depending on the command verb and user-selected options, a command may continue to run after returning a response and control to the Client. In this case, the Command Status data also includes a Job Identifier that can be used to display the status of the command job at a later time.

The implementation MUST return the Command Status data element as the first data element in a Command Response. Implementations MUST NOT include any data elements or properties in the Command Status data element other than those defined here.

The Command Status data element includes

- a Status property
- a Status Tag property
- a Processing Error property
- a Processing Error Tag property
- one or more Message data elements

- a Job data element

The Status property describes the outcome of the command. The Command Status documents the disposition of the command from the perspective of the CLP Session protocol. Status for each command is one of four values: COMMAND COMPLETED, COMMAND SPAWNED, COMMAND PROCESSING FAILED, or COMMAND EXECUTION FAILED. If the command fails, then the Client can examine the subsequent Command Status data element to determine the cause of the failure. The Status Tag property is the descriptive string corresponding to the numeric value of the Status property. The complete list of Status and Status Tag values can be found in Table 5 Command Status Values and Tags.

Processing Errors are conditions detected by the Command Processor prior to creating a job to execute the command. These are generally syntax related errors. There are two properties defined for a Processing Error. The Processing Error property identifies the specific error that occurred when processing the command. The Processing Error Tag property is the descriptive string corresponding to the numeric value of the Processing Error property. The complete list of Processing Errors can be found in Table 7 Processing Error Values and Tags.

Implementations are NOT REQUIRED to support the Status Tag and Processing Error Tag properties.

3.1.10.1.1 Message Data Elements

The Message element is a text message that describes the disposition of the command. By default, the status message is presented in English text.

The Message data element includes:

- a Message property
- an Owning Entity property
- a Message Id property
- one or more Message Argument properties

When the Message data element is included in a Command Response, the implementation MUST include the Message, Owning Entity, and Message Id properties, and MAY include one or more Message Argument properties. Implementations MUST ensure that the value of the Message Id property together with the value of the Owning Entity property is unique. The Client can use these values to identify corresponding language translations of the Command Status message.

In order to guarantee uniqueness, the Owning Entity property MUST include a prefix string that uniquely identifies the entity that owns and defines the Owning Entity value. The prefix string MUST include a copyrighted, trademarked or otherwise unique name that is owned by the business entity or standards body defining the owning entity string. The implementation MUST NOT return a status message which exceeds 256 characters. Implementations are NOT REQUIRED to support the Message data element.

3.1.10.1.2 Job Data Elements

The Job data element describes the job created to execute the command.

The Job data element includes:

- a Job Identifier property
- a Job Error data element

The Job Identifier is used to identify the MAP job that represents the command execution. The Job Identifier is used to query for Command Status at a later time. The value of the Job Identifier property is the Job Identifier for the job spawned by the MAP to process a command. The implementation MUST include the Job Identifier property whenever it returns a Job data element in a Command Response.

Job Errors are conditions that are detected by the implementation or by the Managed Element target of the command. These errors are detected after a job is created to execute the command. The Job Error properties provide details of the cause of the command failure. The Job Error data element includes:

- an Execution Error property
- an Execution Error Tag property
- one or more Message data elements
- a CIM Status property
- a CIM Status Description property
- a Severity property
- a Severity Description property
- a Probable Cause property
- a Probable Cause Description property
- one or more Recommended Action properties
- an Error Source property
- an Error Source Form property
- an Error Source Form Description property

When the Job Error data element is included in a Command Response, the implementation MUST include the Execution Error, CIM Status, and Severity properties. When the Job Error data element is included in a Command Response, the implementation MAY include Message data elements and MAY include any other properties of the Job Error data element not explicitly required. A complete description of Job Error properties and values is found in 3.2.3.2

The implementation MUST include the Status property in the Command Status data element. Implementations MAY include the Status Tag property and MAY include the Message data element. The implementation MUST NOT include the Processing Error or Processing Error Tag properties in the Command Status data element unless the command Status is COMMAND PROCESSING FAILED. When the command Status is COMMAND PROCESSING FAILED, the implementation MUST include the Processing Error property in the Command Status data element.

When the command Status is COMMAND EXECUTION FAILED, the implementation MUST include the Job data element in the Command Status data element and MUST include the Job Error data element in the Job data element.

When the command Status is COMMAND PROCESSING FAILED, the implementation MUST NOT include the Job data element in the Command Status data element.

When the command Status is COMMAND SPAWNED, the implementation MUST include the Job data element in the Command Status data element and MUST NOT include the Job Error data element in the Job data element.

When the command Status is COMMAND COMPLETED, the implementation MUST include the Job data element in the Command Status data element and MUST NOT include the Job Error data element in the Job data element. When the command Status is Completed, the implementation MUST include the Command Results data element in the Command Response.

3.1.10.2 Command Results Data Elements

The output data elements for Command Results are defined by command verb-specific Command Results schema.

Command Results data elements include

- Command-specific output data elements as defined by the command verb's specification
- Target-specific data elements as defined by the SM Profiles

The specification for each CLP command verb documents the applicable Command Results schema for that command. See Section 4 for command specifications and their corresponding Command Result schema.

In addition, CLP commands return output data that is relevant to the particular Managed Element or elements that were addressed in the command target. Target-specific data elements are returned in the Command Results data.

See CLP-to-CIM Mapping Specification [13] for the keywords and value specifications that are relevant to each class of target ME.

3.1.11 Session Prompt

CLP session output MUST include a session prompt. The CLP defines a specific output format and character string that are used to delineate the session prompt.

CLP session output MUST be of the following form:

<OUTPUT><IMP PROMPT><CLP PROMPT>

where

OUTPUT	is the Command Response, formatted according to one of the CLP-defined output formats
IMP PROMPT	is the implementation's optional prompt string that conforms to the following constraints

- MUST NOT contain the CLP PROMPT string

CLP PROMPT	is the CLP standard prompt string, "`->` ". The CLP prompt string MUST appear after each Command Response.

Implementations MUST include the CLP prompt string, "`->` " (hyphen, greater than, space) after every Command Response returned to the Client. Implementations MUST NOT return any text after the prompt.

Implementations MAY omit an implementation-provided prompt string. If the implementation provides an implementation prompt string, the prompt string MUST be inserted after the CLP Command Response data and before the CLP prompt string.

Implementations MAY vary the implementation prompt string from Command Response to Command Response (i.e. the implementation is not required to maintain a consistent prompt string).

The CLP is a command-response text-based message protocol. An implementation of the CLP Service MUST return a response to each command presented by the Client. Implementations of the CLP Service MUST NOT accept any further commands from the Client until after the implementation returns a Command Response for the currently outstanding command.

Since the CLP is primarily for use by a human user, the CLP Service MUST return a Command Response within a "reasonable amount of time". The CLP Service MUST be capable of spawning any commands that it determines to be long running. When commands are spawned, the CLP Service MUST return an interim Command Response containing the Job Identifier that is to be used by the Client to retrieve the Command Status and results when the command completes.

3.1.12 Extending the command line protocol

The CLP can be extended by a vendor in one of the following ways:

- Supplying vendor-specific command verbs, options, target addresses, or properties
- Adding vendor-specific information to standard CLP command verb output

Vendor extensions are conspicuously named as described in Section 3.2.6 OEM Extensions so that the user is aware that the use of the extension is non-standard. The syntax section of this document defines areas of vendor extensibility and the requirements in effect for those extensions.

3.2 Syntax

The CLP implements a small and easily remembered set of verbs (Section 4) that apply across the entire target address space (Section 3.1.3). This allows a user to quickly understand the function available to them and then apply that knowledge across a wide variety of environments. These verbs provide a consistent set of output (Section 3.1.10.) which further simplifies understanding by both the new and experienced user as they move from implementation to implementation and simple to complex behaviors. As the user becomes more experienced and sophisticated they can further refine the behavior of these verbs using a set of command line options that are also standard across the entire CLP verb space (Section 5).

3.2.1 Basic Command Syntax

3.2.1.1 Character Set, Delimiters, Special and Reserved Characters

All implementations of the CLP MUST interpret the characters provided by the transport as UTF8 representation of the characters, including those in Table 1 and MUST interpret the characters in Table 1 according to the description included in Table 1.

Table 1 CLP Reserved Characters and Character Sequences

Character or Sequence	Name	Description / Uses
" "	space	Command line term separator.
`	escape character	Escape character (the backquote character), use in front of reserved characters to instruct the command parser to use the reserved character without special meaning.
<cr> <lf> <cr><lf>	end-of-line	Each of these sequences are accepted as an end-of-line indicator.
<escape character><end-of-line>	line continuation	An escape character character placed immediately before the end-of-line sequence indicates that the current line is continued (appended) to the following line.
,	comma	Delimits items in an option argument term that is to be interpreted as a list of option arguments.
=	equal sign	Separates a property name from its value and must not have a space before or after it in an expression of a property and its value.
-	hyphen	When preceded by a space, the hyphen is the CLP option indicator.
/ \	address term separator	Separates the UFiT terms of a target address.
=>	association separator	Used to separate an association from the portions of the target address term which identify the instance(s) the association references.
.	dot	Recognized as a special target address term meaning "this container".
..	dot-dot	Recognized as a special target address term meaning "the container of this container".
( )	parentheses	In an option argument term which is a comma separated list, delineates the values of an argument from the next option argument.
"	double quote	Delineates a string of text that may contain the CLP term separator (space) so that the CLP Command Processor will treat the delineated text as one string.
"-> "	CLP PROMPT (hyphen, greater-than, space)	Literal representation of the CLP prompt.
SESSION	SESSION	Reserved target representing the current session.

3.2.1.2 Case Sensitivity

The general CLP command line syntax is not case sensitive. Command verb, option, target and property names may be expressed in any combination of upper and lower case characters.

Implementations MUST accept command verb, option, target, and property names expressed in any combination of upper and lower case characters. For example, “show”, “Show”, and “SHOW” are all valid expressions of the CLP verb “show”. For readability, this specification will document all verb, option, target, and property names in lower case.

Option arguments and property values MAY be case sensitive. Implementations MUST observe and retain all use of case in option arguments and property values.

3.2.1.3 Command Line Terms

A CLP command line consists of four types of terms: verb, options and any option arguments, target address, and target properties. Each term on the command line is separated from other terms by the CLP command term separator character, “ “ (space); See Section 3.2.1.1 for the list of CLP special and reserved characters. Implementations MUST recognize the CLP command term separator character, beginning of line, and end of line as delimiting terms on the command line.

Implementations MUST recognize the end-of-line character as terminating a single command line unless it is preceded by the CLP escape character.

A single command line MAY be continued across end-of-line by using the CLP escape character immediately before the end-of-line character. Command processing MUST be initiated when the complete command has been received by the CLP.

3.2.1.3.1 Verb

The implementation MUST expect the command verb to be the first term in a command. The command verb MUST be one of the specified CLP command verbs listed in Section 4 or an OEM command line extended form as defined in Section 3.2.6.2.2. Implementations MUST NOT support any verbs other than the CLP verbs defined in this specification and any command verbs identified as OEM verbs according to Section 3.2.6.

3.2.1.3.2 Options and Option Arguments

Command options MAY be included immediately after the command verb. Command options are recognized by the option indicator character, “-“ (single hyphen). If options are specified in a command, the options and their arguments MUST occur immediately after the command verb and before the optional target term and target properties.

Command options either require an argument or require no argument.

Options that require no argument MUST be separated from subsequent options, target address, or target property names by the command term separator character. Options that require arguments MUST also be separated from their option argument term by the command term separator character. Implementations MUST interpret a “,” (comma) as delimiting arguments in the option argument term unless the comma is preceeded by a left parenthesis “(“ in which case the implementation MUST ignore any commas which occur prior to a matching right parenthesis “)”. An option argument will be a single argument name or a single argument/value pair. If while parsing a command line an implementation encounters an option it does not recognize, the implementation MUST NOT execute the command and MUST return Command Status of COMMAND PROCESSING FAILED and a Processing Error of INVALID OPTION.

3.2.1.3.3 Target Address

A Target Address is a string that follows the Target Address Syntax and is used to identify the target of a command. The SM CLP supports several types of target addressing. It supports individual instance addressing as defined in the SM Managed Element Addressing Specification [4]. Instance addressing is extended to include support for Relative Target Addresses. It adds support for addressing instances of a particular class as well as support for addressing an instance or instances of an association relative to target instances that the association references.

Implementations MUST require that if the command target address is included in the Command Line, the command target address is the first term that is not an option after the command verb. Implementations MUST require that when the command target term is included in the command, the command target term appears before any target property names.

The implementation MUST observe the following ordered rules for determining if the first non-option or option argument term after the verb should be treated as a target address term:

- If the first non-option or option argument term after the verb contains one of the four CLP reserved addressing terms ("." [dot], ".." dot dot, address term separator, "=>" association separator, or SESSION), the implementation MUST interpret the term as a target address term.
- If the first non-option or option argument term after the verb ends in an integer or an "*" [asterisk], the implementation MUST interpret the term as a target address term.
- When the Command Processor discovers that the first non-option or option argument term after the verb is not a target address term, the Command Processor MUST assume that the term, and any subsequent terms, are property names of the Resultant Target.

These rules enable a Client wishing to ensure consistent results when specifying a target address term to do so through the inclusion of one of the CLP reserved addressing terms.

The order of precedence for determining the Resultant Target of a command is defined in Section 3.1.3.

3.2.1.3.4 Target Address Syntax

Target addresses in the SM CLP may be composed of the following parts:

- UFiT – a UFcT concatenated with an integer suffix. Selects a specific Managed Element in a given container.
- UFsT – a UFcT concatenated with an asterisk. Selects all instances of the type specified by the UFcT within a given container.
- address term separator (/ or \) – When located at the beginning of a target address term, represents the root of the address space. When used to separate two UFiTs in an address, represents an Addressing Association between them.
- association separator (=>) – delimits an Association Class within a target address. The portion of the target address preceding, and optionally following, the association identify one, or both, Managed Elements referenced by the target association instance.

- dot (.) – reserved term meaning “this target”
- dot dot (..) – reserved term meaning “the container of this target”

Each UFiT is a short, text string identifier of a Managed Element in the address space of the MAP. A UFiT is of the form “`UFcT<integer suffix>`” where the first component is a short, text string tag that identifies the class of Managed Element and the second component is an integer suffix that uniquely identifies the Managed Element in its container.

An instance address is a sequence of Managed Element tags, or UFiTs, separated by the slash character. Any UFiT that is followed by a slash character indicates that the remaining target address path is contained within that Managed Element. A UFiP is an Absolute Target Address which references exactly one Managed Element and does not contain any of the CLP addressing extensions (dot, dot dot, or SESSION). The CLP Command Line grammar is formally defined in Section 4.14.

The general syntax of an instance address is as follows (in ABNF form):

```
[ <address term separator>] *[ (  “.” /  “..” / <UFiT>) <address term
separator>] <UFiT>
```

By successively interpreting each term of the target address term and performing any substitutions necessary, it is possible to create a UFiP identifying the Managed Element target of the command.

Some SM CLP commands can be invoked against a UFsT. When the target address path terminates in a UFsT, the Command Processor interprets the command to be targeted to all Managed Elements of that class within that container or uses the class tag as a selector for the action specified by the verb. The general syntax of a target address path of this type is:

```
[<address term separator>] *[ (  “.” /  “..” / <UFiT>) <address term
separator>] <UFsT>
```

As mentioned above, the SM CLP supports addressing an instance or instances of an association. The general syntax for targeting an association is:

```
[[<address term separator>] *[ (  “.” /  “..” / <UFiT>) <address term
separator>] <UFiT>] “=>”<Association Class>[“=>”<address term separator>
[*(<UFiT> <address term separator>) <UFiT>] ]
```

The target address terms leading up to the first occurrence of “=>” are used in accordance with the rules for generating an instance address path. This instance address path identifies one of the Managed Elements referenced by the target association. Following these rules, if no target address terms proceed the first “=>”, the CDT will be selected as the referenced instance. The <Association Class> identifies the association class that will be searched for instances. The 2nd occurrence of “=>” and following additional target address terms are optional. The 2nd “=>” is the ending delimiter of the <Association Class>. The target address terms which follow are used to construct an instance address path identifying the other Managed Element referenced by the association. If the 2nd set of address terms is omitted, the implementation will return all instances of the association class that reference the instance addressed on the lefthand side.

Using the target notation, the following examples target addresses can be constructed:

```
/
/system1
\system1
system1
/system1/alarm3
alarm3
../rack3
hw1/./../../rack3
..
.
/system1=>AssociatedPowerManagementService=>/system1/service
24
../system1/cpu1=>SystemDevice=>/system1
../system1/cpu1=>SystemDevice
```

3.2.1.3.5 Target Address Evaluation

CLP commands fall into two categories: commands that accept a command target term and commands that do not. For commands that accept a target term, the target term may be optional. The rules of precedence which govern choosing between an Implicit Command Target, the CDT, and a command target term are detailed in Section 3.1.3.2. This section describes the rules for evaluating the target term if it is specified.

The target term can be either a Relative Target Address or an Absolute Target Address. The target term will identify a specific Managed Element, a set of Managed Elements identified by their class, a specific association, or the associations of a particular association class which reference a specific Managed Element.

Rules for Addressing a Specific Association

If the target address term includes exactly two occurrences of the character sequence “=>”, the implementation MUST evaluate the target address term according to the following rules:

- The implementation MUST interpret the characters between the two occurrences of “=>” as identifying the Association Class.
- The implementation MUST interpret all characters prior to the first occurrence of “=>” as a single token. The implementation MUST evaluate the token according to the Instance Addressing Rules below.
- If any characters occur after the 2nd occurrence of “=>”, the implementation MUST interpret the characters as a UFiP.

If a UFiP was specified after the 2nd occurrence of “=>”, the implementation MUST set Resultant Target for the command to the association of the type specified by the Association Class such that the association references the Managed Element identified by the instance address preceeding the 1st “=>” and the association references the Managed Element identified by the UFiP following the 2nd occurrence of “=>”.

Rules for Addressing a Set of Associations

If the target address term includes exactly one occurrence of the character sequence “=>”, the target address term is assumed to be targeting all associations of a particular class which reference an instance and the implementation MUST evaluate the target address term according to the following rules:

- The implementation MUST interpret the characters between the two occurrences of “=>” as identifying the Association Class.
- The implementation MUST interpret all characters prior to the first occurrence of “=>” as a single instance address. The implementation MUST evaluate the instance address according to the Instance Addressing Rules below.

The implementation MUST set the Resultant Target to the set of associations such that for each association in the set, the association is of the type specified by the Association Class and references the Managed Element identified by the instance address preceeding the 1st occurrence of “=>” in the target address term.

Rules for Addressing a Target Instance/Class

If the target address term does not include any occurrences of the character sequence “=>”, the implementation MUST evaluate the target address term according to the Rules for Addressing a Target Instance below.

Rules for Addressing a Target Instance

Implementations MUST evaluate instance address terms according to the following rules:

- If the instance address term begins with the address term separator, the instance address term is considered to be an Absolute Target Address. The implementation MUST interpret an Absolute Target Address as relative to the AdminDomain instance which is the root of the MAP’s address space.

- If the instance address term does not begin with the address term separator, the instance address term is considered a Relative Target Address. The implementation MUST interpret a Relative Target Address as relative to the Current Default Target and MUST prepend the instance address term with the UFiP of the Current Default Target and an address term separator prior to evaluating the instance address term.
- Implementations MUST evaluate the instance address term from left to right as follows, using the address term separator as a token delimiter:
  - If the token is a "." [dot], remove the token from the instance address term.
  - If the token is a ".." [dot dot], remove the token from the instance address term and remove the preceeding UFiT, if a preceeding UFiT is present.
  - If the token is a UFiT, leave it in the instance address term.
  - If the token is a UFsT, leave it in the instance address term.

After evaluating the instance address term using the above rules, the instance address term will be a UFiP and is the Resultant Address produced by applying the Instance Addressing Rules.

3.2.1.3.6 Target Properties

Many CLP verbs accept target property terms as input to the command. Target property terms consist of a target property name and a target property value(s). Implementations MUST interpret terms appearing in the Command Line after the target address term as target property terms. Implementations MUST interpret target property names in a case insensitive manner.

When the target address is omitted (implied), any non-option name terms MUST be target property terms.

The implementation MAY process the Properties in any order. If the implementation indicates that the job has completed successfully, the Client is assured that all of the Properties have been applied successfully. However, in the case of an error, it is the responsibility of the Client to query which properties, if any, were applied successfully. Applying properties one command at a time is a deterministic way to determine which properties get applied for any given command. When a structured output is specified, the implementation MUST return string values for each property name and property value such that the implementation will accept the property name and property value as input when they are specified according to the rules in Section 3.2.1.3.7.

3.2.1.3.7 Rules for Specifying Target Property Values

Some CLP verbs accept target properties and values. When a user specifies a target property and value on the command line, the implementation MUST enforce the following syntax:

1. target property name and value are specified: [property name]=[property value]
2. If the property value contains a CLP reserved character, the value must be enclosed in quotes. If the property value includes a " [double quote] character, the " [double quote] must be escaped using the CLP escape character
3. If the property is multi-valued (an array), individual values are delimited with a single , [comma] character,

The specific format of the value for a property is defined in the CLP-to-CIM Mapping Specification [13].

3.2.1.4 Reserved Values for Option Arguments and Properties

Certain strings are recognized and reserved in order to have consistent meaning across the CLP. These reserved strings are used for option arguments and property value settings. Implementations MUST recognize the CLP reserved strings and observe the specified behavior identified by that string in a manner consistent with its definition. E.g. Use of the reserved string “default” MUST always have the same interpretation across all CLP verbs, options, and properties. The table below describes each reserved string, its common interpretation, and its acceptable range of values, if applicable.

Table 2 CLP Reserved Strings

Reserved String	Interpretation
`all`	References the entire set of values for a particular target, option, or property.
`associations`	References the entire set of associations of a command target.
`verbs`	References the set of valid verbs for the target.
`default`	Use the default value for this option or property.
`properties`	References the set or a specified subset of properties defined for the target.
`targets`	References the UFiTs in the scope of a Managed Element container.

3.2.2 Output Data Schema

This section describes valid data elements and associated values for inclusion in a Command Response.

3.2.2.1 Command Response Organization

In the absence of communication errors or session termination, every CLP command will result a Command Response being returned to the Client. A Command Response consists of Command Status data and Command Results data. The Command Response data is ordered as follows:

– Command Status data (e.g. successful or errors/exceptions)

– Command Results data (e.g. information generated from/by the command)

See Section 3.1.10 for full descriptions of the data elements. CLP command options described in Section 5 Standard Command Options control the content and format of output data.

3.2.2.2 Common Output Keywords

Common output keywords are those data elements that MAY appear in any response data. Common keywords are used for data organization purposes—identification, grouping, sorting, etc.

Table 3 Common Output Keywords

Keyword	Definition
`association`	Indicates the data is a target association.

Keyword	Definition
`endgroup`	Indicates the end of a group of output data elements.
`group`	Indicates the beginning of a new group of output data elements that are to be interpreted as a group.
`header`	Indicates the beginning of a group of output data elements that define the format of the “group” data elements that follow. The first keyword after the “header” keyword defines the keyword of the value that appears first in each group, the second keyword defines the keyword of the second value, and so on. “header” is used primarily in the “clpcsv” output format.
`property, properties`	Indicates the data is a target property name.
`target`	Indicates the data is a Managed Element that may contain other Managed Elements.
`targets`	Indicates the data lists targets contained in an element.
`ufct`	Indicates the data is a User-Friendly class Tag.
`ufit`	Indicates the data is a User-Friendly instance Tag.
`ufip`	Indicates the data is a User-Friendly instance Path (fully-qualified path).
`verb, verbs`	Indicates the data is a command verb name.
`endoutput`	Keyword indicating the end of a “clpcsv” or “keyword” Command Response.

3.2.3 Command Status Data Elements

3.2.3.1 Command Status Keywords

A Command Response includes Command Status data elements.

The following table lists the keywords used to identify properties of the Command Status data element.

Table 4 Command Status Keywords

Keyword	Definition
`status`	The Status property. This Is one of the values defined in Table 5.
`status_tag`	The status_tag property. This Is one of the values defined in Table 5.
`error`	The Processing Error property detected by the CLP Service. This is one of the integer values in Table 7.
`error_tag`	The error_tag property. This is the string value in Table 7.

The Command Status indicates the processing disposition of the command entered. When the `status` keyword is returned, implementations MUST assign it one of the values listed in Table 5. When the `status_tag` keyword is returned, implementations MUST assign it the value in Table 5 corresponding to the value of the `status` keyword.

Table 5 Command Status Values and Tags

status	status_tag	Description
0	COMMAND COMPLETED	`Status = Completed`. The command and any associated jobs have completed successfully. The command and any ME jobs completed within command execution. No job remains in-flight and no job ID is active for this

status	status_tag	Description
		command.
1	COMMAND SPAWNED	`Status = Spawned`. The command returned an interim response to the Client but continues to run as a spawned job. The Job ID of the spawned command MAY be used to retrieve the Command Status.
2	COMMAND PROCESSING FAILED	`Status = COMMAND PROCESSING FAILED`. No job was created. No job remains in-flight and no job ID is active for this command.
3	COMMAND EXECUTION FAILED	`Status = COMMAND EXECUTION FAILED`. The command and any associated jobs ran to completion and failed. The command and any ME jobs completed within command execution.

The following table lists the keywords used to identify properties of the Message data element. Each `message_arg` identifies a string value for insertion into the message text. If an implementation supports message argument insertion into message text, the implementation MUST identify each insertion location in the message text using the character sequence `{n}`. Implementations MUST interpret the value of *n* as identifying the index of the message argument to insert.

Table 6 Message Keywords

Keyword	Definition
`message`	`Message data element -` A free-form text explanation of the Command Status or error.
`message_id`	`Message Id data element` - A unique text string identifier for the status or error message that can be used by the Client to locate any translations of the message in other languages.
`message_arg`	`Message Argument data element` - Substitution value for insertion into a message.
`owningentity`	`Owning Entity data element` - A unique string identifier for the owner of the message identifier. The owning entity and message id combine to form a unique key for looking up message text translations.

Table 7 lists the valid values for the `error` and `error_`tag keywords. When an implementation includes the `error` and `error_`tag keywords in a Command Response the implementation MUST assign them values from Table 7.

Table 7 Processing Error Values and Tags

error	error_tag	Description
255	COMMAND ERROR – UNSPECIFIED	Unspecified command error – used only when other command errors below are not applicable.
254	COMMAND NOT SUPPORTED	Command is recognized as a CLP command verb but is not supported by this implementation.
253	COMMAND NOT RECOGNIZED	Command is syntactically correct, but implementation does not recognize the first term in the command as a verb. (i.e. cannot report “not supported” because the verb is unknown to the implementation).
252	COMMAND SYNTAX ERROR	Command is recognized as a CLP command verb, but the syntax has not been correctly followed.
251	INVALID OPTION	Command is recognized as a CLP command verb, the syntax is correct, but an option is not valid.

error	error_tag	Description
250	INVALID ARGUMENT	Command is recognized as a CLP command verb, the syntax is correct, but an argument value for an option is not valid.
249	OUTPUT FORMAT NOT SUPPORTED	User selected an output format that is not supported by this implementation.
248	MISSING ARGUMENT	Comand is recognized as a CLP command verb, the syntax is correct, but an argument value for an option is missing.
247	OPTION NOT SUPPORTED	Command is recognized as a CLP command verb, the syntax is correct, but an option is not supported.
246	INVALID TARGET	The first non-option or option argument term after the verb contained a CLP addressing character but did not adhere to the CLP command target term syntax.
245	REQUIRED OPTION MISSING	The specified command requires an option that was not supplied.
244	QUEUE FULL	A job can not be started to execute the command.

3.2.3.2 Job Error Keywords

When the Command Status is COMMAND EXECUTION FAILED, a Job Error will follow the Command Status to describe the details of the failure. The following table defines the valid keywords and values for the Job data element. The accepted values and corresponding descriptions for each value is provided in the tables that follow. For each keyword, implementations MUST return data that conforms to the restrictions specified for the keyword in Table 8.

Table 8 Job Keywords

Keyword	Definition
`job_id`	`Job Identifier` - The Job Identifier is an integer value in the range [1, 65535] inclusive.
`errtype`	`Execution Error` Provides the primary classification of the error.
`errtype_desc`	Execution Error Tag - The character string tag corresponding to the Execution Error.
`cimstat`	`CIM Status` A value that describes the error as it relates to the CIM Server.
`cimstat_desc`	An enumerated string, corresponding to the value of cimstat.
`severity`	A value that describes the severity of the error from the notifier's point of view.
`severity_desc`	An enumerated string, corresponding to the value of severity.
`probcause`	A value that describes the probable cause of the error.
`probcause_desc`	An enumerated string, corresponding to the value of probcause.
`recmdaction`	A free-form string that describes a recommended action. Zero or more recommended actions may appear per Job Error occurrence.
`errsource`	A string that identifies the Managed Element generating this Job Error instance.
`errsourceform`	A value that identifies the format of the error source string identifier.
`errsourceform_desc`	A free-form string describing and corresponding to the error source format.

The `Execution Error` property communicates the primary category of the Job Error. If an implementation includes the `errtype` keyword in a Command Response, the implementation MUST assign the keyword one of the values from Table 9. If an implementation includes the `errtype_desc` keyword in a Command Response, the implementation MUST assign the keyword the value from Table 9 which corresponds to the value assigned to the `errtype` keyword. The values for `errtype` and `errtype_desc` correspond to the ValueMap and Values for the ErrorType property of CIM_Error.

Table 9 Error Type Values and Descriptions

errtype	errtype_desc	Description
0	Unknown	
1	Other	
2	Communications Error	The command or operation cannot be initiated because the ME is not responding. OR The job is terminated because the target ME is not responsive and the MAP cannot determine the progress of the operation. Note that the state change activity MAY still be in-progress at the ME but the implementation cannot communicate with the ME to determine the status.
3	Quality of Service Error	
4	Software Error	
5	Hardware Error	
6	Environmental Error	
7	Security Error	
8	Oversubscription Error	
9	Unavailable Resource Error	CLP Service cannot acquire needed internal resources to process the command.
10	Unsupported Operation Error	

The CIM Status property communicates any management layer or instrumentation layer errors encountered by the CLP Service in its attempt to initiate the requested operations for the specified targets. Errors that occur when attempting to set the value for a property result in one of the errors listed in Table 10 CIM Status Code Values and Descriptions being returned. If an implementation includes the `cimstat` keyword in a Command Response, the implementation MUST assign the keyword one of the values from Table 10. If an implementation includes the `cimstat_desc` keyword in a Command Response, the implementation MUST assign the `cimstat_desc` keyword the value from Table 10 which corresponds to the value assigned to the `cimstat` keyword. The values for `cimstat` and `cimstat_desc` correspond to the ValueMap and Values for the CIMStatusCode property of CIM_Error.

Table 10 CIM Status Code Values and Descriptions

cimstat	cimstat_desc	Description
1	CIM_ERR_FAILED	A general, unspecified error occurred.
2	CIM_ERR_ACCESS_DENIED	User does not have proper authorization to use command. OR Command was authorized for user, but user is not authorized to perform a resulting operation on this or a dependent target ME. OR User does not have access to a CIM

cimstat	cimstat_desc	Description
		resource.
3	CIM_ERR_INVALID_NAMESPACE	The target namespace does not exist.
4	CIM_ERR_INVALID_PARAMETER	Verb is recognized, command syntax is correct, option names are correct, but an argument value for an option is not valid. One or more target property values or option arguments that specify target properties is invalid.
5	CIM_ERR_INVALID_CLASS	The class indicated by the UFcT or the Association Class does not exist in the scope of the command.
6	CIM_ERR_NOT_FOUND	Command target is not found. The requested UFiT could not be found or was unresponsive.
7	CIM_ERR_NOT_SUPPORTED	Command is valid but target specified does not support the necessary operation or operations needed to carry out the command. OR User has requested an operation that is not supported by this target ME.
8	CIM_ERR_CLASS_HAS_CHILDREN	Operation cannot be carried out on this class since it has subclasses with instances.
9	CIM_ERR_CLASS_HAS_INSTANCES	Operation cannot be carried out on this class since it has instances.
10	CIM_ERR_INVALID_SUPERCLASS	Operation cannot be carried out since superclass does not exist.
11	CIM_ERR_ALREADY_EXISTS	Operation cannot be carried out because the specified UFiT already exists.
12	CIM_ERR_NO_SUCH_PROPERTY	The specified Property does not exist for the command target.
13	CIM_ERR_TYPE_MISMATCH	The value supplied for a property is incompatible with the property's data type.
14	CIM_ERR_QUERY_LANGUAGE_NOT_SUPPORTED	(RESERVED FOR FUTURE USE)
15	CIM_ERR_INVALID_QUERY	(RESERVED FOR FUTURE USE)
16	CIM_ERR_METHOD_NOT_AVAILABLE	The extrinsic Method could not be executed for the command target. OR An operation is currently executing on the target Managed Element or the Managed Element is performing an internal function such that the requested operation cannot be concurrently executed. OR Target ME is in use by another session.
17	CIM_ERR_METHOD_NOT_FOUND	User has requested an operation that is not recognized by the target ME. The specified extrinsic method could not be found for the command target.
18	CIM_ERR_UNEXPECTED_RESPONSE	The returned response from the Managed

cimstat	cimstat_desc	Description
		Element was unexpected. OR Job spawned by previous command has ended prematurely and failed. For example, a firmware update may abort if retrieval of an image from a URI times-out.
19	CIM_ERR_INVALID_RESPONSE_DESTINATION	(RESERVED FOR FUTURE USE)
20	CIM_ERR_NAMESPACE_NOT_EMPTY	(RESERVED FOR FUTURE USE)

The Severity property communicates the urgency of the error, from the MAP's perspective. If an implementation includes the `severity` keyword in a Command Response, the implementation MUST assign the keyword one of the values from Table 11. If an implementation includes the `severity_desc` keyword in a Command Response, the implementation MUST assign the `severity_desc` keyword the value from Table 11 which corresponds to the value assigned to the `severity` keyword. The values for `severity` and `severity_desc` correspond to the ValueMap and Values for the PerceivedSeverity property of CIM_Error. "1" is not specified in the ValueMap and therefore is reserved by the CLP.

Table 11 Severity Values and Descriptions

severity	severity_desc	Description
0	Unknown	The severity is unknown or unassigned by the implementation.
1	Reserved	This value is reserved and should not be used.
2	Low	Used for non-critical issues such as invalid parameters, incorrect usage, unsupported functionality.
3	Medium	Used to indicate action is needed, but the situation is not serious at this time.
4	High	Used to indicate action is needed immediately.
5	Fatal	Used to indicate a loss of data or unrecoverable system or service failure.

The Probable Cause and Probable Cause Description properties identify the probable cause of an execution error. Any errors generated by the Managed Element itself are characterized in the Probable Cause property, described in Table 12. If an implementation includes the `probcause` keyword in a Command Response, the implementation MUST assign the keyword one of the values from Table 12. If an implementation includes the `probcause`_desc keyword in a Command Response, the implementation MUST assign the `probcause`_desc keyword the value from Table 12 which corresponds to the value assigned to the `probcause` keyword. The values for `probcause` and `probcause_desc` correspond to the ValueMap and Values for the ProbableCause property of CIM_Error.

Table 12 Probable Cause Values and Descriptions

probcause	probcause_desc	Description
0	Unknown	
1	Other	
2	Adapter/Card Error	
3	Application Subsystem Failure	

probcause	probcause_desc	Description
4	Bandwidth Reduced	
5	Connection Establishment Error	
6	Communications Protocol Error	
7	Communications Subsystem Failure	
8	Configuration/Customization Error	
9	Congestion	
10	Corrupt Data	
11	CPU Cycles Limit Exceeded	
12	Dataset/Modem Error	
13	Degraded Signal	
14	DTE-DCE Interface Error	
15	Enclosure Door Open	
16	Equipment Malfunction	
17	Excessive Vibration	
18	File Format Error	
19	Fire Detected	
20	Flood Detected	
21	Framing Error	
22	HVAC Problem	
23	Humidity Unacceptable	
24	I/O Device Error	
25	Input Device Error	
26	LAN Error	
27	Non-Toxic Leak Detected	
28	Local Node Transmission Error	
29	Loss of Frame	
30	Loss of Signal	
31	Material Supply Exhausted	
32	Multiplexer Problem	
33	Out of Memory	
34	Output Device Error	
35	Performance Degraded	
36	Power Problem	
37	Pressure Unacceptable	
38	Processor Problem (Internal Machine Error)	
39	Pump Failure	
40	Queue Size Exceeded	
41	Receive Failure	
42	Receiver Failure	
43	Remote Node Transmission Error	

probcause	probcause_desc	Description
44	Resource at or Nearing Capacity	
45	Response Time Excessive	
46	Retransmission Rate Excessive	
47	Software Error	
48	Software Program Abnormally Terminated	
49	Software Program Error (Incorrect Results)	
50	Storage Capacity Problem	
51	Temperature Unacceptable	
52	Threshold Crossed	
53	Timing Problem	
54	Toxic Leak Detected	
55	Transmit Failure	
56	Transmitter Failure	
57	Underlying Resource Unavailable	
58	Version Mismatch	
59	Previous Alert Cleared	
60	Login Attempts Failed	
61	Software Virus Detected	
62	Hardware Security Breached	
63	Denial of Service Detected	
64	Security Credential Mismatch	
65	Unauthorized Access	
66	Alarm Received	
67	Loss of Pointer	
68	Payload Mismatch	
69	Transmission Error	
70	Excessive Error Rate	
71	Trace Problem	
72	Element Unavailable	
73	Element Missing	
74	Loss of Multi Frame	
75	Broadcast Channel Failure	
76	Invalid Message Received	
77	Routing Failure	
78	Backplane Failure	
79	Identifier Duplication	
80	Protection Path Failure	
81	Sync Loss or Mismatch	
82	Terminal Problem	
83	Real Time Clock Failure	

probcause	probcause_desc	Description
84	Antenna Failure	
85	Battery Charging Failure	
86	Disk Failure	
87	Frequency Hopping Failure	
88	Loss of Redundancy	
89	Power Supply Failure	
90	Signal Quality Problem	
91	Battery Discharging	
92	Battery Failure	
93	Commercial Power Problem	
94	Fan Failure	
95	Engine Failure	
96	Sensor Failure	
97	Fuse Failure	
98	Generator Failure	
99	Low Battery	
100	Low Fuel	
101	Low Water	
102	Explosive Gas	
103	High Winds	
104	Ice Buildup	
105	Smoke	
106	Memory Mismatch	
107	Out of CPU Cycles	
108	Software Environment Problem	
109	Software Download Failure	
110	Element Reinitialized	
111	Timeout	
112	Logging Problems	
113	Leak Detected	
114	Protection Mechanism Failure	
115	Protecting Resource Failure	
116	Database Inconsistency	
117	Authentication Failure	
118	Breach of Confidentiality	
119	Cable Tamper	
120	Delayed Information	
121	Duplicate Information	
122	Information Missing	
123	Information Modification	

probcause	probcause_desc	Description
124	Information Out of Sequence	
125	Key Expired	
126	Non-Repudiation Failure	
127	Out of Hours Activity	
128	Out of Service	
129	Procedural Error	
130	Unexpected Information	

3.2.4 Output Data Formats

The CLP specifies the following named, selectable formats for output data: “text”, “clpcsv”, “keyword”, and “clpxml”. These data formats are defined in the following sections.

Implementations MUST support “text” format and MUST provide “text” format output as the default output setting. When other output data formats are supported, implementations MUST allow the user to override the format on a per-command basis using the command option `output`. Implementations MUST also provide an environment setting to control output format for all commands, unless overridden by the user. If an implementation supports at least one output format other than “text”, the implementation MUST support the “clpxml” output format.

3.2.4.1 Text Format

The output format “text” is the default format for output. Output in “text” format is not recommended to be parsed by an automated agent. This format is suitable only to be read by a person. Text output format will vary from implementation to implementation.

When “text” output format is selected, implementations MAY use any output text wording that is deemed appropriate to convey the Command Response data elements to the user. When the Command Response is presented in “text” format, the implementation MAY provide execution status data as part of the text description of the Command Results or, if the command is successful, the implementation MAY NOT include an explicit statement of execution status in the Command Response.

3.2.4.2 Structured Outputs

The CLP specification defines three structured output formats. These are “clpcsv”, “keyword”, and “clpxml”. When returning a Command Response formatted according to a structured output, the implementation MUST use the specific keywords and values identified in Section 3.2.3.1 and Section 3.2.3.2 for each data element included in the Command Status data element.

For information on the rules governing the use of the `output` option to select an output format, see Section 5.10.

3.2.4.2.1 Comma-Separated Value Format

The output format “clpcsv” requests the command to format the output in a comma-separated value format.

In “clpcsv” output format, each line of the output data contains a list of string items separated by the comma character (“,”). When returning a string that contains a comma, the implementation

MUST use the CLP escape character to specify that the comma is to be interpreted as part of the string and not as the “clpcsv” item delimiter.

All “clpcsv” output data is organized as rows in a table. In “clpcsv” output format, there will always be a “clpcsv” table to represent the Command Status. A Command Response MAY contain more than one “clpcsv” table.

Each line of the “clpcsv” output data has as its first item either the “header” or the “group” keyword. Rows beginning with the “header” keyword specify the start of a new table and the items in the comma-separated list of keywords identify the output data elements that appear in each row of the table. Rows beginning with the “group” keyword specify a row of table values for the preceding header. Implementations MUST indicate the end of the Command Results using the string “header,endoutput” on a line by itself. Implementations MAY include blank lines or lines that have an (octothorp) # character in the first character position in the output. If an implementation includes blank lines or lines that have an (octothorp) # character in the first character position in the output, the implementation MUST NOT impart a meaning to the blank lines.

The general form of the “clpcsv” output format is as follows:

```
header,status,status_tag
group,status,status tag
header,error,error_tag
group,error,error tag
header,owningentity,message_id,message,message_arg,...
group,owner identifier,message identifier,message
text,message arg 1,...
          .
          .
          .
group,owner identifier,message identifier,message
text,message arg 1,...

header,job_id
group,job id

header,errtype,errtype_desc,cimstat,cimstat_desc,severity,s
everity_desc,probcause,probcause_desc
group,error type,error descr,cimstatus
value,cimstat_desc,severity,severity desc,probable
cause,probable cause description

header,errsource,errsourceform,errsourceform_desc
group,Target Address of error source,SMA Target
Address,description

header,owningentity,message_id,message,message_arg,...
group,owner identifier,message identifier,message
text,message arg 1,...
```

```
group,message identifier,message text,message arg 1,...

header,recmdaction
group,recommended action 1
group,recommended action 2
     .
     .
     .
group,recommended action n

header,command
group,command verb name
     .
     .
     .
Verb and target-specific keywords
          .
          .
          .
header,<keyword1>,<keyword2>,…,<keywordn>
group,<value_for_keyword1>,<value_for_keyword2>,…,<value_for_keywordn>
group,<value_for_keyword1>,<value_for_keyword2>,…,<value_for_keywordn>
 :
group,<value_for_keyword1>,<value_for_keyword2>,…,<value_for_keywordn>
[next table—optional]

header,endoutput
```

If an implementation includes OEM output for a command issued with a CLP verb, the implementation MUST return the OEM output after the standard output for the command and before the final "endoutput" keyword. If an implementation includes OEM defined keywords for inclusion in the output, the implementation MUST define each keyword using the convention for OEM name extensions defined in Section 3.2.6.1.

If an implementation is returning "clpcsv" output for an OEM Command Form command, the implementation MUST return Command Status using the standard keyword structure and MUST substitute the "command" keyword and subsequent output with the keyword "oemcommand" followed by the vendor defined output.

3.2.4.2.2 Keyword=Value Format

The output format "keyword" requests the command to format the output in a "keyword=value" format. To select "keyword" format explicitly, the implementations MUST accept "keyword" as the argument value for the `format` argument to the `output` option.

In "keyword" output format, output data element items appear in sequence as "<keyword>=<value>"items separated by the end-of-line character sequence. Any value that contains the end-of-line sequence MUST use double quotes around the value.

Implementations MUST specify a group of “keyword” items that are to be interpreted as a single item or collection using the “begingroup” keyword and a value identifying the type of data. Implementations MUST terminate a group using the “endgroup” keyword. Once a “begingroup” keyword appears in the output, all keywords that follow should be interpreted as part of a group until the next “endgroup”. Implementations MUST indicate the end of the Command Response using the “endoutput” keyword. Implementations MAY include blank lines or lines that have an (octothorp) # character in the first character position in the output. If an implementation includes blank lines or lines that have an (octothorp) # character in the first character position in the output, the implementation MUST NOT impart a meaning to the blank lines.

The general form of the “keyword” output format is as follows:

```
commandline=the commandline that was processed
status=Integer job status code
status_tag=String job status
error=integer processing error code, if there is one
error_tag=string description of processing error
begingroup=message
owningentity=organization owning a message
message_id=string identifier for the message, unique with
the value of owningentity
message=message text
message_arg=Insertion value 1 for the message
    .
    .
    .
message_arg=Insertion value n for the message
endgroup
job_id=Identifier for the job created to execute command
errtype=Integer code for the high level category of
execution error
errtype_desc=string description of the execution error
cimstat=integer code for the CIM error
cimstat_desc=string description of the CIM error
severity=integer code indicating the severity of the error
severity_desc=description corresponding to the severity
code
probcause=integer code indicating the probable cause of the
execution error
probcause_desc=description corresponding to the probable
cause code
errsource=Target Address of error source
errsourceform=SMA Target Address
errsourceform_desc=SM Target Address
recmdaction=Free form string describing action to take to
resolve error
begingroup=message
owningentity=organization owning a message
```

```
message_id=string identifier for the message, unique with
the value of owningentity
message=message text
message_arg=Insertion value 1 for the message
    .
    .
    .
message_arg=Insertion value n for the message
endgroup
     .
     .
     .
command=<verbname>
     .
     .
     .
Verb and target-specific keywords
     .
     .
     .
endoutput
```

If an implementation includes OEM output for a command issued with a CLP verb, the implementation MUST return the OEM output after the standard output for the command and before the final “endoutput” keyword. If an implementation includes OEM defined keywords for inclusion in the output, the implementation MUST define each keyword using the convention for OEM name extensions defined in Section 3.2.6.1.

If an implementation is returning “keyword” output for an OEM Command Form command, the implementation MUST return Command Status using the standard keyword structure and MUST substitute the “command” keyword and subsequent output with the keyword “oemcommand” followed by the vendor defined output.

3.2.4.2.3 XML Format

The output format “clpxml” requests the command to format the output in an XML document format. To select “clpxml” format explicitly, implementations MUST accept “clpxml” as the value for the `format` argument to the `option` option.

In “clpxml” output format, the output data is a well-formed XML document. The XML document schema (tags, etc.) is defined per-command. An outline of the XML document specific to each CLP verb is located in the Output section for that verb in Section 4.

The XML schema defining the Command Response data element is defined using XSD in 6.3.2Appendix B. The XML schema is intended to address the requirements of users of the CLP for a simple, parsable schema to represent CLP output. It is not intended as a data exchange format to fully represent a CIM instance or class. If an implementation returns Command Response data as an XML document, the implementation MUST ensure the document default namespace is www.dmtf.org/smash/2005/01/clpxml. OEM vendors MAY extend the Command Response schema. If OEM vendors extend the Command Response schema, the implementation MUST place the OEM extensions into a distinct namespace and MUST define a namespace prefix which follows the convention for OEM name extensions defined in Section 3.2.6.1.

3.2.5 Internationalization / Localization

This section outlines the support for internationalization and localization provided for by the CLP specification. For the purposes of the CLP specification, internationalization is interpreted to mean the substitution of strings in one language for strings having equivalent meaning in another language. Localization refers to the formatting of information for conformance with the norms of a particular locality.

3.2.5.1 Command Input

CLP implementations MUST NOT provide support for internationalization of command line terms. CLP implementations MAY provide support for localization of input data. Furthermore, a CLP implementation MUST NOT support alternative strings for CLP command verbs, option names, reserved strings, and target property names except as OEM Extensions (See Section 3.2.6 OEM Extensions) . Commands written using alternative strings for these command line terms will not be portable from implementation to implementation.

3.2.5.2 Command Output

3.2.5.2.1 CLP Service-side Localization

CLP implementations MAY support localized CLP command output. If the implementation supports localized output, the implementation MUST support the session setting "`language`" and follow the described use of the setting as given in Section 5.10.

3.2.5.2.2 Client-side Localization

CLP implementations MAY support localization of CLP command output by the Client. To support localization of output data at the Client, a CLP implementation MUST support the following capabilities:

- At least one of the structured output modes (See Section 3.2.4.2)
- Capability to report a Message Owner and Message Identifier for each translatable message when a structured output mode is selected (See Section 3.2.2)

3.2.5.3 Locale

The CLP does not specify a mechanism for setting a locale in the environment in order to perform translations of units of data. Implementations are expected to manage establishment of data units (measures, date/time, etc.) via Managed Element settings.

Implementations MUST include the appropriate units designation in "text" and structured output formats for each Managed Element property returned. This provides the Client the information needed to perform any translation of units locally.

OEMs MAY provide extensions to the standard unit designations per the OEM extensions described in the next section.

3.2.6 OEM Extensions

The CLP allows an OEM to add support for vendor-unique commands and output data.

A vendor MAY extend the CLP in the following ways:

- o By providing OEM commands conforming to one of the specified CLP Extended Forms, and/or
- o By providing OEM output keywords

3.2.6.1 OEM Extension Name Strings

OEM Extension Name Strings used for CLP command line terms MUST be identified by the CLP standard prefix “OEM” followed by a vendor-unique identification string so that they will exist in a namespace separate from those that are specified by this document. Conversely, implementations MUST NOT support any other command verbs other than those specified in this specification or those identified as vendor-specific using an OEM Extension Name String as documented here.

The OEM string portion of the prefix MUST uniquely identify the entity that owns and defines the command. The string MUST include a copyrighted, trademarked or otherwise unique name that is owned by the business entity or standards body defining the command.

The standard portion of the string, “OEM”, and the vendor identifier string MUST be case insensitive (“`OEM`”==“`oem`”==“`Oem`” and “`VENDOR`”==“`vendor`”==“`Vendor`”), where the term “vendor” is not taken to be a literal and instead is a value such as “Acme”.

3.2.6.2 Command Extension Forms

The CLP recognizes two forms of command extension:

- o CLP Verb Extended Form
- o OEM Command Line Extended Form

3.2.6.2.1 CLP Verb Extended Form

The CLP Verb Extended Form, or CLP Verb Form, REQUIRES that a standard CLP command verb appear as the first term on the command line. A vendor MAY define vendor-specific command line terms for options, option arguments, target addresses, and/or target properties, as long as those terms follow the semantics defined in the CLP specification.

Terms

- o CLP Verb / Options and /or Option Arguments
- o OEM Options and/or Option Argument(s)
- o OEM Target Addresses and/or Property Names

Syntax

```
<CLP verb> *[”-”<CLP option> [OEM<vendor><arg name string>]]      `
    *[-OEM<vendor><optionname> [OEM<vendor><arg name string>]] `
    OEM<vendor><target address string>                         `
```

```
      *(OEM<vendor><property name string>)
```

Rules

- o The CLP Specification defines the behavior of the verb and associated CLP options when the option is specified with a CLP defined argument.
- o OEM Arguments to SM CLP options and property names and values MUST observe CLP syntax and delimiter rules.
- o The implementation MUST NOT support OEM option arguments that are inconsistent with the behavior of the CLP option. The behavior of the OEM defined argument is vendor specific. For example, an OEM argument to the `display` option can not be used to modify the targets of a command.
- o Implementations MUST NOT accept a short form for an OEM Option.
- o Implementations of OEM targets/properties MUST observe / adhere to the specified CLP verb/option behaviors
- o When defining OEM target name addresses, implementations MUST observe the CLP command delimiter characters but are NOT REQUIRED to follow CLP target naming addressing syntax or semantics.

The behavior of OEM defined options is outside the scope of this specification. Vendors are free to define the format of arguments to OEM options as their needs dictate. This specification places no restrictions on whether options are separated from their arguments by a delimiter, whether options conditionally accept arguments, etc. Therefore when an OEM defined option is included in a command, it may be necessary to have a priori knowledge of the option in order to deterministically parse the command line.

3.2.6.2.2 OEM Command Line Extended Form

OEM Command Line Extended Form, or OEM Command Form, allows a vendor to provide access to vendor-specific commands and command formats. OEM Command Form is indicated by an OEM Extension Name String as the first term on the command line. This term signals a fully OEM-defined command format. Other than this requirement, the commands specified in OEM commands space, arguments, etc are suggested to remain in line with those presented in the CLP Specification, but are not controlled or defined in any way by this document**.**

Terms

- o Full OEM-specified command format
- o Includes form where an OEM extension appears in every CLP command line term position

Syntax

```
   OEM<vendor> <vendor-specified command line syntax>
   OEM<vendor><verb> <vendor-specified command line syntax>
```

(Note the CLP term separator after the first term)

Rules

The vendor completely defines the command syntax, behavior, target addressing, etc. that appears after the first term, where the first term is prefixed by "OEM".

3.2.6.3 Output Extensions

3.2.6.3.1 Vendor-specific Keywords

A vendor MAY supply additional output data elements in the response to any CLP command. All vendor-supplied keyword names MUST be prefixed with the string "OEM" followed by a string name uniquely identifying the vendor. For example, if vendor "ZYX" introduced an output data element keyword "foobar", the resulting keyword would be "OEMZYXfoobar".

3.2.6.3.2 Vendor-specific Messages and Message Files

Vendors MAY define and identify vendor-specific messages using the standard SM CLP message keywords `message_id`, `message_arg`, and `owningentity` as defined in the section "Output Data Schema".
While the keywords and schema for command output are defined by SM CLP, the format of any message files local to the Client is outside the scope of this specification.

4 SM CLP Verbs

This section gives a listing of all the verbs supported by the CLP, requirements for support by implementations, and a short description of their basic functionality. The subsections will further define the behavior of each verb.

The documentation for each verb includes a statement of the command line syntax, a description of applicable targets of the verb, and a definition of the command output content, including output keywords to be included in structured forms of output. Where the effect of a supported option is unique to the verb, it will also be explicitly described. For a complete list of options, including which verbs they are supported with, see Section 5.

Examples are for information only. When an example contradicts with specification text elsewhere in the document, the specification text should be considered the authority. Examples are shown using `Courier New` font. Each example consists of a description of the example in *italics*, the CLP command line emphasized using **bold** text, and the Command Response in flat text. General rules and requirements for the Command Response are specified in Section 3.1.10 Output Data. When examples do not include the `output` option with a `format` argument, it is assumed that session default format is that of the example.

Requirements in the table below are interpreted as follows:

MUST	Implementations MUST support the verb for all target types.
PROFILE	Implementations MUST support the verb when the target is of a particular Profile and the verb is applicable to a target in that Profile. Specific by-target requirements are found in the CLP-to-CIM Mapping Specification [13].

The SM CLP syntax and semantics can be comprehended by a human user without intimate knowledge of the CIM Schema. The command verbs, options, targets and properties are described below in a traditional "man page" format, complete with command execution status and output data element descriptions. However, in order to implement a CLP Service, a developer will need to know the mapping of CLP verb option/target/property combinations to the CIM Schema. This mapping information is not included here, but is collected in a separate specification as noted above.

Table 13 Verb Support Requirements

Command	Requirement	Definition and usage
`cd`	MUST	Used to set the Current Default Target (navigate the target address space of the MAP).
`create`	PROFILE	Used to create new instances and associations in the address space of the MAP This is only allowed for specific target object types as defined by the profiles and specific MAP implementation.
`delete`	PROFILE	Used to destroy instances in the address space of the MAP. This is only allowed for specific target object types as defined by the profiles and specific MAP implementation.
`dump`	PROFILE	Move a binary image from the MAP to a URI.
`exit`	MUST	Used to terminate a CLP session.
`help`	MUST	Used to get context sensitive help. The functionality is the same as the –`help` option with the addition of help for targets.

Command	Requirement	Definition and usage
`load`	PROFILE	Move a binary image to the MAP from a URI.
`reset`	PROFILE	Used to cause a target with power/process control to cycle states from enabled to disabled and back to enabled.
`set`	MUST	Used to set a property or set of properties to a specific value.
`show`	MUST	Used to show values of a property or contents of a collection/target.
`start`	PROFILE	Used to cause a target with power/process control to change states to a higher run level.
`stop`	PROFILE	Used to cause a target with power/process control to change states to a lower run level.
`version`	MUST	Used to query the version of the CLP implementation (by default) and other CLP elements (when specified).

4.1 cd

The general form of the `cd` command is:

`cd [<`*`options`*`>] [<`*`target`*`>]`

For the `cd` command, implementations MUST support the syntax defined for the `cd-cmd` term in the CLP Grammar defined in Section 4.14.

The `cd` (change default target) command is used to navigate the target address space of the implementation. The command changes the Current Default Target for the session. The new target address path is specified on the command line using the standard CLP target syntax and evaluated using the target address evaluation rules. These are documented in Section 3.2.1.3.4 and Section 3.2.1.3.5, respectively. An implementation MUST support any target address construction that is legal according to those rules. As a result, the command supports both relative and absolute path changes. If a command target term is specified, implementations of the cd command MUST evaluate the command target term to a UFiP, validate that the UFiP references a Managed Element in the address space of the MAP, and assign the CDT property of the Managed Element referenced by SESSION to the UFiP.

Implementations of the `cd` command MUST support usage without an argument. If no arguments appear on the command line, the command MUST NOT change the Current Default Target and MUST return the Current Default Target path as output.

The reserved character sequence ".." (dot-dot) MUST be supported by implementations. If dot-dot is the command target term specified in a `cd` command, the implementation MUST set the Current Default Target to the UFiP of the immediate container as defined by the SM Instance Addressing Specification [4]. Since SM Instance Addressing Specification [4]defines a hierarchical containment graph, the use of ".." is unambiguous (i.e. selects one and only one path within the address space). If no target is specified on the command line, the implementation MUST NOT attempt to validate that the CDT addresses a valid Managed Element.

Since the CLP supports relative addressing using the reserved character sequence "..", it is possible to construct a target address that nominally references a point beyond the beginning of the session's root administration domain. Applying the target address term evaluation rules defined in Section 3.2.1.3.5 Target Address Evaluation will result in the target address term being resolved to address the session's root administration domain target address. Thus an attempt to use the cd command to change the Current Default Target to a point beyond the beginning of address space will result in the CDT being assigned to the session's root administration domain.

4.1.1 Valid targets

Implementations of the `cd` command MUST accept an Absolute or a Relative Target Address for the command target term. The cd command has an Implicit Command Target which is the session to which the SESSION Reserved Target will resolve.

4.1.2 Options

Implementations of the `cd` command MUST support the following options, in addition to those specified in Table 16.

`-default`	When the option 'default' appears on the command line with no other options, the command resets the Current Default Target to the session's default target (the root admin domain of the MAP). This option MUST be supported. If the command includes this option and a target was explicitly specified, the implementation MUST NOT change the CDT and MUST return a Processing Error of COMMAND SYNTAX ERROR.

4.1.3 Output

4.1.3.1 Text Format

The Command Results data MUST include the Current Default Target that is in effect when the command completes.

If the implementation cannot determine the current target address (due to error) the implementation MUST return text indicating the CDT is invalid.

4.1.3.2 Structured Format

The returned data MUST include any status data in the standard format at the top of the response. The `cd` command MUST then return the current address with the keyword "ufip".

4.1.3.2.1 XML Output

The implementation MUST return the `cd` element in the `response` element as defined in the Command Response schema in 6.3.2Appendix B.

```
<cd>
     <ufip> User Friendly instance Path of the CDT </ufip>
</cd>
```

4.1.3.2.2 CSV

Implementations MUST use the following form when returning Command Results for the `cd` command in “clpcsv” format.

```
header,command
group,cd
header,ufip
group, User Friendly instance Path of the CDT
```

4.1.3.2.3 Keyword

Implementations MUST use the following form when returning Command Results for the `cd` command in “keyword” format.

```
command=cd
ufip=User Friendly instance Path of the CDT
endoutput
```

4.1.4 Examples

The following examples assume that the user is on a 2 CPU system and each command starts with the Current Default Target set to /system1/cpu2 .

```
Returns the Current Default Target and does not change it.
No validation of the CDT is performed by the
implementation.
-> cd
/system1/cpu2

Returns the Current Default Target and does not change it.
The CDT is validated and appropriate Command Response data
returned if it is no longer valid.

-> cd .
/system1/cpu2

Moves to the parent (container) of the Current Default
Target (up the tree) one level.
-> cd ..
/system1

Move up the containment tree two levels.
-> cd ../..
/
```

Changes Current Default Target to the (absolute) target /system1/cpu1.

```
-> cd /system1/cpu1
/system1/cpu1
```

Changes Current Default Target to /system1/cpu2/temp1 (assuming a target named with a UFiT of “temp1” exists within the /system1/cpu2).

```
-> cd temp1
/system1/cpu2/temp1
```

Move to the session’s Current Default Target’s parent and then to “cpu1”.

```
-> cd ../cpu1
/system1/cpu1
```

Return the error (the target does not resolve) and then return the current target.

```
-> cd cpu1
No such target found
/system1/cpu2
```

Return the error (the target does not resolve) and then return the current target.

```
-> cd –o format=clpxml /cpu1
```

```
     <?xml version="1.0" encoding="UTF-8"?>
<response xmlns="http://www.dmtf.org/smash/2005/01/clpxml"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xsi:schemaLocation="http://www.dmtf.org/smash/2005/01/clpxm
l smclp_command_response.xsd">
  <command>
    <inputline>cd -o format=clpxml /cpu1</inputline>
  </command>
  <cmdstat>
    <status>3</status>
    <status_tag>COMMAND EXECUTION FAILED</status_tag>
          <job>
            <job_id>243</job_id>
            <joberr>
              <errtype>1</errtype>
              <cimstat>6</cimstat>
<cimstat_desc>CIM_ERR_NOT_FOUND</cimstat_desc>
<severity>2</severity>
```

```
</joberr>
</job>
</cmdstat>
  <cd>
    <ufip>/system1</ufip>
  </cd>
</response>
```

Change CDT to /system1/cpu3.

-> **cd –o format=clpxml /system1/cpu3**

```
     <?xml version="1.0" encoding="UTF-8"?>
<response xmlns="http://www.dmtf.org/smash/2005/01/clpxml"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xsi:schemaLocation="http://www.dmtf.org/smash/2005/01/clpxm
l smclp_command_response.xsd">
     <command>
          <inputline>cd -o format=clpxml
/system1/cpu3</inputline>
     </command>
     <cmdstat>
          <status>0</status>
          <status_tag>COMMAND COMPLETED</status_tag>
     <job>
          <job_id>3</job_id>
     </job>
     </cmdstat>
     <cd>
          <ufip>/system1/cpu3</ufip>
     </cd>
</response>
```

Change CDT to /system1/cpu3.

-> **cd –o format=clpcsv /system1/cpu3**

```
header,commandline
group,cd –o format=clpcsv /system1/cpu3
header,status,status_tag
group,0,COMMAND COMPLETED
header,job_id
group,3
header,command
group,cd
header,ufip
group,/system1/cpu3
header,endoutput
```

Change CDT to /system1/cpu3.

```
 -> cd –o format=keyword /system1/cpu3
commandline=cd –o format=keyword /system1/cpu3
status=0
status_tag=COMMAND COMPLETED
job_id=3
command=cd
ufip=/system1/cpu3
endoutput
```

System3 is currently unresponsive

```
 -> cd –o format=keyword /system3
commandline=cd –o format=keyword /system3
status=3
status_tag=COMMAND EXECUTION FAILED
job_id=3
errtype=2
cimstat=6
severity=2
command=cd
ufip=/system3
endoutput
```

Bad syntax on target term

```
 -> cd –output format=keyword //system3
commandline=cd –output format=keyword //system3
status=2
status_tag=COMMAND PROCESSING FAILED
error=246
error_tag=INVALID TARGET
command=cd
ufip=/system3
endoutput
```

Change CDT to SESSION

```
 -> cd SESSION
/map1/settings1/setting5
```

4.2 create

The general form of the `create` command is:

> ***create [<options>] <target> [<property of new target>=<value>] [<property of new target>=<value>]***

For the `create` command, implementations MUST support the syntax defined for the `create-cmd` term in the CLP Grammar defined in Section 4.14.

The create command is used to create new target objects in the target address space of the implementation. Create is only supported in certain specific target profiles. This command MUST be supported on any implementation capable of creating new target objects (e.g. log records). The exact support will be determined by the profiles supported on that implementation.

When creating a new instance the command target is the object to be created. The class of the object being created is determined by the class tag section of the target address passed to the command. If the target is a specific UFiT, the implementation MUST create an instance with the specific UFiT if possible or return an error if creation is not possible. If the Resultant Target terminates in a UFsT, the path up to and including the penultimate term determines the effective target container where the implementation MUST create an instance of the class identified by the UFcT specified by the UFsT if possible or return an error if it is not possible.

This command does not support usage without command line parameters. Implementations MUST require an explicit command target term. The number of options and properties required will vary depending on the command and the target. Specific per-target required options and properties are documented in the CLP-to-CIM Mapping Specification [13].

4.2.1 Valid targets

This command is supported when it is specified in a target mapping [CLP-to-CIM Mapping Specification [13]] for a profile which the implementation supports. For all targets that do not support the usage of the `create` command, implementations MUST NOT show the `create` command in a command listing as being available. The behavior of this command does change on a per target basis. Implementations of the `create` command MUST accept an Absolute or a Relative Target Address for the command target term.

4.2.2 Options

Implementations of the create command MUST support the following options, in addition to those specified in Table 16. The implementation MUST support other options for the command as specified in the specific target profile.

`-default`	When the option `default` appears on the command line the command will create a target with default data. When specific properties are specified on the command line with this option, the target will be created with default data for all properties not specified. This option MUST be supported, unless it is specifically excluded by a target profile.

4.2.3 Output

4.2.3.1 Text Format

If an object was created, the implementation MUST return information about the created object appropriate to the output modifiers selected by the command. If no object was create, the implementation MUST indicate this result.

4.2.3.2 Structured Format

The returned data MUST include any status data in the standard format at the top of the response. The create command MUST then return a list of the created objects with the keyword "instance". If the implementation can not create any objects, it SHOULD not return any additional data or keywords.

4.2.3.2.1 XML Output

The implementation MUST return the `create` element in the `response` element in the returned XML document as defined in the Command Response schema in 6.3.2Appendix B.

```
<create>
     <instance>
          <ufip>User Friendly instance Path identifying target
</ufip>
          <properties>Properties of the Managed Element.
               <property>A property of Managed Element. Each
property element is defined per the xsd.</property>
               <property>A property of Managed Element. Each
property element is defined per the xsd.</property>
          </properties>
     <instance>
</create>
```

4.2.3.2.2 CSV

Implementations MUST use the following form when returning Command Results for the `create` command in "clpcsv" format.

header,command

group,create

header,instance

header,ufip

group,*UFiP of created instance*

4.2.3.2.3 Keyword

Implementations MUST use the following form when returning Command Results for the `create` command in "keyword" format.

```
command=create
```

```
begingroup=instance
ufip=User Friendly instance Path of created instance
endgroup
endoutput
```

4.2.4 Examples

The following examples try to create a log entry in a log called message log. The rules for creating a log entry can be found in the Message Log profile.

```
A log record is successfully created.

-> create log1/record event="The dummy test ran" `
 probablecause="Running a dummy test" `
 recommendedactions="Don't run dummy test" `
 severity=unknown source=me

Event = The dummy test ran
RecordID = 75
ProbableCause = Running a dummy test
RecommendedActions = Don't run dummy test
Severity = unknown
Source = me

Fail to create a log record due to missing properties.
-> create log1/record*
Create failed—-missing required properties.

Fail to create a log record due to missing properties.
-> create –output format=clpxml log1/record*
<?xml version="1.0" encoding="UTF-8"?>
<response xmlns="http://www.dmtf.org/smash/2005/01/clpxml"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xsi:schemaLocation="http://www.dmtf.org/smash/2005/01/clpxm
l
smclp_command_response.xsd">
     <command>
          <inputline>create -output format=clpxml
log1/record*</inputline>
     </command>
     <cmdstat>
          <status>3</status>
          <status_tag>COMMAND EXECUTION FAILED</status_tag>
          <job>
               <job_id>89</job_id>
               <joberr>
                    <errtype>1</errtype>
```

```
                        <errtype_desc>Unknown</errtype_desc>
                        <cimstat>4</cimstat>

          <cimstat_desc>CIM_ERR_INVALID_PARAMETER</cimstat_desc>
                        <severity>2</severity>
                    </joberr>
               </job>
          </cmdstat>
          <create></create>
     </response>
```

Fail to create a log record due to missing properties.

```
-> create –output format=keyword log1/record*

commandline=create -output format=keyword log1/record*
status=3
status_tag=COMMAND EXECUTION FAILED
job_id=89
errtype=1
errtype_desc=unknown
cimstat=4
cimstat_desc=CIM_ERROR_INVALID_PARAMETER
severity=2
command=create
endoutput
```

Successfully create a new user account on the MAP.

```
-> create –o format=clpxml /map1/user* userid=someuser
password=somepassword
<?xml version="1.0" encoding="UTF-8"?>
<response xmlns="http://www.dmtf.org/smash/2005/01/clpxml"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xsi:schemaLocation="http://www.dmtf.org/smash/2005/01/clpxm
l
smclp_command_response.xsd">
     <command>
          <inputline>create -o format=clpxml /map1/user*
userid=someuser password=somepassword</inputline>
     </command>
     <cmdstat>
          <status>0</status>
          <job>
               <job_id>45</job_id>
          </job>
     </cmdstat>
     <create>
          <instance>
```

```
            <ufit ufct="user" instance="4">user4</ufit>
            <ufip>/map1/user4</ufip>
            <properties>
                 <property>
                      <name>userid</name>
                      <value>
                           <val>someuser</val>
                      </value>
                 </property>
                 <property>
                      <name>password</name>
                      <value>
                           <val></val>
                      </value>
                 </property>
            </properties>
            </instance>
       </create>
  </response>
```

Successfully create a new user account on the MAP.

```
-> create –o format=clpcsv /map1/user* userid=someuser\
     password=somepassword
header,commandline
group,create –o format=clpcsv /map1/user* userid=someuser
     password=somepassword
header,status
group,0
header,job_id
group,45
header,command
group,create
header,instance
header,ufip
group,/map1/user4
header,endoutput
```

Successfully create a new user account on the MAP.

```
-> create –o format=keyword /map1/user* userid=someuser
     password=somepassword
commandline=create –o format=keyword /map1/user
     userid=someuser password=somepassword
status=0
job_id=45
command=create
begingroup=instance
ufip=/map1/user4
endgroup
```

```
    endoutput
```

4.3 delete

The general form of the `delete` command is:

`delete [<`*`options`*`>] <`*`target`*`>`

For the `delete` command, implementations MUST support the syntax defined for the `delete-cmd` term in the CLP Grammar defined in Section 4.14.

The `delete` command is used to remove target. The `delete` is only supported in certain specific target profiles. This command MUST be supported on any implementation capable of deleting target objects (e.g. log records). The exact support on a MAP will be determined by the profiles supported in that implementation.

This command supports usage with and without command line options. If the Resultant Target terminates in a UFsT, the path up to and including the penultimate term determines the target of the command. The implementation MUST delete all instances of the type specified by the UFcT indicated by the UFsT that are immediately contained in the target or return an error.

Even if the target for the `delete` command is such that executing the command will result in deleting the Managed Element indicated by the CDT, the implementation MUST delete all of the referenced targets in the scope, including the target referenced by the CDT. It is possible that the CDT will then refer to a target that has been deleted. The user will discover that the CDT is invalid the next time the Resultant Target for a command is the same as the value of the CDT.

4.3.1 Valid targets

This command is supported when it is specified in a target mapping [CLP-to-CIM Mapping Specification [13]] for a profile which the implementation supports. For all targets that do not support the usage of the `delete` command, implementations MUST NOT show the `delete` command in a command listing as being available. The behavior of this command does not change on a per target basis. Implementations of the `delete` command MUST accept an Absolute or a Relative Target Address for the command target term.

4.3.2 Options

Implementations of the delete command MUST support the following options, in addition to those specified in Table 16. The implementation MUST support other options for the command as specified in the specific target profile.

`-f, -force`	The option `force` allows an object(s) to be deleted, ignoring any policy that might cause the implementation to normally not execute the command. When this option is used the implementation MUST execute this deletion if at all possible, without regard to consequences. This option SHOULD be supported.

4.3.3 Output

4.3.3.1 Text Format

Implementations MUST return Command Result data which includes a list of deleted targets when the `verbose` argument is included with the `output` option and implementations MAY specify the list of deleted targets using range notation. If no targets are deleted, the implementation MUST indicate that no targets were deleted. For example, the implementation could return the string “No targets deleted” or an appropriate translation.

4.3.3.2 Structured Format

The returned data MUST include any status data in the standard format at the top of the response. The delete command MUST then return a list of the deleted objects.

4.3.3.2.1 XML Output

The implementation MUST return the `delete` element in the `response` element as defined in the Command Response schema in 6.3.2Appendix B

```
          <delete>
               <target>
                    <instance>
                         <ufip>
     User Friendly instance Path identifying target.
                         </ufip>
                    </instance>
                    <target>
     Recursive target elements representing Managed Elements
     contained in initial target.
                    </target>
                         .
                         .
                         .
               </target>
               .
               .
               .
          </delete>
```

4.3.3.2.2 CSV

Implementations MUST use the following form when returning Command Results for the `delete` command in “clpcsv” format.

```
header,command

group,create

header,instance

header,ufip

group,UFiP of deleted instance
```

```
     .
     .
     .
group,UFiP of deleted instance
header,endoutput
```

4.3.3.2.3 Keyword

Implementations MUST use the following form when returning Command Results for the `delete` command in "keyword" format.

```
command=delete
ufip= UFiP of deleted instance
     .
     .
     .
ufip= UFiP of deleted instance
endoutput
```

4.3.4 Examples

The following example deletes a single log entry on a MAP that supports log deletion. For additional examples see the target profiles which implement this command. Note that in the following examples, if no output format is specified, the default output format is in effect. For these examples, the default output mode is text.

```
     Delete the specific log record record1 contained in log1.
     -> delete log1/record1
     log1/record1 deleted.

     Delete all instances of record contained in log1. ie, clear
     the event log.
     -> delete –o verbose log1/record*
     record1 deleted.
     record2 deleted.
     record3 deleted.
     record4 deleted.
     record5 deleted.
     5 records successfully deleted.

     Delete all instances of record contained in log1. ie, clear
     the event log.  View all of the results in reverse order.
     -> delete –o verbose,end,order=reverse log1/record*
     record5 deleted.
     record4 deleted.
     record3 deleted.
     record2 deleted.
```

```
record1 deleted.
5 records successfully deleted.
```

Deleting all instances of record contained in log1 is successful.

```
-> delete –o terse log1/record*
5 records successfully deleted.
```

Deleting record3 in log1 in system34 is successful. Requested error only output, so nothing is returned.

```
-> delete –o error log1/record3
```

Deleting record3 in log1 in system34 is successful.

```
-> delete –o error,format=clpxml /system34/log1/record3
<?xml version="1.0" encoding="UTF-8"?>
<response xmlns="http://www.dmtf.org/smash/2005/01/clpxml"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xsi:schemaLocation="http://www.dmtf.org/smash/2005/01/clpxm
l
smclp_command_response.xsd">
     <command>
          <inputline>delete -o error,format=clpxml
/system34/log1/record3</inputline>
     </command>
     <cmdstat>
          <status>0</status>
          <status_tag>COMMAND COMPLETED</status_tag>
          <job>
               <job_id>45</job_id>
          </job>
     </cmdstat>
          <delete>
               <target>
                    <instance>
                         <ufit ufct="record"
instance="1">record3</ufit>
                    <ufip>/system34/log1/record3</ufip>
                    </instance>
               </target>
          </delete>
</response>
```

Deleting record3 in log1 in system34 is successful.

```
-> delete –o format=clpcsv /system34/log1/record3

header,commandline

group,delete –o format=clpcsv /system34/log1/record3
```

```
header,status,status_tag
group,0,COMMAND COMPLETED
header,job_id
group,45
header,command
group,delete
header,ufip
group,/system34/log1/record3
header,endoutput
```

Deleting record3 in log1 in system34 is successful.

```
-> delete –o format=keyword log1/record3

commandline=delete –o format=keyword log1/record3
status=0
status_tag=COMMAND COMPLETED
job_id=45
command=delete
ufip=/system34/log1/record3
endoutput
```

Attempt to delete a record that does not exist

```
-> delete –o format=clpxml log1/record334
<?xml version="1.0" encoding="UTF-8"?>
<response xmlns="http://www.dmtf.org/smash/2005/01/clpxml"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xsi:schemaLocation="http://www.dmtf.org/smash/2005/01/clpxm
l
smclp_command_response.xsd">
     <command>
          <inputline>delete -o format=clpxml
log1/record334</inputline>
     </command>
     <cmdstat>
          <status>3</status>
          <status_tag>COMMAND EXECUTION FAILED</status_tag>
          <job>
               <job_id>5349</job_id>
          <joberr>
               <errtype>1</errtype>
               <errtype_desc>Other</errtype_desc>
               <cimstat>6</cimstat>

     <cimstat_desc>CIM_ERR_NOT_FOUND</cimstat_desc>
               <severity>2</severity>
```

```
                    <severity_desc>Low</severity_desc>
                    <errsource/>
                    <errsourceform_desc/>
               </joberr>
               </job>
          </cmdstat>
          <delete></delete>
     </response>

     Deleting individual records is not supported by this
     implementation
     -> delete –o format=clpxml log3/record334

     <?xml version="1.0" encoding="UTF-8"?>
     <response xmlns="http://www.dmtf.org/smash/2005/01/clpxml"
     xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
     xsi:schemaLocation="http://www.dmtf.org/smash/2005/01/clpxm
     l
     smclp_command_response.xsd">
          <command>
               <inputline>delete -o format=clpxml
     log3/record334</inputline>
          </command>
          <cmdstat>
               <status>3</status>
               <status_tag>COMMAND EXECUTION FAILED</status_tag>
               <job>
                    <job_id>2359</job_id>
               <joberr>
                    <errtype>1</errtype>
                    <errtype_desc>Other</errtype_desc>
                    <cimstat>7</cimstat>
          <cimstat_desc>CIM_ERR_NOT_SUPPORTED</cimstat_desc>
                    <severity>2</severity>
                    <severity_desc>Low</severity_desc>
               </joberr>
               </job>
          </cmdstat>
          <delete></delete>
     </response>

     Deleting individual records is not supported by this
     implementation
     -> delete –o format=clpcsv log3/record334


     header,commandline
     group,delete –o format=clpcsv log3/record334
     header,status,status_tag
```

```
group,3,COMMAND EXECUTION FAILED
header,job_id
group,2359
header,errtype,errtype_desc,cimstat,cimstat_desc,severity,s
everity_desc
group,1,Other,7,CIM_ERR_NOT_SUPPORTED,2,Low
header,command
group,delete
header,endoutput
```

Deleting individual records is not supported by this implementation

-> **delete –o format=keyword log3/record334**

```
commandline=delete –o format=keyword log3/record334
status=3
status_tag=COMMAND EXECUTION FAILED
job_id=2359
errtype=1
errtype_desc=Other
cimstat=7
cimstat_desc=CIM_ERR_NOT_SUPPORTED
severity=2
severity_desc=Low
command=delete
endoutput
```

Delete all of the users in the current target.

-> delete user*

```
user[1-20] successfully deleted.
```

Delete all records in log1 (only four exist)

-> **delete –o format=clpxml log1/record***

```
     <?xml version="1.0" encoding="UTF-8"?>

     <response xmlns="http://www.dmtf.org/smash/2005/01/clpxml"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xsi:schemaLocation="http://www.dmtf.org/smash/2005/01/clpxml
smclp_command_response.xsd">

          <command>

               <inputline>delete -o format=clpxml log1/record*
</inputline>

          </command>

          <cmdstat>

               <status>0</status>
               <status_tag>COMMAND COMPLETED</status_tag>
               <job>
                    <job_id>45</job_id>
               </job>
          </cmdstat>
          <delete>
               <target>
                    <instance>
                    <ufit ufct="record"
instance="1">record1</ufit>
                         <ufip>/system34/log1/record1</ufip>
                    </instance>
               </target>
               <target>
                    <instance>
                      <ufit ufct="record"
instance="2">record2</ufit>
                      <ufip>/system34/log1/record2</ufip>
                    </instance>
               </target>
               <target>
                    <instance>
                    <ufit ufct="record"
instance="3">record3</ufit>
                    <ufip>/system34/log1/record3</ufip>
                    </instance>
               </target>
               <target>
                    <instance>
                         <ufit ufct="record"
instance="4">record4</ufit>
                         <ufip>/system34/log1/record4</ufip>
                    </instance>
               </target>
          </delete>
```

```
</response>

Delete all records in log1 (only four exist)
-> delete –o format=clpcsv log1/record*

header,commandline
group,delete –o format=clpcsv log1/record*
header,status,status_tag
group,0,COMMAND COMPLETED
header,job_id
group,45
header,command
group,delete
header,ufip
group,/system34/log1/record1
group,/system34/log1/record2
group,/system34/log1/record3
group,/system34/log1/record4
header,endoutput

Delete all records in log1 (only four exist)
-> delete –o format=keyword /system34/log1/record*


commandline=delete –o format=keyword /system34/log1/record*
status=0
status_tag=COMMAND COMPLETED
job_id=45
command=delete
ufip=/system34/log1/record1
ufip=/system34/log1/record2
ufip=/system34/log1/record3
ufip=/system34/log1/record4
endoutput
```

4.4 dump

The general form of the `dump` command is:

`dump -destination <`*`URI`*`> [<`*`options`*`>] [<target>]`

For the `dump` command, implementations MUST support the syntax defined for the `dump-cmd` term in the CLP Grammar defined in Section 4.14.

The dump command is used to take a binary image from an ME and send it to a specific location (specified as a URI). This command is only supported on certain specific target profiles. This command MUST be supported on any implementation which manages binary images. The exact support on a MAP will be determined by the profiles supported on that implementation.

If the `destination` option is not supplied by the Client, the implementation MUST NOT execute the command and MUST return Command Status of COMMAND PROCESSING FAILED and a Processing Error of REQUIRED OPTION MISSING.

4.4.1 Valid targets

This command is supported when it is specified in a target mapping [CLP-to-CIM Mapping Specification [13]] for a profile which the implementation supports. For all targets that do not support the usage of the `dump` command, implementations MUST NOT show the `dump` command in a command listing as being available. Implementations of the `dump` command MUST accept an Absolute or a Relative Target Address for the command target term.

4.4.2 Options

The following are valid options for the dump command in addition to those specified in Table 16. The implementation MUST support other options for the command as specified in the specific target profile.

`-destination <`*`URI`*`>`

The option `destination` tells the implementation the target to which it will transfer the binary image. The value specified MUST be a valid URI. This option MUST be supported on all MAP implementations of this command and is required on the command line every time this command is executed

The URI specified MAY contain a scheme which indicates the explicit service and location to be used to capture the dumped data. If the URI specified is relative (i.e. does not indicate a scheme), then the implementation MUST interpret the URI according to the rules specified in Section 3.1.3.2

4.4.3 Output

4.4.3.1 Text Format

Implementations MUST include the source and target addresses in the Command Results data and MUST indicate whether or not the operation was successful. For example, if the command was successful, an implementation could return the following string:

```
<target address> transferred to <URI>
```

If the file is not transferred, the implementation could return the following string:

```
<target address> not transferred
```

4.4.3.2 Structured Format

The returned data MUST include any status data in the standard format at the top of the response. The dump command MUST then return the target address with the keyword "source" and the destination URI with the keyword "destination". If the destination is an address within the MAP address space, the implementation MUST identify the destination using the keyword "ufip". If the destination is a URI, the implementation MUST identify the destination using the keyword "uri". The Client will need to check the Command Status to determine whether or not the transfer was successful.

4.4.3.2.1 XML Output

The implementation MUST return `dump` element in the `response` element as defined in the Command Response schema in 6.3.2Appendix B

```
<dump>
     <source>
          <ufip> Full path of source of dump </ufip>
     </source>
     <destination>
          <ufip> Full path of destination of dump </ufip>
     </destination>
</dump>
```

4.4.3.2.2 CSV

Implementations MUST use the following form when returning Command Results for the `dump` command in "clpcsv" format.

```
header,command
group,dump
header,source
header,ufip
group,User Friendly instance path of the source
header,destination
group,uri
```

group,*Unifom Resource Locator for the destination*
header,endoutput

4.4.3.2.3 Keyword

Implementations MUST use the following form when returning Command Results for the `dump` command in "keyword" format. If the destination is local to the MAP, the `ufip` keyword MUST be used instead of the `uri` keyword.

```
command=dump
begingroup=source
ufip=UFiP of source
endgroup
begingroup=destination
uri=URI of the destination
A. endgroup
endoutput
```

4.4.4 Examples

```
Transfer the binary image of memory1 to an ftp site.
-> dump memory1 –destination `
ftp://myserver.com/pub/fwimage.img
memory1 transferred to ftp://myserver.com/pub/fwimage.img

Transfer fails because userid/password for destination is
invalid
-> dump –destination `
ftp://adminstrator:passw0rd@myserver.com/private/administra
tor/memory.dmp –o format=clpxml memory1

<?xml version="1.0" encoding="UTF-8"?>
<response xmlns="http://www.dmtf.org/smash/2005/01/clpxml"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xsi:schemaLocation="http://www.dmtf.org/smash/2005/01/clpxm
l
smclp_command_response.xsd">
     <command>
          <inputline>dump -destination
ftp://adminstrator:passw0rd@myserver.com/private/administra
tor/memory.dmp -o format=clpxml memory1</inputline>
     </command>
     <cmdstat>
          <status>3</status>
          <status_tag>COMMAND EXECUTION FAILED</status_tag>
          <job>
               <job_id>234324</job_id>
          <joberr>
```

```
                <errtype>1</errtype>
                <errtype_desc>Other</errtype_desc>
                <cimstat>1</cimstat>
                <cimstat_desc>CIM_ERR_FAILED</cimstat_desc>
                <severity>2</severity>
                <probcause>60</probcause>
                <probcause_desc>Login Attempts
Failed</probcause_desc>
          </joberr>
          </job>
     </cmdstat>
     <dump/>
</response>
```

Transfer fails because userid/password for destination is invalid

```
-> dump –destination `
ftp://adminstrator:passw0rd@myserver.com/private/administra
tor/memory.dmp –o format=clpcsv memory1

header,commandline
group,dump -destination
ftp://adminstrator:passw0rd@myserver.com/private/administra
tor/memory.dmp -o format=clpcsv memory1
header,status,status_tag
group,3,COMMAND EXECUTION FAILED
header,job_id
group,234324
header,errtype,errtype_desc,cimstat,cimstat_desc,severity,s
everity_desc,probcause,probcause_desc
group,1,Other,1,CIM_ERR_FAILED,2,60,Login Attempts Failed
header,command
group,dump
header,endoutput
```

Transfer fails because userid/password for destination is invalid.

```
-> dump –destination `
ftp://adminstrator:passw0rd@myserver.com/private/administra
tor/memory.dmp –o format=keyword memory1
commandline=dump -destination
ftp://adminstrator:passw0rd@myserver.com/private/administra
tor/memory.dmp -o format=keyword memory1
status=3
status_tag=COMMAND EXECUTION FAILED
job_id=234324
errtype=1
errtype_desc=Other
```

```
cimstat=1
cimstat_desc=CIM_ERR_FAILED
severity=2
severity_desc=Low
probcause=60
probcause_desc=Login Attempts Failed
command=dump
endoutput
```

Client requests help for the dump command.
-> dump –help –o format=clpxml

```
<?xml version="1.0" encoding="UTF-8"?>
<response xmlns="http://www.dmtf.org/smash/2005/01/clpxml"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xsi:schemaLocation="http://www.dmtf.org/smash/2005/01/clpxm
l
smclp_command_response.xsd">
     <command>
          <inputline>dump -help -o
format=clpxml</inputline>
     </command>
     <cmdstat>
          <status>0</status>
          <job>
               <job_id>26210</job_id>
          </job>
     </cmdstat>
     <dump>
          <help>
               <text>The dump command is used to take a
binary image from an ME and transfer it to another
location.  This destination can be within the MAP and
specifed using a UFiP.  The destination need not be within
the MAP.  If the destination is not within the MAP it can
be specified using a URI.</text>
          </help>
     </dump>
</response>
```

Client requests help for the dump command.
-> dump –help –o format=clpcsv

```
header,commandline
group,dump –help –o format=clpcsv
header,status
group,0
header,job_id
```

```
group,26210
header,command
group,dump
header,help
group,The dump command is used to take a binary image from
an ME and transfer it to another location.  This
destination can be within the MAP and specifed using a
UFiP.  The destination need not be within the MAP.  If the
destination is not within the MAP it can be specified using
a URI.
header,endoutput
```

Client requests help for the dump command.

-> dump -help

```
commandline=dump -help
status=0
job_id=26210
command=dump
help=The dump command is used to take a binary image from
an ME and transfer it to another location.  This
destination can be within the MAP and specifed using a
UFiP.  The destination need not be within the MAP.  If the
destination is not within the MAP it can be specified using
a URI.
endoutput
```

memory1 is successfully transferred.

**-> dump –destination `
ftp://adminstrator:passw0rd@myserver.com/private/administra
tor/memory.dmp –o format=clpxml memory1**

```
<?xml version="1.0" encoding="UTF-8"?>
<response xmlns="http://www.dmtf.org/smash/2005/01/clpxml"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xsi:schemaLocation="http://www.dmtf.org/smash/2005/01/clpxm
l
smclp_command_response.xsd">
     <command>
          <inputline>dump -destination
ftp://adminstrator:passw0rd@myserver.com/private/administra
tor/memory.dmp -o format=clpxml memory1</inputline>
     </command>
     <cmdstat>
          <status>0</status>
          <job>
               <job_id>385</job_id>
          </job>
     </cmdstat>
     <dump>
          <source>
               <ufip>/system1/memory1</ufip>
          </source>
          <destination>
<uri>ftp://adminstrator:passw0rd@myserver.com/private/admin
istrator/memory.dmp</uri>
          </destination>
     </dump>
</response>
```

Command is missing required option.

-> dump –o format=clpxml memory1

```
     <?xml version="1.0" encoding="UTF-8"?>
     <response xmlns="http://www.dmtf.org/smash/2005/01/clpxml"
     xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
     xsi:schemaLocation="http://www.dmtf.org/smash/2005/01/clpxm
     l
     smclp_command_response.xsd">
          <command>
               <inputline>dump -o format=clpxml
     memory1</inputline>
          </command>
          <cmdstat>
               <status>2</status>
               <error>251</error>
               <error_tag>REQUIRED OPTION MISSING</error_tag>
          </cmdstat>
          <dump></dump>
     </response>
```

Job is spawned to transfer memory1

-> dump –destination `
ftp://adminstrator:passw0rd@myserver.com/private/administra
tor/memory.dmp –o format=clpxml memory1

```
     <?xml version="1.0" encoding="UTF-8"?>
     <response xmlns="http://www.dmtf.org/smash/2005/01/clpxml"
     xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
     xsi:schemaLocation="http://www.dmtf.org/smash/2005/01/clpxm
     l
     smclp_command_response.xsd">
          <command>
               <inputline>dump -destination
     ftp://adminstrator:passw0rd@myserver.com/private/administra
     tor/memory.dmp -o format=clpxml memory1</inputline>
          </command>
          <cmdstat>
               <status>1</status>
               <job>
                    <job_id>385</job_id>
               </job>
          </cmdstat>
          <dump>
               <source>
                    <ufip>/system1/memory1</ufip>
               </source>
               <destination>
          <uri>ftp://adminstrator:passw0rd@myserver.com/private/
     administrator/memory.dmp</uri>
               </destination>
          </dump>
     </response>
```

Job is spawned to transfer memory1

-> dump –destination `
ftp://adminstrator:passw0rd@myserver.com/private/administra
tor/memory.dmp –o format=clpcsv memory1

```
header,commandline
group,dump -destination
ftp://adminstrator:passw0rd@myserver.com/private/administra
tor/memory.dmp memory1
header,status
group,1
header,job_id
group,385
header,command
group,dump
header,source
header,ufip
group,/system1/memory1
header,destination
group,uri
group,ftp://adminstrator:passw0rd@myserver.com/private/admi
nistrator/memory.dmp
header,endoutput
```

Job is spawned to transfer memory1

-> dump –destination `
ftp://adminstrator:passw0rd@myserver.com/private/administra
tor/memory.dmp –o format=keyword memory1

```
commandline=dump -destination
ftp://adminstrator:passw0rd@myserver.com/private/administra
tor/memory.dmp memory1
status=1
job_id=385
command=dump
begingroup=source
ufip=/system1/memory1
endgroup
begingroup=destination
uri=ftp://adminstrator:passw0rd@myserver.com/private/admini
strator/memory.dmp
endgroup
endoutput
```

4.5 exit

The general form of the `exit` command is:

`exit [<`*`options`*`>]`

For the `exit` command, implementations MUST support the syntax defined for the `exit-cmd` term in the CLP Grammar defined in Section 4.14.

The `exit` command terminates the user's current CLP session. This command MUST be supported. When this command is received, implementations MUST initiate a graceful shutdown of the underlying transport. Implementations MUST return Command Response data indicating the session terminated normally.

4.5.1 Valid targets

The `exit` command has an Implicit Command Target of the Managed Element representing the CLP session.where the command is issued. If the Command Line includes an target term, the implementation MUST NOT execute the command and MUST return a Command Status of COMMAND PROCESSING FAILED and a Processing Error of COMMAND SYNTAX ERROR in the Command Response data.

4.5.2 Options

Implementations of the exit command MUST support the options specified Table 16.

4.5.3 Output

4.5.3.1 Text Format

The Command Result data MUST Command Status.

4.5.3.2 Structured Format

The returned data MUST include any status data in the standard format at the top of the response.

4.5.3.2.1 XML Output

The implementation MUST return the `exit` element in the `response` element as defined in the Command Response schema in 6.3.2Appendix B.

```
<exit>
</exit>
```

4.5.3.2.2 CSV

Implementations MUST use the following form when returning Command Results for the `exit` command in “clpcsv” format.

```
header,command

group,exit

header,endoutput
```

4.5.3.2.3 Keyword

Implementations MUST use the following form when returning Command Results for the `help` command in “keyword” format.

```
command=exit
endoutput
```

4.5.4 Examples

```
Examine the effect of the exit command.
-> exit –x
If run without the examine option, this command will exit
the current CLP session.

Display help for the help command
-> exit -help
The exit command is used to exit a CLP session.

Exit the current session
-> exit –output format=clpxml
<?xml version="1.0" encoding="UTF-8"?>
<response xmlns="http://www.dmtf.org/smash/2005/01/clpxml"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xsi:schemaLocation="http://www.dmtf.org/smash/2005/01/clpxm
l
smclp_command_response.xsd">
     <command>
          <inputline>exit -output format=clpxml</inputline>
     </command>
```

```
          <cmdstat>
               <status>0</status>
               <job>
                    <job_id>2332</job_id>
               </job>
          </cmdstat>
          <exit></exit>
     </response>
```

4.6 help

The general form of the `help` command is:

`help [<options>] [<`*`target`*`>] [<properties>]`

`help [<options>] [<`*`command`*`> [<`*`target`*`>]] [<properties>]`

For the `help` command, implementations MUST support the syntax defined for the `help-cmd` term in the CLP Grammar defined in Section 4.14.

The `help` command is used to display help to a user about the usage of a specific command or provide more details on a target. The `help` command is an exception to the rule that command targets must always be instances of Managed Elements. Implementations of the `help` command will also accept CLP command names and OEM command names as targets. Implementations MUSTsupport this command for all targets and verbs. The text that is returned by this command can be defined by an OEM as required for their specific market. If the command is followed by a target and zero or more property names, the implementation MUST return help appropriate for the specified target and MAY return help specific to the properties specified. If the `help` command term is followed by a verb (with or without a explicit target), the implementation MUST return help appropriate for the verb and if available, specific uses of that verb for the current target.

When `help` is issued with no target specified, the implementation MUST return the default `help` output, which MUST include instructions on how to use the `help` command. The default help output MUST include instructions on how to query the list of verbs supported for a target.

4.6.1 Valid targets

Implementations MUST support the `help` command for all the targets in an MAP implementation.

4.6.2 Options

Implementations of the `help` command MUST support the following options, in addition to those specified in Table 16. These options MUST be supported by MAP implementations, but MAY have no effect (return the same data as if they were not used) on some implementations.

`-output verbose`	Implementation SHOULD return extensive help text.

`-output terse`	Implementation SHOULD return a short form of help text.

4.6.3 Output

4.6.3.1 Text Format

The Command Result data MUST include an OEM-defined set of text describing help for the target and/or verb specified on the command line.

4.6.3.2 Structured Format

The returned data MUST include any status data in the standard format at the top of the response. The help command MUST then return an OEM defined set of text describing the help for the target and/or command specified on the command line. The keyword “helptext” MUST be used when returning this text.

4.6.3.2.1 XML Output

The implementation MUST return the `help` element in the `response` element as defined in the Command Response schema in 6.3.2Appendix B.

```
<help>
     <text>
               :
          Free-form text
               :
     </text>
</help>
```

4.6.3.2.2 CSV

Implementations MUST use the following form when returning Command Results for the `help` command in “clpcsv” format.

```
header,command
group,help
header,help
group,The help text.
header,endoutput
```

4.6.3.2.3 Keyword

Implementations MUST use the following form when returning Command Results for the `help` command in “keyword” format.

```
command=help
help=The help text.
endoutput
```

4.6.4 Examples

```
     Display help for the log target.
```

```
-> help log1
log1 is a message log and has records in it.
```

Display help for the help command

```
-> help help
The Help command is used to view useful information about
targets and verbs.
```

Display help for the dump command

```
-> help –o format=clpxml dump

<?xml version="1.0" encoding="UTF-8"?>
<response xmlns="http://www.dmtf.org/smash/2005/01/clpxml"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xsi:schemaLocation="http://www.dmtf.org/smash/2005/01/clpxm
l
smclp_command_response.xsd">
     <command>
          <inputline>help -o format=clpxml dump</inputline>
     </command>
     <cmdstat>
          <status>0</status>
          <job>
               <job_id>989</job_id>
          </job>
     </cmdstat>
     <help>
          <text>The dump command is used to take a binary
image from an ME and transfer it to another location.  This
destination can be within the MAP and specifed using a
UFiP.  The destination need not be within the MAP.  If the
destination is not within the MAP it can be specified using
a URI.</text>
     </help>
</response>
```

Display help for the dump command

```
-> help –o format=clpcsv dump

header,commandline
group,help –o format=clpcsv dump
header,status
group,0
header,job_id
group,989
header,command
group,help
header,help
```

```
group,The dump command is used to take a binary image from
an ME and transfer it to another location.  This
destination can be within the MAP and specifed using a
UFiP.  The destination need not be within the MAP.  If the
destination is not within the MAP it can be specified using
a URI.
header,endoutput
```

Display help for the dump command

-> help –o format=keyword dump

```
commandline=help –o format=keyword dump
status=0
job_id=989
command=help
help=The dump command is used to take a binary image from
an ME and transfer it to another location.  This
destination can be within the MAP and specifed using a
UFiP.  The destination need not be within the MAP.  If the
destination is not within the MAP it can be specified using
a URI.
endoutput
```

Examine option used with the help command.

-> help –x –o format=clpxml /system1

```
<?xml version="1.0" encoding="UTF-8"?>
<response xmlns="http://www.dmtf.org/smash/2005/01/clpxml"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xsi:schemaLocation="http://www.dmtf.org/smash/2005/01/clpxm
l
smclp_command_response.xsd">
     <command>
          <inputline>help -x -o format=clpxml
/system1</inputline>
     </command>
     <cmdstat>
          <status>0</status>
     </cmdstat>
     <help>
          <examine>
               <text>If run without the examine option,
this command will return help about /system1.</text>
          </examine>
     </help>
</response>
```

Examine option used with the help command.

-> help –x –o format=clpcsv /system1

```
header,commandline
group,help -x –o format=clpcsv /system1
header,status
group,0
header,job_id
group,989
header,command
group,help
header,examine
group,If run without the examine option, this command will
return help about /system1
header,endoutput
```

Examine option used with the help command.
-> help –x –o format=keyword /system1

```
commandline=help –x –o format=keyword system1
status=0
job_id=989
command=help
examine=If run without the examine option, this command
will return help about /system1.
endoutput
```

4.7 load

The general form of the `load` command is:

```
    load -source <URI> [<options>] [<target>]
```

For the `load` command, implementations MUST support the syntax defined for the `load-cmd` term in the CLP Grammar defined in Section 4.14.

The `load` command is used to take a binary image from a specific source location (specified as a URI) and place it at the specified target address. The exact behavior of the `load` command is profile and implementation specific. The profile may dictate whether the desired action is a simple file transfer or whether it includes an implicit installation of the transferred image. In the case of an implicit installation, it may be implementation dependent whether additional actions are required to complete the installation process. This command is only supported on certain specific target profiles. The `load` command MUST be supported on any implementation which manages binary images. The exact support on a MAP will be determined by the profiles supported on that implementation.

If the `source` option is not supplied by the Client, the implementation MUST NOT execute the command and MUST return Command Status of COMMAND PROCESSING FAILED and a Processing Error of REQUIRED OPTION MISSING.

4.7.1 Valid targets

This command is supported when it is specified in a target mapping [CLP-to-CIM Mapping Specification [13]] for a profile which the implementation supports. For all targets that do not support the usage of the `load` command, implementations MUST NOT show the `load` in a command listing as being available. Implementations of the `load` command MUST accept an Absolute or a Relative Target Address for the command target term.

4.7.2 Options

Implementations of the load command MUST support the following options, in addition to those specified in Table 16. The implementation MUST support other options for the command as specified in the specific target profile.

`-source <URI>`	This option tells the implementation the target from which it will transfer the binary image. The value specified MUST be a valid URI. This option MUST be supported on all implementations of this command. This option is required on the command line every time this command is executed (unless `–h` is used). The URI specified MAY contain a scheme which indicates the explicit service and location to be used to retrieve the binary image. If the URI specified is relative (i.e. does not indicate a scheme), then the implementation MUST interpret the URI according to the rules specified in Section 3.1.3.2

4.7.3 Output

4.7.3.1 Text Format

The Command Result data MUST include the source URI and the target instance address, and MUST indicate whether or not the command was successful. For example, if the command was successful, an implementation could return:

```
     <URI> transferred to <target address>
```

If the image is not transferred successfully, the implementation could return the following string:

```
     <URI> not transferred
```

4.7.3.2 Structured Format

The returned data MUST include any status data in the standard format at the top of the response. The load command MUST then return the target address with the keyword "destination" and the source URI with the keyword "source". If the image is not transferred the implementation SHOULD only return the URI address (with the "source" keyword).). If the source is an address within the MAP address space, the implementation MUST identify the source using the keyword "ufip". If the destination is a URI, the implementation MUST identify the source using the keyword "uri"

4.7.3.2.1 XML Output

The implementation MUST return the `load` element in the `response` element as defined in the Command Response schema in Section 4.14.

```
<load>
     <source>
          <uri> Full path of source of load </uri>
     </source>
     <destination>
          <ufip> Full path of destination of load </ufip>
     </destination>
</load>
```

4.7.3.2.2 CSV

Implementations MUST use the following form when returning Command Results for the `load` command in "clpcsv" format.

```
header,command
group,load
header,source
header,uri
group,URI of the source of the image to load
header,destination
header,ufip
group,UFiP of the destination for the image
```

4.7.3.2.3 Keyword

Implementations MUST use the following form when returning Command Results for the `load` command in “keyword” format.

```
command=load
begingroup=source
uri= URI of the source of the image to load
endgroup
begingroup=destination
ufip= UFiP of the destination for the image
endgroup
endoutput
```

4.7.4 Examples

```
Firmware image is successfully loaded from ftp site.
-> load –source ftp://myserver.com/pub/fwimage.img `
firmwareimage
ftp://myserver.com/pub/firmwareimage.img transferred to
firmwareimage

Firmware image is successfully loaded from an authenticated
ftp site.
-> load –source `
ftp://adminstrator:passw0rd@myserver.com/private/ `
administrator/firmware.img –o format=clpxml softwareid1

<?xml version="1.0" encoding="UTF-8"?>
<response xmlns="http://www.dmtf.org/smash/2005/01/clpxml"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xsi:schemaLocation="http://www.dmtf.org/smash/2005/01/clpxm
l
smclp_command_response.xsd">
     <command>
          <inputline>load -source
ftp://adminstrator:passw0rd@myserver.com/private/administra
tor/firmware.img -o format=clpxml softwareid1</inputline>
     </command>
     <cmdstat>
          <status>0</status>
          <job>
               <job_id>385</job_id>
          </job>
     </cmdstat>
     <load>
          <source>
     <uri>ftp://adminstrator:passw0rd@myserver.com/private/
administrator/firmware.img</uri>
```

```
          </source>
          <destination>
               <ufip>/system1/softwareid1</ufip>
          </destination>
     </load>
</response>
```

Firmware image is successfully loaded from an authenticated ftp site.

-> load –source ` ftp://adminstrator:passw0rd@myserver.com/private/` administrator/firmware.img –o format=clpcsv softwareid1

```
header,commandline
group,load -source
ftp://adminstrator:passw0rd@myserver.com/private/administra
tor/firmware.img -o format=clpcsv softwareid1
header,status
group,0
header,job_id
group,385
header,command
group,load
header,source
header,uri
group,ftp://adminstrator:passw0rd@myserver.com/private/admi
nistrator/firmware.img
header,destination
header,ufip
group,/system1/softwareid1
endoutput
```

Firmware image is successfully loaded from an authenticated ftp site.

-> load –source ` ftp://adminstrator:passw0rd@myserver.com/private` administrator/firmware.img –o format=keyword softwareid1

```
commandline=load -source
ftp://adminstrator:passw0rd@myserver.com/private/administra
tor/firmware.img -o format=keyword softwareid1
status=0
job_id=385
command=load
begingroup=source
uri=ftp://adminstrator:passw0rd@myserver.com/private/admini
strator/firmware.img
endgroup
```

```
begingroup=destination
ufip=/system1/softwareid1
endgroup
endoutput
```

Firmware image cannot be loaded from an authenticated ftp site due to bad credentials.

-> load –source \`
ftp://adminstrator:passw0rd@myserver.com/private/\`
administrator/firmware.img –o format=clpxml softwareid1

```
<?xml version="1.0" encoding="UTF-8"?>
<response xmlns="http://www.dmtf.org/smash/2005/01/clpxml"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xsi:schemaLocation="http://www.dmtf.org/smash/2005/01/clpxm
l
smclp_command_response.xsd">
     <command>
          <inputline>load -source
ftp://adminstrator:passw0rd@myserver.com/private/administra
tor/firmware.img -o format=clpxml softwareid1</inputline>
     </command>
     <cmdstat>
          <status>2</status>
          <status_tag>COMMAND EXECUTION FAILED</status_tag>
          <job>
               <job_id>234324</job_id>
          <joberr>
               <errtype>1</errtype>
               <errtype_desc>Other</errtype_desc>
               <cimstat>1</cimstat>
               <cimstat_desc>CIM_ERR_FAILED</cimstat_desc>
               <severity>2</severity>
               <probcause>60</probcause>
               <probcause_desc>Login Attempts
Failed</probcause_desc>
          </joberr>
          </job>
     </cmdstat>
     <dump/>
</response>
```

4.8 reset

The general form of the `reset` command is:

`reset [<options>] [<`*`target`*`>]`

For the `reset` command, implementations MUST support the syntax defined for the `reset-cmd` term in the CLP Grammar defined in Section 4.14.

The `reset` command resets the target's state. This behavior can be modified to take the target to a specific state through the use of options.

This command supports usage with and without command line options.

4.8.1 Valid targets

This command is supported when it is specified in a target mapping [CLP-to-CIM Mapping Specification [13]] for a profile which the implementation supports. For all targets that do not support the usage of the `reset` command, implementations MUST NOT show the `reset` command in a command listing as being available. Implementations of the `reset` command MUST accept an Absolute or a Relative Target Address for the command target term.

The behavior of state change commands for each UFcT is defined in the CLP-to-CIM Mapping Specification [13].

4.8.2 Options

The following are valid options for the reset command in addition to those specified in Table 16 Verb and Option Support

`-f, -force`	Forces the implementation to reset the object, ignoring any policy that might cause the implementation to normally not execute the command. The implementation MUST execute this reset if at all possible, without regard to consequences.
`-`**`resetstate`** `<`*`value`*`>`	This option is used to indicate to the implementation what target-specific state to take the target through (i.e. NMI reset, soft, hard, interrupt_only, etc). . The value is a string corresponding to the state requested. Valid values are target specific and are specified in the CLP-to-CIM Mapping Specification [13]. This option SHOULD be supported. If this option is not specified, implementations MUST behave as if it was specified with an argument of `targetdefaultvalue`, where the `targetdefaultvalue` is defined for the class in the CLP-to-CIM Mapping Specification [13].

4.8.3 Output

4.8.3.1 Text Format

Implementations MUST return Command Result data which includes the target address that was reset (if any) and the time and date when the reset started. Implementations are free to return the time and date in any format that meets their needs. If no targets were reset, the implementation MUST indicate this result.

4.8.3.2 Structured Format

Implementations MUST include any status data in the standard format at the top of the response. If the target was successfully reset, the implementation MUST then return the target and the time the reset was initiated.

4.8.3.2.1 XML Output

The implementation MUST return the `reset` element in the `response` element as defined in the Command Response schema in 6.3.2Appendix B.

```
<reset>
     <ufip>target address the command was invoked against
</ufip>
     <timestamp>Time reset occurred if completed synchronously,
returned in CIM datetime format</timestamp>
</reset>
```

4.8.3.2.2 CSV

Implementations MUST use the following form when returning Command Results for the `reset` command in "clpcsv" format.

```
header,command
group,reset
header,ufip,timestamp
group,User Friendly instance path of target of reset job,time
reset occurred if completed synchronous to command
header,endoutput
```

4.8.3.2.3 Keyword

Implementations MUST use the following form when returning Command Results for the `reset` command in "keyword" format.

```
command=reset
ufip=User Friendly instance path of target of reset
timestamp=time reset occurred if completed synchronous to
command
endoutput
```

4.8.4 Examples

```
     Attempt to reset the operating system on system1 succeeds.
     -> reset /system1
     /system1 reset at 10:40am 1/1/01

     Attempt to reset the operating system on system1 fails.
```

```
-> reset –o format=clpxml /system1/os1

<?xml version="1.0" encoding="UTF-8"?>
<response xmlns="http://www.dmtf.org/smash/2005/01/clpxml"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xsi:schemaLocation="http://www.dmtf.org/smash/2005/01/clpxm
l
smclp_command_response.xsd">
     <command>
          <inputline>reset -o format=clpxml
/system1/os1</inputline>
     </command>
     <cmdstat>
          <status>3</status>
          <job>
               <job_id>4824</job_id>
          <joberr>
               <errtype>2</errtype>
               <cimstat>1</cimstat>
               <severity>2</severity>
          </joberr>
          </job>
     </cmdstat>
     <reset>
          <instance>
               <ufit ufct="os" instance="1">os1</ufit>
               <ufip>/system1/os1</ufip>
          </instance>
    </reset>
</response>

Attempt to reset the operating system on system1 fails.
-> reset –o format=clpcsv /system1/os1

header,commandline
group,reset –o format=clpcsv /system1/os1
header,status
group,3
header,job_id
group,4824
header,errtype,cimstat,severity
group,2,1,2
header,command
group,reset
header,ufip
group,/system1/os1
header,endoutput
```

Attempt to reset the operating system on system1 fails

-> reset –o format=keyword /system1/os1

```
commandline=reset –o format=keyword /system1/os1
status=3
job_id=4824
errtype=2
cimstat=1
severity=2
command=reset
ufip=/system1/os1
endoutput
```

Attempt to reset the operating system on system1 is successful

-> reset –w –o format=clpxml /system1/os1

```
<?xml version="1.0" encoding="UTF-8"?>
<response xmlns="http://www.dmtf.org/smash/2005/01/clpxml"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xsi:schemaLocation="http://www.dmtf.org/smash/2005/01/clpxm
l
smclp_command_response.xsd">
     <command>
          <inputline>reset -w -o format=clpxml
/system1/os1</inputline>
     </command>
     <cmdstat>
          <status>0</status>
          <job>
               <job_id>92341</job_id>
          </job>
     </cmdstat>
     <reset>
          <instance>
               <ufit ufct="os" instance="1">os1</ufit>
               <ufip>/system1/os1</ufip>
          </instance>
        <timestamp>20050130145904.000000-300</timestamp>
     </reset>
</response>
```

Attempt to reset the operating system on system1 is successful.

-> reset -w –o format=clpcsv /system1/os1

```
header,commandline
```

```
group,reset –w –o format=clpcsv /system1/os1
header,status
group,0
header,job_id
group,92341
header,command
group,reset
header,ufip
group,/system1/os1,20050130145904.000000-300
header,endoutput
```

Attempt to reset the operating system on system1 is successful.

-> reset -w –o format=keyword /system1/os1

```
commandline=reset -w –o format=keyword /system1/os1
status=0
job_id=92341
command=reset
ufip=/system1/os1
timestamp=20050130145904.000000-300
endoutput
```

4.9 set

The general form of the `set` command is:

`set [<options>] [<`*`target`*`>] <`*`propertyname`*`>=<`*`value`*`>`

For the `set` command, implementations MUST support the syntax defined for the `set-cmd` term in the CLP Grammar defined in Section 4.14.

The `set` command is used to set the value of one or more of a target's properties. The command can accept a target and series of keyword=value pairs which it will try and apply. Implementations MUST support the `set` command.

The implementation MAY allow the user to set multiple property values for a single target. The implementation MAY set the property values in the order of properties given on the command line.

Implementations MUST NOT allow the user to set properties on multiple targets with a single command.

If an error occurs, the implementation MAY continue to attempt to set properties. For any property where the implementation fails to assign the user supplied value the implementation MAY set the property to a default value or the implementation MAY NOT change the value of the property at all. The user should check the command output or the target itself for the value of each property to determine which values were set.

It is possible that changing the value of a property specified by the user will result in the implementation changing the value of another property which was not specified by the user, in which case the implementation SHOULD return both properties and their values in the output.

The `set` command requires command line arguments.

4.9.1 Valid targets

The `set` command is valid for any target/property pair that is not read only. For all targets that do not support the usage of the `set` command, the `set` command MUST NOT show up in a command listing as being available. Implementations of the `set` command MUST accept an Absolute or a Relative Target Address for the command target term.

4.9.2 Options

Implementations of the `set` command MUST support the options specified in Table 16.

4.9.3 Output

4.9.3.1 Text Format

Implementations MUST return Command Result data which includes each of the properties that were specified in the command and their current values. Note that the current value of a property MAY be different from that requested on the command line due to implementation constraints, policies, or vendor rules.

4.9.3.2 Structured Format

The returned data MUST include any status data in the standard format at the top of the response. The set command MUST then return a list of the properties that were set with the property name as the keyword and the value to which it was set as the value

4.9.3.2.1 XML Output

The implementation MUST return the `set` element in the `response` element as defined in the Command Response schema in 6.3.2Appendix B.

```
<set>
     <instance>
          <ufip>
User Friendly instance Path identifying target
          </ufip>
          <properties>
               <property>
A modified property of Managed Element. Each property
element is defined per the xsd.
               </property>
          </properties>
     </instance>
</set>
```

4.9.3.2.2 CSV

Implementations MUST use the following form when returning Command Results for the `set` command in “clpcsv” format.

```
header,command
group,set

header,instance
header,ufip
group,UFiP of the target instance for the set command

header,properties
header,property_name,property_val,property_valstring,...,propert
y_val,property_valstring,property_type,property_description,prop
erty_readonly,property_maxlen,property_maxvalue,property_minvalu
e,units,value,valuemap,...value,valuemap
group,property name, property value, property value string if it
is a valuemap,val/valstring repeat if multi-
valued,type,description,readonly,max length,max. value,minimum
value,units, enumerated value,corresponding valuemap, additional
value/valuemap pairs repeated if enumeration

header,endoutput
```

4.9.3.2.3 Keyword

Implementations MUST use the following form when returning Command Results for the `cd` command in "keyword" format.

```
command=set
begingroup=instance
ufip=UFiP of Managed Element targeted by command
begingroup=property
property_name=Property name
property_val=Property value
property_valstring=String corresponding to property value if
value/valuemap
     .
     Additional values if property is an array
     .
property_val=Property value
property_valstring=String corresponding to property value if
value/valuemap
endgroup
     .
     Additonal property groups
     .
endgroup
endoutput
```

4.9.4 Examples

```
     Sets the system's name to "sam"
     -> set /system1 name=sam
     name=sam


     Sets password to 12345
     -> set /map1/user3 password=12345
     password=12345


     Sucessfully sets a new userid and password
     -> set /map1/user3 userid=joesmith password=passw0rd
     userid=joesmith
     password=passw0rd
```

```
The password fails.  Implementation specific behavior
whether the userid is updated or not.  This implementation
updated the userid. It is also implementation specific
whether the password is echoed to the screen.
-> set /map1/user3 userid=joesmith password=12345

Password 12345 does not meet password rules.
userid=joesmith
password=


The password fails.  Implementation specific behavior
whether the userid is updated or not.  This implementation
did not update the userid. It is also implementation
specific whether the password is echoed to the screen.
-> set /map1/user3 userid=joesmith password=12345


Password 12345 does not meet password rules.
userid=olduserid
password=


Sets name to a string that contains spaces
-> set /system1 name=”Human Resources Server”
name=”Human Resources Server”


Attempt to enable loadbalancing on Ethernet port 2 fails
because teamed NIC is not configured. Notice that other
values already exist for the property that could not be set
and these values are returned.

-> set –o format=clpxml port2
enabledcapabilities=loadbalancing

<?xml version="1.0" encoding="UTF-8"?>
<response xmlns="http://www.dmtf.org/smash/2005/01/clpxml"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xsi:schemaLocation="http://www.dmtf.org/smash/2005/01/clpxm
l

smclp_command_response.xsd">
     <command>
          <inputline>set -o format=clpxml port2
enabledcapabilities=loadbalancing</inputline>
     </command>
     <cmdstat>
          <status>3</status>
          <job>
               <job_id>4758</job_id>
               <joberr>
     <errtype>1</errtype>
     <cimstat>1</cimstat>
     <severity>2</severity>
     <probcause>108</probcause>
     <probcause_desc>Software Environment
Problem</probcause_desc>
     <recmdaction>Configure partner NIC prior to
enabling.</recmdaction>
               </joberr>
          </job>
     </cmdstat>
     <set>
          <instance>
          <ufit ufct="port" instance="2">port2</ufit>
     <ufip>/system78/port2</ufip>
               <properties>
               <property>
     <name>enabledcapabilities</name>
     <multivalue>
     <value>
     <val>3</val><valstring>wakeonlan</valstring>
     </value>
     <value>
     <val>2</val><valstring>alertonlan</valstring>
     </value>
     </multivalue>
                    </property>
               </properties>
          </instance>
</set>
</response>
```

Attempt to enable loadbalancing on Ethernet port 2 fails because teamed NIC is not configured. Notice that other values already exist for the property that could not be set and these values are returned.

-> set –o format=clpcsv port2 enabledcapabilities=loadbalancing

```
header,commandline
group,set –o format=clpcsv port2
enabledcapabilities=loadbalancing
header,status
group,3
header,job_id
```

```
group,4758
header,errtype,cimstat,severity,probcause,probcause_desc
group,1,1,2,108,Software Environment Problem
header,recmdaction
group,Configure partner NIC prior to enabling.
header,command
group,set
header,instance
header,ufip
group,/system78/port2
header,properties
header,property_name,property_val,property_valstring,proper
ty_val,property_valstring
group,enabledcapabilities,3,wakeonlan,2,alertonlan
header,endoutput
```

Attempt to enable loadbalancing on Ethernet port 2 fails because teamed NIC is not configured. Notice that other values already exist for the property that could not be set and these values are returned.

-> set –o format=keyword port2 enabledcapabilities=loadbalancing

```
commandline=set –o format=keyword port2
enabledcapabilities=loadbalancing
status=3
job_id=4758
errtype=1
cimstat=1
severity=2
probcause=108
probcause_desc=Software Environment Problem
recmdaction=Configure partner NIC prior to enabling.
command=set
begingroup=instance
ufip=/system78/port2
begingroup=property
property_name=enabledcapabilities
property_val=3
property_valstring=wakeonlan
property_val=2
property_valstring=alertonlan
endgroup
endgroup
endoutput
```

Attempt to enable loadbalancing on Ethernet port 2 fails because teamed NIC is not configured. Notice that this property currently does not have any values assigned.

-> set –o format=clpxml port2 enabledcapabilities=loadbalancing

```
<?xml version="1.0" encoding="UTF-8"?>
<response xmlns="http://www.dmtf.org/smash/2005/01/clpxml"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xsi:schemaLocation="http://www.dmtf.org/smash/2005/01/clpxm
l

smclp_command_response.xsd">
     <command>
          <inputline>set -o format=clpxml port2
enabledcapabilities=loadbalancing</inputline>
     </command>
     <cmdstat>
          <status>3</status>
          <job><job_id>4578</job_id>
          <joberr>
               <errtype>1</errtype>
               <cimstat>1</cimstat>
               <severity>2</severity>
               <probcause>108</probcause>
               <probcause_desc>Software Environment
Problem</probcause_desc>
               <recmdaction>Configure partner NIC prior to

enabling.</recmdaction>
          <messages>
                    <message>
     <owningentity>OEMxyz</owningentity>
     <messageid>23</messageid>
     <messagetext>NIC Team {2} could not be configured.
NIC {1} must be configured prior to configuring NIC
{3}</messagetext>
    <messagearg>
     <index>1</index>
     <value>1</value>
    </messagearg>
    <messagearg>
     <index>2</index>
     <value>1</value>
    </messagearg>
    <messagearg>
     <index>3</index>
     <value>2</value>
    </messagearg>
                    </message>
                    <message>
     <owningentity>OEMxyz</owningentity>
                    <messageid>1</messageid>
                    </message>
               </messages>
          </joberr>
          </job>
     </cmdstat>
     <set>
          <instance>
          <ufit ufct="port" instance="2">port2</ufit>
     <ufip>/system78/port2</ufip>
               <properties>
                    <property>
                    <name>enabledcapabilities</name>
               <multivalue></multivalue>
     </property>
               </properties>
          </instance>
     </set>
</response>
```

Attempt to enable loadbalancing on Ethernet port 2 fails because teamed NIC is not configured. Notice that this property currently does not have any values assigned.

-> set –o format=clpcsv port2 enabledcapabilities=loadbalancing

```
header,commandline
group,set –o format=clpcsv port2
enabledcapabilities=loadbalancing
header,status
group,3
header,job_id
group,4578
header,errtype,cimstat,cimstat_desc,severity,severity_desc,
probcause,probcause_desc
group,1,1,2,108,Software Environment Problem
header,owningentity,message_id,message,message_arg,message_
arg,message_arg
group,OEMxyz,23,"NIC Team {2} could not be configured, NIC
{1} must be configured prior to configuring NIC {3},1,1,2
group,OEMxyz,001,Please consult product documentation.
header,recmdaction
group,Configure partner NIC prior to enabling.
header,command
```

```
group,set
header,instance
header,ufip
group,/system78/port2
header,properties
header,property_name
group,enabledcapabilities
header,endoutput
```

Attempt to enable loadbalancing on Ethernet port 2 fails because teamed NIC is not configured. Notice that this property currently does not have any values assigned.

-> set –o format=keyword port2 enabledcapabilities=loadbalancing

```
commandline=set –o format=keyword port2
enabledcapabilities=loadbalancing
status=3
job_id=4578
errtype=1
cimstat=1
severity=2
probcause=108
probcause_desc=Software Environment Problem
begingroup=message
owningentity=OEMxyz
message_id=23
message=NIC Team {2} could not be configured, NIC {1} must
be configured prior to configuring NIC {3}
message_arg=1
message_arg=1
message_arg=2
endgroup
begingroup=message
owningentity=OEMxyz
message_id=001
message=Please consult product documentation.message_arg=1
endgroup
recmdaction=Configure partner NIC prior to enabling.
command=set
begingroup
ufip=/system78/port2
begingroup
property_name=enabledcapabilities
endgroup
endoutput
```

Successfully enable failover and WakeOnLAN support on Ethernet port 2. Notice we're assigning multiple values.

-> set –o format=clpxml port2 enabledcapabilities=failover,wakeonlan

```
<?xml version="1.0" encoding="UTF-8"?><response
xmlns="http://www.dmtf.org/smash/2005/01/clpxml"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xsi:schemaLocation="http://www.dmtf.org/smash/2005/01/clpxm
l

smclp_command_response.xsd">
     <command>
          <inputline>set -o format=clpxml /system78/port2
enabledcapabilities=failover,wakeonlan</inputline>
     </command>
     <cmdstat>
          <status>0</status>
          <job>
               <job_id>7623</job_id>
          </job>
     </cmdstat>
     <set>
          <instance>
            <ufit instance="2" ufct="port">port2</ufit>
          <ufip>/system78/port2</ufip>
     <properties>
       <property>
     <name>enabledcapabilities</name>
     <multivalue>
     <value>
     <val>4</val>
     <valstring>failover</valstring>
     </value>
     <value>
     <val>3</val>
     <valstring>wakeonlan</valstring>
     </value>
</multivalue>
                    </property>
               </properties>
          </instance>
     </set>
</response>
```

Successfully enable failover and WakeOnLAN support on Ethernet port 2. Notice we're assigning multiple values.

```
-> set –o format=clpcsv port2
enabledcapabilities=failover,wakeonlan
```

```
header,commandline
group,set /system78/port2
enabledcapabilities=failover,wakeonlan
header,status
group,0
header,job_id
group,7632
header,instance
header,ufip
group,/system78/port2
header,properties
header,property_name,property_val,property_valstring,proper
ty_val,property_valstring
group,enabledcapabilities,4,failover,3,wakeonlan
header,endoutput
```

Successfully enable failover and WakeOnLAN support on Ethernet port 2. Notice we're assigning multiple values.

```
-> set –o format=keyword port2
enabledcapabilities=failover,wakeonlan
```

```
commandline=set /system78/port2
enabledcapabilities=failover,wakeonlan
status=0
job_id=7632
command=set
begingroup=instance
ufip=/system78/port2
begingroup=property
property_name=enabledcapabilities
property_val=4
property_valstring=failover
property_val=3
property_valstring=wakeonlan
endgroup
endgroup
endoutput
```

Set the default output format to xml.

```
-> set SESSION outputformat=clplxml
/map1/sessions1/setting5
     default output format set to XML.
```

Back door method to change the CDT.

```
-> set SESSION cdt=/system5/cpu3
/map1/sessions1/setting5
     Current Default Target is /system5/cpu3
```

Set the OEM property OEMxyzpropertyA.

```
-> set /system1 OEMxyzpropertyA=somevalue_
/system1
     OEMxyzpropertyA equals somevalue.
```

Set a property. Vendor returns data outside the spec.

```
-> set –o format=keyword /system1 PrimaryOwnerName=TheOwner
commandline=set –o format=keyword /system1
PrimaryOwnerName=TheOwner
status=0
job_id=7632
command=set
begingroup=instance
ufip=/system78/port2
begingroup=property
property_name=PrimaryOwnerName
property_val=TheOwner
endgroup
endgroup
oemxyz_message=The Contact information may need to be
updated.
endoutput
```

Set a property. Vendor returns data outside the spec.

```
-> set –o format=clpcsv /system1 PrimaryOwnerName=TheOwner
header,commandline
group,set –o format=clpcsv /system1
PrimaryOwnerName=TheOwner
header,status
group,0
```

```
header,job_id
group,7632
header,instance
header,ufip
group,/system1
header,properties
header,property_name,property_val,property_valstring,proper
ty_val,property_valstring
group,PrimaryOwner,TheOwner
header,oemxyz_message
group, The Contact information may need to be updated.
header,endoutput
```

Enable software2 for memory1. The association is explicitly identified.

-> set /system1/memory1=>ElementSoftwareIdentity=>/system1/swid1/software2 IsCurrent=true

```
software2 is now active for memory1.
```

Attempt to change which software2 is current on memory1 fails because there are multiple instances of ElementSoftwareIdentity which reference memory1.

-> set /system1/memory1=>ElementSoftwareIdentity IsCurrent=true

```
Failed to set property “IsCurrent”.  Ambiquous address
“/system1/memory1=>ElementSoftwareIdentity”
```

Enable software2. Only one instance of ElementSoftwareIdentify references software2.

-> set /system1/swid1/software2=>ElementSoftwareIdentity IsCurrent=true

```
software2 is now active for memory1.
```

Attempt to suspend a currently running job.

-> set /map1/jobqueue1/job23 jobstate=suspend

```
Failed to suspend job23.  The targeted job does not support
suspension.
```

Stop traffic over nic1.

```
-> set /system4/nic1 enabledstate=quiesce
Traffic over nic1 has been quiesced.
```

4.10 show

The general form of the `show` command is:

`show [<options>] [<`*`target`*`>] [<`*`properties`*`>]`

For the `show` command, implementations MUST support the syntax defined for the `show-cmd` term in the CLP Grammar defined in Section 4.14.

The `show` command is used to display information about Managed Elements. It can be used to view information about a single Managed Element, a tree of Managed Elements, or Managed Elements matching a property value filter. When executed against a Resultant Address which ends in a UFiT without any other options, the `show` command will display information about the single instance identified by the Resultant Address. When executed against a Resultant Address which ends in a UFsT, the `show` command will display information about contained instances of the type specified by the UFcT specified in the UFsT. When used with the `level` option, the `show` command can be used to view a tree of Managed Elements.

The `display` option can be used with the `show` command to control the instances for which information is returned, as well as the information returned for an instance. Using the `properties` argument to the `display` option, with the `show` command a user can effectively search the address space (or a branch) for instances having a certain property or a property/value pair. When the `show` command and `display` option are used in this fashion, it effectively provides a query function. This `display` option can also be used to restrict the Managed Element instances and Association instances for which information will be returned using the `targets` and `associations` arguments respectively. For more information on using the display option, see Section 5.4.

If the Resultant Address terminates in a UFsT, the path up to and including the penultimate term of the Resultant Address determines the Resultant Target of the command. The implementation MUST return each instance which is contained within the Resultant Target of the type specified by the UFcT specified in the UFsT. The implementation MUST search the containment hierarchy below the Resultant Target for instances of the type specified by the UFcT specified in the UFsT to the depth specified by the `level` option. The implementation MUST return information about the contained instances such that the information returned conforms with any restrictions or expansions specified by other options to the command. It is possible that zero instances are contained and thus there will not be any instances for which to return information. This is not an error and will be handled by implementations returning an appropriate representation of an empty result set.

If the `show` command is specified without the `display` option, implementations MUST interpret the command as if the command was specified with the `display` option with an argument of `all`. If the `show` command is specified without the `level` option, the implementation MUST behave as if the `level` option was specified with an argument of ‘1’.

The command MUST support usage with and without options.

4.10.1 Valid targets

All targets and target/property combinations are valid for this command and its behavior generally does not change based on target or property. In specific profiles (i.e. logs) the behavior may be slightly different as defined in that profile. Implementations of the `show` command MUST accept an Absolute or a Relative Target Address for the command target term.

4.10.2 Options

The following are valid options for the show command in addition to those specified in Table 16.

`-l, -level <`*`value`*`>`	Controls the target depth level for the containment hierarchy retrieval. Default for <value> is "1" ie. "the current target only". Other values can retrieve "current target plus 'n-1' levels deep". To retrieve all levels recursively the argument value "`all`" can be used with the `level` option. All MAP implementations SHOULD support this option.
`-d, -display <`*`arg values`*`>`	Selects the category of information that is displayed about a target ME. Valid option argument values for this option include `associations`, `targets`, `properties`, `verbs`, and `all`. All implementations MUST support this option and any option argument values that are applicable to a specific target. OEMs may also add values using the OEM_ namespace as defined in the OEM extensions section of this document. When this option is not specified, the show command behaves as if this option was included with an argument of `all`.
`-a, -all`	This option instructs the implementation to return all data element types subject to any filtering of categories by the `display` option. Implementations MUST support this option.

The following table lists each type of data element that may be returned by the `show` command and indicates whether the type requires the `all` option be specified in the command line in order to be included in the Command Results. A value of "yes" in a cell indicates that elements of the corresponding data element type will not returned unless the `all` option is included with the `show` command. For each data element type listed in Table 14 Data Element Types and `all` option such that the column labeled "-all required" includes a value of "yes", implementations MUST return elements of the specified data element type if and only if the `all` option is specified. For each data element type listed in Table 14 Data Element Types and `all` option such that the column labeled "-all required" is blank, implementations MUST return elements of the specified data element type irrespective of whether or not the `all` option is specified.

Table 14 Data Element Types and `all` option

Data Element Type	Corresponding display argument	-all required
Required Properties	properties	
Recommended Properties	properties	Yes
Core Properties	properties	Yes
OEM Properties	properties	Yes
SM CLP verbs	verbs	
OEM verbs	verbs	Yes
Addressing Associations	associations	
Non-addressing Associations	associations	
SM CLP targets	targets	
OEM Targets	targets	Yes

4.10.3 Output

4.10.3.1 Text Format

When contained targets are returned, the Command Results output MUST include a list of contained targets. If the Command Results contain multiple targets, the implementation MUST return results such that the target containment hierarchy is unambiguous. If properties are returned, the implementation MUST return results such that the target to which the properties belong is unambiguous. If the command resulted in no data, the implementation SHOULD not return any data.

4.10.3.2 Structured Format

The returned data MUST include any status data in the standard format at the top of the response.

4.10.3.2.1 XML Output

The XML document fragment indicates the general form of XML encoded Command Results for the `show` command. When show is used with the `level` option, multiple instances of the <target> element will be returned. Target containment hierarchy is explicitly indicated via <target> element nesting. The implementation MUST return the `show` element in the `response` element as defined in the Command Response schema in 6.3.2Appendix B.

```
<show>
     <target>
     <instance>
          <ufip>
User Friendly instance Path identifying target
```

```
               </ufip>
               <properties>Properties of the Managed Element.
               <property>
     A property of Managed Element. Each property element is
     defined per the xsd.
               </property>
               <property>
     A property of Managed Element. Each property element is
     defined per the xsd.
               </property>
               </properties>
            <associations>
                <association>
                    <ufct>Association class name</ufct>
                    <ufip>User Friendly instance Path of other
referenced Managed Element.</ufip>
                </association>
            </associations>
            <verbs>
               <standardverbs> CLP term separator delimited list
of CLP verbs</standardverbs>
               <oemverbs>CLP term separator delimited list of
OEM verbs</oemverbs>
            </verbs>
          </instance>
          <target>Recursive target elements representing Managed
Elements contained in initial target.</target>
          </target>
     </show>
```

4.10.3.2.2 CSV

When the show command is used with the `–level` option, results for multiple Managed Elements MAY be returned. Each Managed Element is returned as a block starting with:

header,instance

Unlike the "clpxml" format, the target containment hierarchy is not indicated in the Command Results structure. Instead, the `targets` header identifies that a "clpcsv" table of UFiPs of contained Managed Elements will follow if this target contains other targets.

Implementations MUST use the following form when returning Command Results for the `set` command in "clpcsv" format.

```
header,command
group,show
header,instance
header,ufip
group,User Friendly Instance Path of Managed Element results are
for
```

```
header,properties
header,property_name,property_val,property_valstring,...,propert
y_val,property_valstring,property_type,property_description,prop
erty_readonly,property_maxlen,property_maxvalue,property_minvalu
e,units,value,valuemap,...value,valuemap
group,property name,property value,string rep of
value,..,value,string rep of
value,type(MOF),description(MOF),readonly(MOF),maxlen(MOF),max
value(MOF),min value(MOF),units(MOF),enum value,map for enum
value,...,enum value,map for enum value,
        .
        .
        .
group,name,value,string rep of value,..,value,string rep of
value,type(MOF),description(MOF),readonly(MOF),maxlen(MOF),max
value(MOF),min value(MOF),units(MOF),enum value,map for enum
value,...,enum value,map for enum value,
header,targets
header,ufip,....,ufip
group,contained UFiP 1,...,contained UFiP n
header,verbs
header,verb,...,verb
group,verb name,...,verb name
header,associations
header,association
header,ufct,ufip
group,UFcT of association,UFiP of other reference
header,property_name,property_val,property_valstring,...,propert
y_val,property_valstring,property_type,property_description,prop
erty_readonly,property_maxlen,property_maxvalue,property_minvalu
e,units,value,valuemap,...value,valuemap
group,property name,property value,string rep of
value,..,value,string rep of
value,type(MOF),description(MOF),readonly(MOF),maxlen(MOF),max
value(MOF),min value(MOF),units(MOF),enum value,map for enum
value,...,enum value,map for enum value
        .
        .
        .
group,property name,property value,string rep of
value,..,value,string rep of
value,type(MOF),description(MOF),readonly(MOF),maxlen(MOF),max
value(MOF),min value(MOF),units(MOF),enum value,map for enum
value,...,enum value,map for enum value

        .
        .
        .
```

```
header,association
header,ufct,ufip
group,UFcT of association,UFiP of other reference
header,property_name,property_val,property_valstring,...,propert
y_val,property_valstring,property_type,property_description,prop
erty_readonly,property_maxlen,property_maxvalue,property_minvalu
e,units,value,valuemap,...value,valuemap
group,property name,property value,string rep of
value,..,value,string rep of
value,type(MOF),description(MOF),readonly(MOF),maxlen(MOF),max
value(MOF),min value(MOF),units(MOF),enum value,map for enum
value,...,enum value,map for enum value

        .
        .
        .

group,property name,property value,string rep of
value,..,value,string rep of
value,type(MOF),description(MOF),readonly(MOF),maxlen(MOF),max
value(MOF),min value(MOF),units(MOF),enum value,map for enum
value,...,enum value,map for enum value
```

4.10.3.2.3 Keyword

Implementations MUST use the following form when returning Command Results for the `cd` command in "keyword" format.

```
command=show
begingroup=instance
ufip=UFiP of Managed Element results are for
begingroup=property
property_name=property name
property_val=property value
endgroup
     .
     Additional property groups
     .

begingroup=targets
ufip=UFiP of contained target
     .
     Additional contained targets
     .
endgroup

begingroup=association
ufct=UFcT of association
ufip=UFiP on other end of association
```

```
begingroup=property
property_name=Name of Property of association
property_val=value of property of association
endgroup
     .
     Additional properties of the association
     .
endgroup
     .
     Additional associations referencing this Managed Element
     .
begingroup=verbs
verb=Command applicable to this Managed Element
     .
     Additional commands applicable to this Managed Element
     .
endgroup
endgroup
     .
     Additional Managed Elements that are returned as results
     .
endoutput
```

4.10.4 Examples

Show all of the targets in the AdminDomain

```
-> show –display targets /
/
Targets:
     map1
     system1
     system2
     hw1
```

Show the commands that apply to the AdminDomain

```
-> show –display verbs /
show
cd
```

Show the value of the name property on system1

```
-> show –display properties=name /system1
name = deadweight
```

Display the physical containment hierarchy for chassis1

```
-> show –display targets –level all –o format=clpxml
/hw1/chassis1
```

```
     <?xml version="1.0" encoding="UTF-8"?>
     <response xmlns="http://www.dmtf.org/smash/2005/01/clpxml"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xsi:schemaLocation="http://www.dmtf.org/smash/2005/01/clpxml
     smclp_command_response.xsd">
          <command>
               <inputline>show -display targets -level all -o
format=clpxml /hw1/chassis1</inputline>
          </command>
          <cmdstat>
               <status>0</status>
               <status_tag>COMMAND COMPLETED</status_tag>
               <job>
                    <job_id>8734</job_id>
               </job>
          </cmdstat>
          <show>
               <target>
                    <instance>
                    <ufit ufct="chassis"
instance="1">chassis1</ufit>
                    <ufip>/chassis1</ufip>
                    </instance>
                    <target>
                         <instance>
                              <ufit ufct="board"
instance="1">board1</ufit>
                              <ufip>/chassis1/board1</ufip>
                         </instance>
                    <target>
                              <instance>
                              <ufit ufct="card"
instance="1">card1</ufit>
     <ufip>/chassis1/board1/card1</ufip>
                    </instance>
                    <target>
```

```
                              <instance>
                                    <ufit ufct="chip"
instance="1">chip1</ufit>

     <ufip>/chassis1/board1/card1/chip1</ufip>
                              </instance>
                    </target>
                    <target>
                       <instance>
                       <ufit ufct="chip"
instance="2">chip2</ufit>
                       <ufip>/chassis1/board1/card1/chip2</ufip>
                       </instance>
                    </target>
                    </target>
                    </target>
                    <target>
                         <instance>
                         <ufit ufct="powerpkg"
instance="1">powerpkg1</ufit>
                         <ufip>/chassis1/powerpkg1</ufip>
                         </instance>
                    </target>
                    <target>
                         <instance>
                           <ufit ufct="powerpkg"
instance="2">powerpkg2</ufit>
                           <ufip>/chassis1/powerpkg2</ufip>
                         </instance>
                    </target>
                    <target>
                         <instance>
                              <ufit ufct="fanpkg"
instance="1">fanpkg1</ufit>
                              <ufip>/chassis1/fanpkg1</ufip>
                         </instance>
                    </target>
                    <target>
                         <instance>
                           <ufit ufct="fanpkg"
instance="2">fanpkg2</ufit>
                           <ufip>/chassis1/fanpkg2</ufip>
                         </instance>
                    </target>
                    <target>
                         <instance>
                            <ufit ufct="fanpkg"
instance="3">fanpkg3</ufit>
                           <ufip>/chassis1/fanpkg3</ufip>
                         </instance>
                    </target>
                    <target>
                         <instance>
                            <ufit ufct="fanpkg"
instance="4">fanpkg4</ufit>
                        <ufip>/chassis1/fanpkg4</ufip>
                         </instance>
                    </target>
               </target>
          </show>
     </response>
```

Display the associations which reference the target.

-> show –display associations –o format=clpxml /system1/powersup3

```
     <?xml version="1.0" encoding="UTF-8"?>
     <response xmlns="http://www.dmtf.org/smash/2005/01/clpxml"
     xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
     xsi:schemaLocation="http://www.dmtf.org/smash/2005/01/clpxm
     l smclp_command_response.xsd">
          <command>
               <inputline>show -display associations -o
     format=clplxml /system1/powersup3</inputline>
          </command>
          <cmdstat>
               <status>0</status>
               <status_tag>COMMAND COMPLETED</status_tag>
               <job>
                    <job_id>129</job_id>
               </job>
          </cmdstat>
          <show>
               <target>
                    <instance>
                    <ufit ufct="powersup"
     instance="3">powersup3</ufit>
                         <ufip>/system1/powersup3</ufip>
                         <associations>
                              <association>
                                   <ufct>suppliespower</ufct>
                                   <reference>
                                        <name>dependent</name>
                                        <instance>
                                             <ufit ufct="system"
     instance="1">system1</ufit>

          <ufip>/system1</ufip>
                                        </instance>
                                   </reference>
                                   <reference>
                                        <name>antecedent</name>
                                        <instance>
                                             <ufit
     ufct="powersup" instance="3">powersup3</ufit>

          <ufip>/system1/powersup3</ufip>
                                        </instance>
                                   </reference>
                              </association>
                              <association>
                                   <ufct>realizes</ufct>
                                   <reference>
                                        <name>antecedent</name>
                                        <instance>
                                             <ufit
     ufct="powerpkg" instance="2">powerpkg2</ufit>

          <ufip>/chassis23/powerpkg2</ufip>
                                        </instance>
                                   </reference>
                                   <reference>
                                        <name>dependent</name>
                                        <instance>
                                             <ufit
     ufct="powersup" instance="3">powersup3</ufit>

          <ufip>/system1/powersup3</ufip>
                                        </instance>
                                   </reference>
                              </association>
                              <association>
                                   <ufct>SystemDevice</ufct>
                                   <reference>

          <name>partcomponent</name>
                                        <instance>
                                             <ufit
     ufct="powersup" instance="3">powersup3</ufit>

          <ufip>/system1/powersup3</ufip>
                                        </instance>
                                   </reference>
                                   <reference>
          <name>groupcomponent</name>
                                        <instance>
                                             <ufit ufct="system"
     instance="1">system1</ufit>

          <ufip>/system1</ufip>
                                        </instance>
                                   </reference>
                              </association>
                         </associations>
                    </instance>
               </target>
          </show>
     </response>
```

Display the SuppliesPower and Realizes associations which reference the target.

-> show –display associations=(SuppliesPower,Realizes) –o format=clpxml /system1/powersup3

```
<?xml version="1.0" encoding="UTF-8"?>
<response xmlns="http://www.dmtf.org/smash/2005/01/clpxml"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xsi:schemaLocation="http://www.dmtf.org/smash/2005/01/clpxm
l

smclp_command_response.xsd">
     <command>
          <inputline>show -display
associations=(suppliespower,realizes)
/system1/powersup3</inputline>
     </command>
     <cmdstat>
          <status>0</status>
          <status_tag>COMMAND COMPLETED</status_tag>
          <job>
               <job_id>129</job_id>
          </job>
     </cmdstat>
     <show>
          <target>
               <instance>
               <ufit ufct="powersup"
instance="3">powersup3</ufit>
                    <ufip>/system1/powersup3</ufip>
                    <associations>
                         <association>
                              <ufct>suppliespower</ufct>
                              <reference>
                                   <name>dependent</name>
                                   <instance>
                                        <ufit ufct="system"
instance="1">system1</ufit>

     <ufip>/system1</ufip>
                                   </instance>
                              </reference>
                              <reference>
                                   <name>antecedent</name>
                                   <instance>
                                        <ufit
ufct="powersup" instance="3">powersup3</ufit>

     <ufip>/system1/powersup3</ufip>
                                   </instance>
                              </reference>
                         </association>
                         <association>
                              <ufct>realizes</ufct>
                              <reference>
                                   <name>antecedent</name>
                                   <instance>
                                        <ufit
ufct="powerpkg" instance="2">powerpkg2</ufit>
     <ufip>/chassis23/powerpkg2</ufip>
                                   </instance>
                              </reference>
                              <reference>
                                   <name>dependent</name>
                                   <instance>
                                        <ufit
ufct="powersup" instance="3">powersup3</ufit>

     <ufip>/system1/powersup3</ufip>
                                   </instance>
                              </reference>
                         </association>
                    </associations>
               </instance>
          </target>
     </show>
</response>
```

View the status of the spawned job above. In this example, the spawned job failed to run to completion.

-> show –o format=clpxml /map1/job1/job385

```
<?xml version="1.0" encoding="UTF-8"?>
<response xmlns="http://www.dmtf.org/smash/2005/01/clpxml"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xsi:schemaLocation="http://www.dmtf.org/smash/2005/01/clpxm
l
smclp_command_response.xsd">
     <command>
          <inputline>show -o format=clpxml
/map1/job1/job385</inputline>
     </command>
     <cmdstat>
          <status>0</status>
```

```
            <job>
                 <job_id>892</job_id>
            </job>
       </cmdstat>
       <show>
            <target>
                 <instance>
                 <ufit ufct="job"
  instance="385">job385</ufit>
                      <ufip>/map1/jobqueue1/job385</ufip>
                      <properties>
                           <property>
                                <name>JobState</name>
                                <value>
                                     <val>10</val>
       <valstring>Exception</valstring>
                                </value>
                           </property>
                           <property>

       <name>TimeOfLastStateChange</name>

       <value><val>20050130145904.000000-300</val></value>
                                <type>datetime</type>
                           </property>
                           <property>
       <name>TimeBeforeRemoval</name>

       <value><val>00000000000500.000000:000</val></value>
                                </property>
                            <property>
                                <name>ElapsedTime</name>

       <value><val>00000000000034.000000:000</val></value>
                                </property>
```

```
                              <property>
                              <name>StartTime</name>

        <value><val>20050130145830.000000-300</val></value>
                              </property>
                              <property>
                              <name>TimeSubmitted</name>

        <value><val>20050130145830.000000-300</val></value>
                              </property>
                         <property>
        <name>TimeBeforeRemoval</name>

        <value><val>00000000000500.000000:000</val></value>
                         </property>
                              <property>
                              <name>name</name>
                              <value>
                                   <val>dump -destination
   ftp://adminstrator:passw0rd@myserver.com/private/administra
   tor/memory.dmp memory1</val>
                              </value>
                         </property>
                    </properties>
               </instance>
          </target>
     </show>
</response>
```

The command below would show all the profiles and subprofiles and their properties implemented in the MAP Admin Domain

```
-> show -all /profiles1/profile
UFIT : /profiles1
  UFIT : /profiles1/profile1
     RegisteredName : SM MAP Profile
     RegisteredVersion : 1.0
```

```
      RegisteredOrganization : DMTF
      AdvertiseType : SLP
    UFIT : /profiles1/profile2
      RegisteredName : SM Base System Profile
      RegisteredVersion : 1.0
      RegisteredOrganization : DMTF
      AdvertiseType : SLP
```

-> show =>elementconformstoprofile
/system1

```
UFIP : /profiles1/profile2
```

The command below would show the names (RegisteredName property) of the profile and subprofile(s) implemented for a specific computer system.

-> show –display properties=registeredname profiles1/profile2

```
RegisteredName : SM Base System Profile
```

-> show /profiles1/profile2=>subprofilerequiresprofile

```
UFIP : /profiles1/subprofile2
```

This command queries the value of the PrimaryOwnerName and ResetCapability properties by using the –display properties=ResetCapability:PrimaryOwnerName as a filter on these results.

-> show –display properties=ResetCapability:PrimaryOwnerName\ /system1

```
  /system1
    properties
      ResetCapability = Disabled (3)
      PrimaryOwnerName = Some guy
```

This command queries the value of the PrimaryOwnerName and Dedicated properties using the –display properties=ResetCapability:Dedicated as a filter on these results.

```
-> show –display `
properties=(ResetCapability,PrimaryOwnerName) –o
format=clpxml /system1
```

```
<?xml version="1.0" encoding="UTF-8"?>

<response xmlns="http://www.dmtf.org/smash/2005/01/clpxml"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xsi:schemaLocation="http://www.dmtf.org/smash/2005/01/clpxml
smclp_command_response.xsd">

  <command>

     <inputline>show -display
properties=(ResetCapability,PrimaryOwnerName) -o
format=clpxml /system1 </inputline>

  </command>

  <cmdstat>

     <status>0</status>

  </cmdstat>

  <show>

     <target>

        <instance>

          <ufit ufct="system" instance="1">system1</ufit>

           <ufip>/system1</ufip>

           <properties>

              <property>

                 <name>ResetCapability</name>

                 <value>

                    <val>3</val>

                               <valstring>Disabled</valstring>

                 </value>

              </property>

              <property>

                 <name>Dedicated</name>

                          <multivalue>

                 <value>
```

```
                    <val>4</val>
                               <valstring>Router</valstring>
                 </value>
                 <value>
                    <val>3</val>
                               <valstring>Switch</valstring>
                 </value>
                          </multivalue>
               </property>
            </properties>
         </instance>
      </target>
   </show>
</response>
```

This command displays all of the commands applicable to system1

-> show –display verbs –all –o format=clpxml /system1

```
<?xml version="1.0" encoding="UTF-8"?>
<response xmlns="http://www.dmtf.org/smash/2005/01/clpxml"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xsi:schemaLocation="http://www.dmtf.org/smash/2005/01/clpxml
smclp_command_response.xsd">
   <command>
      <inputline>show -display verbs -all -o format=clpxml
/system1</inputline>
   </command>
   <cmdstat>
      <status>0</status>
      <job>
         <job_id>2342</job_id>
      </job>
   </cmdstat>
   <show>
      <target>
```

```
          <instance>
          <ufit ufct="system" instance="1">system1</ufit>
            <ufip>/system1</ufip>
            <verbs>
              <standardverbs>show start stop set reset
   help</standardverbs>
              <oemverbs>OEMxyzDoSomething</oemverbs>
            </verbs>
          </instance>
        </target>
      </show>
   </response>
```

This command displays information about system1 and everything immediately contained within it.

-> show –level 2 –o format=clpxml /system1

```
   <?xml version="1.0" encoding="UTF-8"?>
<response xmlns="http://www.dmtf.org/smash/2005/01/clpxml"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xsi:schemaLocation="http://www.dmtf.org/smash/2005/01/clpxml
smclp_command_response.xsd">
  <command>
    <inputline>show -level 1 -o format=clpxml
/system1</inputline>
  </command>
  <cmdstat>
    <status>0</status>
    <status_tag>COMMAND COMPLETED</status_tag>
    <job>
      <job_id>29345</job_id>
    </job>
  </cmdstat>
  <show>
    <target>
      <instance>
        <ufit ufct="system" instance="1">system1</ufit>
    <ufip>/system1</ufip>
        <properties>
          <property>
            <name>NameFormat</name>
            <value>
```

```
              <val>IP</val>
            </value>
          </property>
          <property>
<name>ResetCapability</name>
          <value>
            <val>5</val>
            </value>
          </property>
          <property>
            <name>IdentifyingDescriptions</name>
      <multivalue>
              <value>
                <val>mac address</val>
              </value>
              <value>
                <val>mac address</val>
              </value>
            </multivalue>
          </property>
          <property>
            <name>OtherIdentifyingInfo</name>
            <multivalue>
              <value>
                <val>0023342312</val>
              </value>
              <value>
                <val>0023342313</val>
              </value>
            </multivalue>
          </property>
          <property>
            <name>Dedicated</name>
            <multivalue>
              <value>
                <val>4</val>
                <valstring>Router</valstring>
              </value>
              <value>
                <val>5</val>
                <valstring>Switch</valstring>
              </value>
            </multivalue>
          </property>
        </properties>
        <associations>
          <association>
            <ufct>ComputerSystemPackage</ufct>
```

```
               <reference>
                  <name>antecedent</name>
                  <instance>
                     <ufit ufct="chassis"
instance="3">chassis3</ufit>
                     <ufip>/hw1/chassis3</ufip>
                  </instance>
               </reference>
               <reference>
                  <name>dependent</name>
                  <instance>
                     <ufit ufct="system"
instance="1">system1</ufit>
                     <ufip>/system1</ufip>
                  </instance>
               </reference>
               <properties>
                  <property>
                     <name>PlatformGUID</name>
                     <value>
                  <val>00992365293059103762850194833920</val>
                  </value>
            </property>
               </properties>
            </association>
            <association>
   <ufct>SuppliesPower</ufct>
<reference>
                  <name>antecedent</name>
                  <instance>
                     <ufit ufct="powersupply"
instance="1">powersupply1</ufit>
   <ufip>/system1/powersupply1</ufip>
   </instance>
</reference>
     <reference>
   <name>dependent</name>
   <instance>
     <ufit ufct="cpu" instance="1">cpu1</ufit>
          <ufip>/system1</ufip>
   </instance>
</reference>
</association>
     <association>
   <ufct>SuppliesPower</ufct>
   <reference>
     <name>antecedent</name>
            <instance>
```

```
                    <ufit ufct="powersupply"
instance="2">powersupply2</ufit>
                    <ufip>/system1/powersupply2</ufip>
          </instance>
               </reference>
               <reference>
                 <name>dependent</name>
                 <instance>
                    <ufit ufct="cpu" instance="1">cpu1</ufit>
                 <ufip>/system1/cpu1</ufip>
     </instance>
</reference>
            </association>
            <association>
               <ufct>AssociatedCooling</ufct>
               <reference>
                 <name>antecedent</name>
                 <instance>
                 <ufit ufct="fan" instance="1">fan1</ufit>
               <ufip>/system1/fan1</ufip>
       </instance>
               </reference>
               <reference>
                 <name>dependent</name>
                 <instance>
                    <ufit ufct="system"
instance="1">system1</ufit>
                    <ufip>/system1</ufip>
                 </instance>
               </reference>
            </association>
            <association>
               <ufct>AssociatedCooling</ufct>
               <reference>
                 <name>antecedent</name>
                 <instance>
                    <ufit ufct="system"
instance="1">system1</ufit>
  <ufip>/system1/fan2</ufip>
  </instance>
               </reference>
               <reference>
                 <name>dependent</name>
                 <instance>
                    <ufit ufct="system"
instance="1">system1</ufit>
                    <ufip>/system1</ufip>
                 </instance>
               </reference>
            </association>
            <association>
               <ufct>Realizes</ufct>
               <reference>
                  <name>antecedent</name>
                  <instance>
                     <ufit ufct="ppowersupply"
instance="1">ppowersupply1</ufit>
  <ufip>/hw1/chassis3/ppowersupply1</ufip>
     </instance>
</reference>
               <reference>
                  <name>dependent</name>
                  <instance>
                     <ufit ufct="powersupply"
instance="1">powersupply1</ufit>
  <ufip>/system1/powersupply1</ufip>
  </instance>
</reference>
            </association>
            <association>
               <ufct>realizes</ufct>
               <reference>
                  <name>dependent</name>
                  <instance>
  <ufit ufct="powersupply" instance="2">powersupply2</ufit>
               <ufip>/system1/powersupply2</ufip>
       </instance>                          </reference>
   <reference>
                  <name>antecedent</name>
                  <instance>
                     <ufit ufct="ppowersupply"
instance="2">ppowersupply2</ufit>
  <ufip>/hw1/chassis3/ppowersupply2</ufip>
                  </instance>
               </reference>
            </association>
            <association>
  <ufct>Realizes</ufct>
               <reference>
                  <name>antecedent</name>
                  <instance>
                     <ufit ufct="ppowersupply"
instance="2">ppowersupply2</ufit>
  <ufip>/hw1/chassis3/ppowersupply2</ufip>
     </instance>
               </reference>
               <reference>
                  <name>dependent</name>
                  <instance>
                     <ufit ufct="powersupply"
instance="2">powersupply2</ufit>
  <ufip>/system1/powersupply2</ufip>
                  </instance>
               </reference>
            </association>
          </associations>
          <verbs>
            <standardverbs>show set stop start
reset</standardverbs>
          </verbs>
       </instance>
       <target>
          <instance>
            <ufit ufct="cpu" instance="1">cpu1</ufit>
            <ufip>/system1/cpu1</ufip>
            <properties>
               <property>
                  <name>Family</name>
                  <value>
                     <val>1</val>
                  </value>
               </property>
               <property>
                  <name>OtherFamilyDescription</name>
          <value>
               <val>SuperSlow100</val>
                  </value>
               </property>
               <property>
                  <name>MaxClockSpeed</name>
                  <value>
                     <val>33</val>
                  </value>
                  <units>Megahertz</units>
               </property>
               <property>
                  <name>CPUStatus</name>
                  <value>
                     <val>1</val>
                  </value>
               </property>
            </properties>
            <verbs>
               <standardverbs>show</standardverbs>
```

```
             </verbs>
          </instance>
        </target>
        <target>
          <instance>
             <ufit ufct="powersup" instance="1">powersup1</ufit>
          <ufip>/system1/powersup1</ufip>
             <properties>
               <property>
                  <name>TotalOutputPower</name>
                  <value>
                    <val>1200000</val>
                  </value>
                  <units>milliwatts</units>
               </property>
             </properties>
             <associations>
             <association>
                  <ufct>SuppliesPower</ufct>
                    <reference>
                               <name>dependent</name>
                               <instance>
                                    <ufit ufct="system"
instance="1">system1</ufit>
                                    <ufip>/system1</ufip>
                               </instance>
                         </reference>
                         <reference>
                               <name>antecedent</name>
                               <instance>
                                    <ufit ufct="powersup"
instance="1">powersup1</ufit>
        <ufip>/system1/powersup1</ufip>
                               </instance>
                         </reference>
               </association>
               <association>
                  <ufct>Realizes</ufct>
                  <reference>
                    <name>antecedent</name>
                    <instance>
                         <ufit ufct="powerpkg"
instance="1">powerpkg1</ufit>
   <ufip>/hw1/chassis3/powerpkg1</ufip>
                    </instance>
                  </reference>
                  <reference>
                    <name>dependent</name>
               <instance>
<ufit ufct="powersup" instance="1">powersup1</ufit>
               <ufip>/system1/powersup1</ufip>
                    </instance>
                  </reference>
               </association>
          </associations>
            <verbs>
               <standardverbs>show</standardverbs>
            </verbs>
          </instance>
        </target>
        <target>
          <instance>
          <ufit ufct="powersup" instance="2">powersup2</ufit>
          <ufip>/system1/powersup2</ufip>
            <properties>
               <property>
                  <name>TotalOutputPower</name>
                  <value>
                    <val>1200000</val>
                  </value>
                  <units>milliwatts</units>
               </property>
            </properties>
          <associations>
            <association>
               <ufct>SuppliesPower</ufct>
     <reference>
                  <name>dependent</name>
                  <instance>
                  <ufit ufct="system" instance="1">system1</ufit>
                    <ufip>/system1</ufip>
                  </instance>
               </reference>
               <reference>
                  <name>antecedent</name>
                  <instance>
                    <ufit ufct="powersup"
instance="2">powersup2</ufit>
                    <ufip>/system1/powersup2</ufip>
                  </instance>
               </reference>
            </association>
            <association>
               <ufct>Realizes</ufct>
               <reference>
                  <name>antecedent</name>
```

```
                 <instance>
                    <ufit ufct="powerpkg"
instance="2">powerpkg2</ufit>
  <ufip>/hw1/chassis3/powerpkg2</ufip>
                 </instance>
              </reference>
              <reference>
                 <name>dependent</name>
                 <instance>
                    <ufit ufct="powersup"
instance="2">powersup2</ufit>
                    <ufip>/system1/powersup2</ufip>
                 </instance>
              </reference>
           </association>
        </associations>
        <verbs>
           <standardverbs>show</standardverbs>
        </verbs>
     </instance>
</target>
     <target>
        <instance>
        <ufit ufct="fan" instance="1">fan1</ufit>
           <ufip>/system1/fan1</ufip>
           <properties>
              <property>
                 <name>VariableSpeed</name>
                 <value>
                    <val>true</val>
                 </value>
              </property>
           <property>
                 <name>DesiredSpeed</name>
                 <value>
                    <val>3600</val>
                 </value>
                 <units>Revolutions per inute</units>
              </property>
              <property>
                 <name>ActiveCooling</name>
                 <value>
                    <val>True</val>
                 </value>
              </property>
           </properties>
           <associations>
              <association>
```

```
                 <ufct>AssociatedCooling</ufct>
                 <reference>
                    <name>dependent</name>
                    <instance>
                         <ufit ufct="system"
instance="1">system1</ufit>
                         <ufip>/system1</ufip>
                    </instance>
                 </reference>
                 <reference>
                    <name>antecedent</name>
                    <instance>
                         <ufit ufct="fan"
instance="1">fan1</ufit>
                         <ufip>/system1/fan1</ufip>
                    </instance>
                 </reference>
              </association>
           </associations>
           <verbs>
              <standardverbs>show set</standardverbs>
           </verbs>
        </instance>
     </target>
     <target>
        <instance>
        <ufit ufct="fan" instance="2">fan2</ufit>
           <ufip>/system1/fan2</ufip>
           <properties>
              <property>
                 <name>VariableSpeed</name>
                 <value>
                    <val>true</val>
                 </value>
              </property>
              <property>
                 <name>DesiredSpeed</name>
                 <value>
                    <val>3600</val>
                 </value>
                 <units>Revolutions per minute</units>
              </property>
              <property>
                 <name>ActiveCooling</name>
                 <value>
                    <val>True</val>
                 </value>
              </property>
```

```
            </properties>
            <associations>
              <association>
                <ufct>AssociatedCooling</ufct>
                <reference>
                  <name>dependent</name>
                  <instance>
                      <ufit ufct="system"
instance="1">system1</ufit>
                      <ufip>/system1</ufip>
                  </instance>
                </reference>
                <reference>
                  <name>antecedent</name>
                  <instance>
                      <ufit ufct="fan"
instance="2">fan2</ufit>
                      <ufip>/system1/fan2</ufip>
                  </instance>
                </reference>
              </association>
            </associations>
            <verbs>
              <standardverbs>show set</standardverbs>
            </verbs>
          </instance>
        </target>
      </target>
  </show>
</response>
```

This command displays information about system1 and everything immediately contained within it.

-> show –level 2 –o format=clpcsv /system1

```
header,commandline
group,show -level 2 –o format=clpcsv /system1
header,status,status_tag
group,0,COMMAND COMPLETED
header,job_id
group,29345
header,command
group,show
header,instance
header,ufip
group,/system1
header,properties
header,property_name,property_val
```

```
group,NameFormat,IP
group,ResetCapability,5
header,property_name,property_val,property_val
group,IdentifyingDescriptions,mac address, mac address
group,OtherIdentifyingInfo,0023342312,0023342313
header,property_name,property_val,property_valstring,property
_val,property_valstring
group,Dedicated,4,Router,5,Switch
header,targets
header,ufip,ufip,ufip,ufip
group,powersup1,powersup2,fan1,fan2
header,associations
header,association
header,ufct,ufip
group,ComputerSystemPackage,/hw1/chassis3
header,property_name,property_val
group,PlatformGUID,00992365293059103762850194833920
header,association
header,ufct,ufip
group,SuppliesPower,/system1/powersup1
header,association
header,ufct,ufip
group,SuppliesPower,/system1/powersup2
header,association
header,ufct,ufip
group,AssociatedCooling,/system1/fan1
header,association
header,ufct,ufip
group,AssociatedCooling,/system1/fan2
header,verbs
header,verb,verb,verb,verb,verb
group,show,set,stop,start,reset
header,instance
header,ufip
group,/system1/cpu1
header,properties
header,property_name,property_val
group,Family,1
group,OtherFamilyDescription,SuperSlow100
header,property_name,property_val,units
group,MaxClockSpeed,33,Megahertz
header,property_name,property_val
group,CPUStatus,1
header,verbs
header,verb
group,show
```

```
header,instance
header,ufip
group,/system1/powersup1
header,properties
header,property_name,property_val,units
group,TotalOutputPower,1200000,milliwatts
header,associations
header,association
header,ufct,ufip
group,Realizes,/hw1/chassis3/powerpkg1
header,association
header,ufct,ufip
group,SuppliesPower,/system1
header,verbs
header,verb
group,show
header,instance
header,ufip
group,/system1/powersup2

header,properties
header,property_name,property_val,units
group,TotalOutputPower,1200000,milliwatts
header,associations
header,association
header,ufct,ufip
group,Realizes,/hw1/chassis3/powerpkg2
header,association
header,ufct,ufip
group,SuppliesPower,/system1
header,verbs
header,verb
group,show
header,instance
header,ufip
group,/system1/fan1
header,properties
header,property_name,property_val,units
group,DesiredSpeed,3600,Revolutions per Minutes
header,property_name,property_val
group,VariableSpeed,true
group,ActiveCooling,true
header,associations
header,association
header,ufct,ufip
group,AssociatedCooling,/system1
header,verbs
header,verb,verb
group,show,set
```

```
header,instance
header,ufip
group,/system1/fan2
header, properties
header,property_name,property_val,units
group,DesiredSpeed,3600,Revolutions per Minutes
header,property_name,property_val
group,VariableSpeed,true
group,ActiveCooling,true
header,associations
header,association
header,ufct,ufip
group,AssociatedCooling,/system1
header,verbs
header,verb,verb
group,show,set
```

This command displays information about system1 and everything immediately contained within it.

-> show –level 2 –o format=keyword /system1

```
commandline=show -level 2 –o format=keyword /system1
status=0
status_tag=COMMAND COMPLETED
job_id=29345
command=show
begingroup=instance
ufip=/system1
begingroup=property
property_name=NameFormat
property_val=IP
endgroup
begingroup=property
property_name=ResetCapability
property_val=5
endgroup
begingroup=property
property_name=IdentifyingDescriptions
property_val=mac address
property_val=mac address
endgroup
begingroup=property
property_name=OtherIdentifyingInfo
property_val=0023342312
property_val=0023342313
endgroup
begingroup=property
property_name=Dedicated
```

```
property_val=4
property_valstring=Router
property_val=5
property_valstring=Switch
endgroup
begingroup=targets
ufip=powersup1
ufip=powersup2
ufip=fan1
ufip=fan2
endgroup
begingroup=association
ufct=ComputerSystemPackage
ufip=/hw1/chassis3
begingroup=property
property_name=PlatformGUID
property_val=00992365293059103762850194833920
endgroup
endgroup
begingroup=association
ufct=SuppliesPower
ufip=/system1/powersup1
endgroup
begingroup=association
ufct=SuppliesPower
ufip=/system1/powersup2
endgroup
begingroup=association
ufct=AssociatedCooling
ufip=/system1/fan1
endgroup
begingroup=association
ufct=AssociatedCooling
ufip=/system1/fan2
endgroup
begingroup=verbs
verb=reset
verb=start
verb=stop
verb=set
verb=show
endgroup
endgroup
begingroup=instance
ufip=/system1/cpu1
begingroup=property
property_name=Family
property_val=1
```

```
endgroup
begingroup=instance
property_name=OtherFamilyDescription
property_val=SuperSlow100
endgroup
begingroup=instance
property_name=MaxClockSpeed
property_val=33
units=Megahertz
endgroup
begingroup=instance
property_name=CPUStatus
property_val=1
endgroup
begingroup=verbs
verb=show

endgroup
endgroup
begingroup=instance
ufip=/system1/powersup1
begingroup=property
property_name=TotalOutputPower
property_val=1200000
units=milliwatts
begingroup=association
ufct=Realizes
ufip=/hw1/chassis3/powerpkg1
endgroup
begingroup=association
ufct=SuppliesPower
ufip=/system1
endgroup
begingroup=verbs
verb=show
endgroup
begingroup=instance
ufip=/system1/powersup2
begingroup=property
property_name=TotalOutputPower
property_val=1200000
units=milliwatts
endgroup
begingroup=association
ufct=Realizes
ufip=/hw1/chassis3/powerpkg2
endgroup
begingroup=association
```

```
ufct=SuppliesPower
ufip=/system1
endgroup
begingroup=verbs
verb=show
endgroup
begingroup=instance
ufip=/system1/fan1
begingroup=property
property_name=DesiredSpeed
property_val=3600
units=Revolutions per Minutes
endgroup
begingroup=property
property_name=VariableSpeed
property_val=true
property_name=ActiveCooling
property_val=true
endgroup
begingroup=association
ufct=AssociatedCooling
ufip=/system1
endgroup
begingroup=verbs
verb=show
verb=set
endgroup
begingroup=instance
ufip=/system1/fan2
begingroup=property
property_name=DesiredSpeed
property_val=3600
units=Revolutions per Minutes
endgroup
begingroup=property
property_name=VariableSpeed
property_val=true
endgroup
begingroup=property
property_name=ActiveCooling
property_val=true
endgroup
begingroup=association
ufct=AssociatedCooling
ufip=/system1
endgroup
begingroup=verbs
verb=show
```

```
verb=set
endgroup
endoutput
```

Show the OperationalStatus of components in system1.

```
-> show –display properties=(OperationalStatus,Name) `
 –level 2  /system1
/system1
  OperationalStatus is OK
  Name is webserver1
/system1/cpu1
  OperationalStatus is OK
  Name is main processor 1
/system1/cpu2
  OperationalStatus is OK
  Name is main processor 2
/system1/cpu3
  OperationalStatus is OK
  Name is main processor 3
/system1/cpu4
  OperationalStatus is Degraded
  Name is main processor 4
/system1/powersup1
  OperationalStatus is Degraded
  Name is AC power 1
/system1/powersup2
  OperationalStatus is OK
  Name is AC power 2
```

Show all of the CPUs that have a bad operational status.

```
-> show –display `
properties=(OperationalStatus==Degraded,Name, `
OperationalStatus)  /system1/cpu*
  /system1/cpu4
     Name is main processor 4
```

```
        OperationalStatus is Degraded
```

Show all of the components that have a bad operational status.

```
-> show -level 2 –display `
properties=(OperationalStatus,OperationalStatus==Degraded) `
/system1
  /system1/cpu4
     OperationalStatus is Degraded
  /system1/powersup1
     OperationalStatus is Degraded
```

Show all of the components that are named “main processor 4”

```
-> show -level 2 –display properties=(Name,Name==`
”Main processor 4”) /system1
  /system1/cpu4
     Name is main processor 4
```

Try to filter using two property values

```
-> show -level 2 –display properties=(Name,Name==`
”Main processor 4”,OperationalStatus==Degraded) /system1
Invalid argument.  Filtering is only supported for a single
property.
```

Show the values for options which can have default values.

```
-> show –display properties –o format=clpxml SESSION
<?xml version="1.0" encoding="UTF-8"?>
<response xmlns="http://www.dmtf.org/smash/2005/01/clpxml"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xsi:schemaLocation="http://www.dmtf.org/smash/2005/01/clpxml
smclp_command_response.xsd">
      <command>
            <inputline>show -display properties -o format=clpxml
SESSION</inputline>
      </command>
      <cmdstat>
            <status>0</status>
      </cmdstat>
      <show>
            <target>
                  <instance>
                  <ufit ufct="setting" instance="5">setting5</ufit>
                        <ufip>/map1/sessions1/setting5</ufip>
                        <properties>
                              <property>
                                    <name>cdt</name>
                                    <value>
                                          <val>/system1/cpu4</val>
                                    </value>
                              </property>
                              <property>
                                    <name>outputformat</name>
                                          <value>

      <val>4</val><valstring>xml</valstring>
                                                </value>
                              </property>
                              <property>
                                    <name>outputverbosity</name>
                                    <value>
                                          <val>terse</val>
                                          <valstring>1</valstring>
                                    </value>
                              </property>
                              <property>
                                    <name>outputlanguage</name>
                                    <value>
                                          <val>eng</val>
                                    </value>
                              </property>
                              <property>
                                    <name>outputposition</name>
                                    <value>
                                          <val>1</val>
                                          <valstring>begin</valstring>
                                    </value>
                              </property>
                              <property>
                                    <name>outputorder</name>
                                    <value>
                                          <val>1</val>
                                    <valstring>default</valstring>
                                    </value>
                              </property>
                              <property>
                                    <name>outputcount</name>
                                    <value>
                                          <val>0</val>
                                    </value>
                              </property>
                              <property>
                                    <name>keep</name>
                                    <value>
                                          <val>300</val>
                                    </value>
                              </property>
                              <property>
                                    <name>wait</name>
                                    <value>
                                          <val>false</val>
                                    </value>
                              </property>
                        </properties>
                  </instance>
            </target>
      </show>
</response>
```

4.11 start

The general form of the `start` command is:

`start [<`*`options`*`>] [<`*`target`*`>]`

For the start command, implementations MUST support the syntax defined for the `start-cmd` term in the CLP Grammar defined in Section 4.14.

The `start` command starts the target. If the target is in a state of running that is equal to or higher than the end state the command will return an error and not change the target's state.

This command supports usage with and without command line options.

4.11.1 Valid targets

This command is supported when it is specified in a target mapping [CLP-to-CIM Mapping Specification [13]] for a profile which the implementation supports. For all targets that do not support the usage of the `start` command, implementations MUST NOT show the `start` command in a command listing as being available. Implementations of the `start` command MUST accept an Absolute or a Relative Target Address for the command target term.

The behavior of state change commands for each UFcT is defined in the CLP-to-CIM Mapping Specification [13].

4.11.2 Options

The following are valid options for the start command in addition to those specified in Table 16.

`-f, -force`	Forces the implementation to start the object, ignoring any policy that might cause the implementation to normally not execute the command. The implementation MUST execute this start if at all possible, without regard to consequences.

4.11.3 Output

4.11.3.1 Text Format

Implementations MUST return Command Result data which includes the target address that was started (if any) and the time and date when the start was initiated. Implementations MAY return the time/date information in any applicable format that meets the implementations' needs. If the implementation did not start the target, the implementation MUST return Command Response data indicating that no targets were started.

4.11.3.2 Structured Format

Implementations MUST include any status data in the standard format at the top of the response. If the target was successfully started, the start command MUST then return the target started and the time the start completed

4.11.3.2.1 XML Output

The implementation MUST return the `start` element in the `response` element as defined in the Command Response schema in 6.3.2Appendix B.

```
<start>
     <ufip>UFiP of target to start</ufip>
     <timestamp>Time reset occurred if completed synchronously,
returned in CIM datetime format</timestamp>
</start>
```

4.11.3.2.2 CSV

When returning Command Results for the `start` command in “clpcsv” format, implementations MUST use the following general form:

```
header,command
group,start
header,ufip,timestamp
group,User Friendly instance path of the Managed Element
targeted by the start command, Time reset occurred if completed
synchronously, returned in CIM datetime format
header,endoutput
```

4.11.3.2.3 Keyword

Implementations MUST use the following form when returning Command Results for the `cd` command in “keyword” format.

```
command=start
ufip=User Friendly instance path of the Managed Element
timestamp= Time reset occurred if completed synchronously,
returned in CIM datetime format
endoutput
```

4.11.4 Examples

```
     Send system1 to its default enabled state
     -> start /system1
     /system1 started at 10:40am 1/1/01

     Attempt to start cpu2 fails because this command is not
     supported on the target.
     -> start –o format=clpxml /system1/cpu2
     <?xml version="1.0" encoding="UTF-8"?>
     <response xmlns="http://www.dmtf.org/smash/2005/01/clpxml"
     xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
     xsi:schemaLocation="http://www.dmtf.org/smash/2005/01/clpxm
     l
     smclp_command_response.xsd">
          <command>
               <inputline>start -o format=clpxml
     /system1/cpu2</inputline>
          </command>
          <cmdstat>
               <status>3</status>
               <job>
                    <job_id>234</job_id>
               <joberr>
                    <errtype>1</errtype>
                    <cimstat>7</cimstat>
                    <severity>2</severity>
               </joberr>
               </job>
          </cmdstat>
          <start>
          <instance>
               <ufit ufct="cpu" instance="2">cpu2</ufit>
               <ufip>/system1/cpu2</ufip>
          </instance>
          </start>
     </response>
```

Attempt to start cpu2 fails because this command is not supported on the target.
-> start –o format=clpcsv /system1/cpu2

```
header,commandline
group,start –o format=clpcsv /system1/cpu2
header,status
group,3
header,job_id
group,234
header,errtype,cimstat,severity
group,1,7,2
header,command
group,start
header,ufip
group,/system1/cpu2
header,endoutput
```

Attempt to start cpu2 fails because this command is not supported on the target.
-> start –o format=keyword /system1/cpu1

```
commandline=start –o format=keyword /system1/cpu2
status=3
job_id=234
```

```
errtype=1
cimstat=7
severity=2
command=start
ufip=/system1/cpu2
endoutput
```

Start system1 and wait for the job to complete.
-> start –o format=clpxml –w /system1

```
<?xml version="1.0" encoding="UTF-8"?>
<response xmlns="http://www.dmtf.org/smash/2005/01/clpxml"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xsi:schemaLocation="http://www.dmtf.org/smash/2005/01/clpxm
l
smclp_command_response.xsd">
     <command>
          <inputline>start -o format=clpxml -w
/system1</inputline>
     </command>
     <cmdstat>
          <status>3</status>
          <job>
               <job_id>234</job_id>
          </job>
     </cmdstat>
     <start>
     <instance>
          <ufit ufct="system" instance="1">system1</ufit>
          <ufip>/system1</ufip>
     </instance>
          <timestamp>20050130145904.000000-300</timestamp>
     </start>
</response>
```

Start system1 and wait for the job to complete.
-> start –w –o format=clpcsv /system1

```
header,commandline
group,start -w –o format=clpcsv /system1
header,status
group,0
header,command
group,start
header,ufip,timestamp
group,/system1,20050130145904.000000-300
header,endoutput
```

```
Start system1 and wait for the job to complete.
-> start –w –o format=keyword /system1

commandline=start -w –o format=keyword /system1
status=0
command=start
ufip=/system1
timestamp=20050130145904.000000-300
endoutput
```

4.12 stop

The general form of the `stop` command is:

```
    stop [<options>] [<target>]
```

For the `stop` command, implementations MUST support the syntax defined for the `stop-cmd` term in the CLP Grammar defined in Section 4.14.

The `stop` command stops the target. If the target is in a state of running that is equal to or lower than the end state, implementations of the `stop` command MUST return an error and MUST NOT change the target's state.

This command supports usage with and without command line options.

4.12.1 Valid targets

This command is supported when it is specified in a target mapping [CLP-to-CIM Mapping Specification [13]] for a profile which the implementation supports. For all targets that do not support the usage of the `stop` command, implementations MUST NOT show the stop command in a command listing as being available. Implementations of the `stop` command MUST accept an Absolute or a Relative Target Addresse for the command target term.

The behavior of state change commands for each UFcT is defined in the CLP-to-CIM Mapping Specification [13].

4.12.2 Options

The following are valid options for the stop command in addition to those specified in Table 16.

`-f, -force`	Forces the implementation to stop the target, ignoring any policy that might cause the implementation to normally not execute the command. The implementation MUST execute this stop if at all possible, without regard to consequences.

4.12.3 Output

4.12.3.1 Text Format

Implementations MUST return Command Result data which includes the target address that was stopped (if any) and the time and date when the stopped was initiated. Implementations MAY return the time/date information in any applicable format that meets the implementations' needs. If the implementation did not stop the target, the implementation MUST return Command Response data indicating that no targets were stop.

4.12.3.2 Structured Format

Implementations MUST include any status data in the standard format at the top of the response. If the target was successfully stopped, the implementation MUST then return the target that was stopped and the time the stop completed.

4.12.3.2.1 XML Output

The implementation MUST return the `stop` element in the `response` element as defined in the Command Response schema in 6.3.2Appendix B.

```
<stop>
     <ufip> target address of Managed Element to stop </ufip>
     <timestamp>Time stop occurred if completed synchronously,
returned in CIM datetime format</timestamp>
</stop>
```

4.12.3.2.2 CSV

Implementations MUST use the following form when returning Command Results for the `stop` command in “clpcsv” format.

```
header,command
group,stop
header,ufip
group,User Friendly instance path of the Managed Element
targeted by the stop command, Time reset occurred if completed
synchronously, returned in CIM datetime format
header,endoutput
```

4.12.3.2.3 Keyword

Implementations MUST use the following form when returning Command Results for the `stop` command in “keyword” format.

```
command=stop
ufip=User Friendly instance path of the Managed Element
timestamp=Time reset occurred if completed synchronously,
returned in CIM datetime format
endoutput
```

4.12.4 Examples

```
     Send system1 to its default disabled state.
     -> stop /system1
     /system1 stopped at 2:59:04 January 30th 2005

     System1 is successfully stopped
     -> stop –o format=clpxml /system1

     <?xml version="1.0" encoding="UTF-8"?>
     <response xmlns="http://www.dmtf.org/smash/2005/01/clpxml"
     xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
     xsi:schemaLocation="http://www.dmtf.org/smash/2005/01/clpxm
     l
     smclp_command_response.xsd">
          <command>
               <inputline>stop -o format=clpxml
     /system1</inputline>
          </command>
          <cmdstat>
               <status>0</status>
               <job>
                    <job_id>9834</job_id>
               </job>
          </cmdstat>
          <stop>
          <instance>
               <ufit ufct="system" instance="1">system1</ufit>
               <ufip>/system1</ufip>
          </instance>
               <timestamp>20050130145904.000000-300</timestamp>
          </stop>
     </response>
```

system1 is successfully stopped

-> stop –o format=clpcsv /system1

```
header,commandline
group,stop –o format=clpcsv /system1
header,status
group,0
header,job_id
group,9834
header,command
group,stop
header,ufip,timestamp
group,/system1,20050130145904.000000-300
header,endoutput
```

system1 is successfully stopped

-> stop –o format=keyword /system1

```
commandline=stop /system1
status=0
job_id=9834
command=stop
ufip=/system1
timestamp=20050130145904.000000-300
endoutput
```

Examine option is specified.

-> stop -x –o format=clpxml /system1

```
<?xml version="1.0" encoding="UTF-8"?>
```

```
<response xmlns="http://www.dmtf.org/smash/2005/01/clpxml"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xsi:schemaLocation="http://www.dmtf.org/smash/2005/01/clpxm
l
smclp_command_response.xsd">
     <command>
          <inputline>stop -x -o format=clpxml
/system1</inputline>
     </command>
     <cmdstat>
          <status>0</status>
     </cmdstat>
     <stop>
          <examine>
               <text>If run without the examine option,
this will stop system1.</text>
          </examine>
     </stop>
</response>
```

Examine option is specified.
-> stop -x –o format=clpcsv /system1

```
header,commandline
group,stop -x –o format=clpcsv /system1
header,status
group,0
header,job_id
group,job id

header,command
group,stop

header,examine
group,If run without the examine option, this will stop
system1.
header,endoutput
```

Examine option is specified.
-> stop -x –o format=keyword /system1

```
commandline=stop -x –o format=keyword /system1
status=0
job_id=923
command=stop
examine=If run without the examine option, this will stop
system1.
endoutput
```

```
Attempt to stop system1 does not complete synchronously.
-> stop /system1

Stopping system1, job id is 9834.

Attempt to stop system1 does not complete synchronously.
-> stop –o format=clpxml /system1

<?xml version="1.0" encoding="UTF-8"?>
<response xmlns="http://www.dmtf.org/smash/2005/01/clpxml"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xsi:schemaLocation="http://www.dmtf.org/smash/2005/01/clpxm
l
smclp_command_response.xsd">
     <command>
          <inputline>stop -o format=clpxml</inputline>
     </command>
     <cmdstat>
          <status>0</status>
          <job>
               <job_id>9834</job_id>
          </job>
     </cmdstat>
     <stop>
     <instance>
          <ufit ufct="system" instance="1">system1</ufit>
          <ufip>/system1</ufip>
     </instance>
     </stop>
</response>

Attempt to stop system1 does not complete synchronously.
-> stop –o format=clpcsv /system1

header,commandline
group,stop –o format=clpcsv /system1
header,status
group,0
header,job_id
group,9834
header,command
group,stop
header,ufip
group,/system1
header,endoutput

Attempt to stop system1 does not complete synchronously.
```

```
-> stop –o format=keyword /system1

commandline=stop –o format=keyword /system1
status=1
job_id=9834
command=stop
ufip=/system1
endoutput
```

4.13 version

The general form of the `version` command is:

`version [<options>]`

For the `version` command, implementations MUST support the syntax defined for the `version-cmd` term in the CLP Grammar defined in Section 4.14.

The `version` command is used to display the version of the SM CLP protocol which this implementation supports. The implementation MUST return the version of the SM CLP protocol with which it is compatible.

4.13.1 Valid Targets

The `version` command does not accept targets. Implementations MUST NOT accept non-option terms for the `version` command. If the implementation receives a command with non-option terms specified, the implementation MUST return Command Status of COMMAND PROCESSING ERROR and a Processing Error of COMMAND SYNTAX ERROR.

4.13.2 Options

Valid options for the `version` command are specified in Table 16 Verb and Option Support.

4.13.3 Output

4.13.3.1 Text Format

The implementation MUST include the string "**`Version 1.0a`**" clearly identified as the CLP Protocol version supported by the implementation.

4.13.3.2 Structured Format

The returned data MUST include any status data in the standard format at the top of the response.

4.13.3.2.1 XML Output

The implementation MUST return the `version` element in the `response` element as defined in the Command Response schema in 6.3.2Appendix B. The implementation MUST return the exact string "**`Version 1.0a`**" as the data contained in the `text` element within the `version` element.

4.13.3.2.2 CSV

When returning Command Results for the `version` command in "clpcsv" format, implementations MUST return exactly the following table:

header,command

group,version

header,text

group,Version 1.0.0

4.13.3.2.3 Keyword

Implementations MUST use the following form when returning Command Results for the `cd` command in "keyword" format.

```
command=version
text=Version 1.0.0
endoutput
```

4.13.4 Examples

```
Run the version command the way its meant to be.
-> version
Version 1.0.0

The version command does not take targets.
-> version /system1
Invalid Syntax, the version command does not support a
target.

The version command is specified with invalid syntax.
-> version –o format=clpxml /system1

<?xml version="1.0" encoding="UTF-8"?>
<response xmlns="http://www.dmtf.org/smash/2005/01/clpxml"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xsi:schemaLocation="http://www.dmtf.org/smash/2005/01/clpxm
l
smclp_command_response.xsd">
     <command>
          <inputline>version -o format=clpxml
/system1</inputline>
     </command>
     <cmdstat>
          <status>2</status>
          <error>252</error>
          <error_tag>COMMAND SYNTAX ERROR</error_tag>
     </cmdstat>
     <version></version>
```

```
 </response>

The version command is successful.
-> version –o format=clpxml
<?xml version="1.0" encoding="UTF-8"?>
<response xmlns="http://www.dmtf.org/smash/2005/01/clpxml"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xsi:schemaLocation="http://www.dmtf.org/smash/2005/01/clpxml
smclp_command_response.xsd">
 <command>
      <inputline>version -o format=clpxml</inputline>
 </command>
 <cmdstat>
      <status>0</status>
      <job>
           <job_id>930</job_id>
      </job>
 </cmdstat>
 <version>
           <text>Version 1.0.0</text>
 </version>
</response>

The version command is successful.
-> version –o format=clpcsv

header,commandline
group,version –o format=clpcsv
header,status
group,0
header,job_id
group,0
header,command
group,version
header,text
group,Version 1.0.0

The version command is successful.
-> version –o format=keyword

command=version –o format=keyword
text=Version 1.0.0
endoutput
```

4.14 Command Grammar

SM CLP Grammar in ABNF Notation (RFC2234) V0.7a

```
;;

;;    Note that line joining is done before tokenization, the "`" (backquote)
character at the

;;    end of the line indicates that the current line is continued the
following line.

;;

;;    WSP, CR, LF, CRLF, SP, DIGIT, ALPHA, DQUOTE, VCHAR are part of core
rules

;;    defined in RFC2234.

;;

tsep  = WSP                   ; command term separator

eol   = CR / LF / CRLF        ; end of line token

dot   = "."

dotdot = ".."

splat = "*"

addrTermSeparator = “/” / “\”  ; address term separator


;;

;; SM CLP command set:

;;

;;    General command grammar rules:

;;          verb: always be the 1st term on the command line.

;;          target: appears as the 1st term that is not an option.

;;          options: recognized by the option indication "-", appear after
verb.

;;          properties: appear after target.

;;

;;          Example: verb [options] [target] [properties]

;;

commands    = cd-cmd

            / create-cmd

            / delete-cmd
            / dump-cmd

            / exit-cmd

            / help-cmd

            / load-cmd

            / reset-cmd

            / set-cmd

            / show-cmd

            / start-cmd

            / stop-cmd

            / version-cmd

            / OEM-cmd


;;

;; SM CLP command verb set:

;;

verb        = cd-verb

            / create-verb

            / delete-verb

            / dump-verb

            / exit-verb

            / help-verb

            / load-verb

            / reset-verb

            / set-verb

            / show-verb

            / start-verb

            / stop-verb

            / version-verb



;;
```

```
;; SM CLP command options:

;;

;;    General option grammar rules:

;;          option = "-" optName [optArg]

;;          options = option *(tsep option)

;;

Options     = all-option

            / default-option

            / destination-option

            / display-option

            / examine-option

            / force-option

            / help-option

            / keep-option

            / level-option

            / output-option

            / resetstate-option

            / source-option

            / version-option

            / wait-option

            / OEM-options


;;

;; SM CLP command properties:

;;

property         = tsep propertyName

properties       = 1*(tsep propertyName)

propertyValue    = 1*VCHAR / quoted-string     ; visible char

propertyValues   = (“” / propertyValue) *("," (“” / propertyValue))  ; for
array data type

quoted-string    = DQUOTE *(qdtext / ("`" DQUOTE)) DQUOTE
```

```
qdtext           = %x21 / %x23-7E              ; any printable char except
DQUOTE

property-assignValue    = propertyName "=" propertyValues

property-selectValue    = propertyName "==" propertyValue

properties-assignValues = 1*(tsep property-assignValue)




;;

;; Detail option rule definitions:

;;    Note that <ruleName> denotes an element referenced in this document:

;;          1. <URI> is defined in RFC2396.

;;          2. <UFcT>, <UFiT>, and <UFiP> are defined by the SM ME Addressing
Specification.

;;          3. <propertyName> is defined in Appendix A of CIM Specification
V2.2.

;;          4. <vendor> is a property in CIM_Product class.

;;

all-option  = tsep all-optionName

default-option = tsep default-optionName

destination-option = tsep destination-optionName tsep (<URI> / <UFiP>)

display-option = tsep display-optionName tsep display-optionArgs

                              *("," display-optionArgs)



display-optionPropArgProperty = (propertyName / property-selectValue)

display-optionArgs = *( "all" / "verbs"
                  /  display-optionAssocArg
                  /  display-optionTargetsArg
                  /  display-optionPropArg

display-optionAssocArg =
  "associations" ["=" (className / (“(“(className *("," className))“)”)) ]

display-optionTargetsArg =
  "targets" ["=" ( target-UFsT  / (“(“target-UFsT *("," target-UFsT)”)”)) ]

display-optionPropArg =
  "properties" ["=" display-optionPropArgProperty /
  ( “(“ display-optionPropArgProperty
  *("," display-optionPropArgProperty ) “)” ) ]
```

```
examine-option = tsep examine-optionName

force-option = tsep force-optionName

help-option = tsep help-optionName

keep-option = tsep keep-optionName tsep 1*DIGIT ["." 1*DIGIT]

level-option = tsep level-optionName tsep (1*DIGIT / "all")

output-option = tsep output-optionName tsep output-optionArgs *("," output-
optionArgs)

output-optionFormatArg = ("text" / “clpcsv” / "keyword" / “clpxml”)

output-optionArgs = *( ("format" "="

    output-optionFormatArg / ( “(“ output-optionFormatArg ”)” ) )

                  / ("error" / "terse" / "verbose")

                  / ("language" "=" 3ALPHA / ( “(“ 3ALPHA ”)” ) )

                  / ("begin" / "end")

                  / ("order" "=" ("default" / "reverse") / ( “(“ "default" /
"reverse" ”)” ))

                  / ("number" "=" (1*DIGIT "-" 1*DIGIT) / ( “(“ (1*DIGIT "-"
1*DIGIT) ”)” ))

                  / ("count" "=" ("all" / 1*DIGIT)) / ( “(“ ("all" / 1*DIGIT)
”)” ))


resetstate-option = tsep resetstate-optionName tsep stateValue

source-option = tsep source-optionName tsep (<URI> / <UFiP>)

version-option = tsep version-optionName

wait-option = tsep wait-optionName

stateValue  = propertyValue


;;

;; Common options applicable to all verbs:

;;

common-options    = *(examine-option / help-option / keep-option / output-
options /

                        version-option / OEM-options)
```

```
;;

;; Command target term formations:

;;

target-Instance   = tsep targetPath

                  / tsep OEM-target

                  / tsep "SESSION"

target-UFsT       = tsep [addrTermSeparator] [addrTerm *(addrTermSeparator
addrTerm) addrTermSeparator] <UFcT> splat ; UFcT*

target-Assoc      = tsep targetPath-Instance "=>" className ["=>"  <UFiP>]

                              ; className is defined in MOF syntax grammar

targetPath        = [addrTermSeparator] [addrTerm *(addrTermSeparator
addrTerm)]

addrTerm          = (dot / dotdot / <UFiT>)


;;

;; Per-command formations:

;;

cd-cmd      = cd-verb [cd-options] [target-Instance] eol

cd-options  = *(default-option / wait-option / common-options)


create-cmd  = create-verb [create-options] (target-Instance / target-UFsT)

                        [properties-assignValues] eol

create-options = *(default-option / wait-option / common-options)


delete-cmd  = delete-verb [delete-options] [target-Instance / target-UFsT]
eol

delete-options = *(force-option / wait-option / common-options)


dump-cmd    = dump-verb [dump-options] destination-option [dump-options]
[target-Instance] eol

dump-options = *(force-option / wait-option / common-options)


exit-cmd    = exit-verb [exit-options] eol
```

```
exit-options = common-options


help-cmd    = help-verb [help-options] [verb]

            / help-verb [help-options] [target-Instance / target-UFsT /
target-Assoc / <UFcT> ] [properties] eol

help-options = *(wait-option / common-options)


load-cmd    = load-verb [load-options] source-option [load-options] [target-
Instance] eol

load-options = *(force-option / wait-option / common-options)


reset-cmd   = reset-verb [reset-options] [target-Instance] eol

reset-options = *(force-option / resetstate-option /

                  wait-option / common-options)


set-cmd     = set-verb [set-options] [target-Instance / target-Assoc]
properties-assignValues eol

set-options = *(default-option / force-option / wait-option / common-options)



show-cmd    = show-verb [show-options] [target-Instance / target-UFsT /
target-Assoc] eol

show-options = *(all-option / default-option / display-option /

                  force-option / level-option / wait-option / common-options)


start-cmd   = start-verb [start-options] [target-Instance] eol

start-options = *(force-option / wait-option /

                  common-options)


stop-cmd    = stop-verb  [stop-options] [target-Instance] eol

stop-options = *(force-option / wait-option /

                  common-options)
```

```
version-cmd = version-verb [version-options] eol

version-options = *(wait-option / common-options)


;;

;; OEM syntax:

;;

OEM-cmd           = "OEM"<vendor><vendor-specified command line syntax> eol

OEM-target        = tsep "OEM"<vendor><vendor-specified targetAddress>

OEM-options       = tsep *(OEM-optionName [tsep OEM-optionArgs])

OEM-optionName    = "-OEM"<vendor><vendor-specified option Name>

OEM-optionArgs    = "OEM"<vendor><vendor-specified option arguments>

OEM-propertyName  = tsep "OEM"<vendor><vendor-specified property name>


;;

;; Verb names:

;;

cd-verb     = "cd"                  ; Note that ABNF strings are case-
insensitive

create-verb = "create"

delete-verb = "delete"

dump-verb   = "dump"

exit-verb   = "exit"

help-verb   = "help"

load-verb   = "load"

reset-verb  = "reset"

set-verb    = "set"

show-verb   = "show"

start-verb  = "start"

stop-verb   = "stop"

version-verb= "version"
```

```
;;
;; Option names:
;;
all-optionName          = "-a" ["ll"]
default-optionName      = "-default"
destination-optionName  = "-destination"
display-optionName      = "-d” [“isplay"]
examine-optionName      = "-x" / "-examine"
force-optionName        = "-f" ["orce"]
help-optionName         = "-h" ["elp"]
keep-optionName         = "-k" ["eep"]
level-optionName        = "-l" ["evel"]
output-optionName       = "-o" ["utput"]
resetstate-optionName   = "-resetstate"
source-optionName       = "-source"
version-optionName      = "-v" ["ersion"]
wait-optionName         = "-w" ["ait"]
```

5 Standard Command Options

Every CLP option is defined to have the same name and behavior across the CLP command verb set. For each CLP verb, implementations MUST recognize the CLP standard options applicable to the verb and observe the specified behavior of each option in a manner consistent with the option definition. E.g. Use of "-o" must always indicate the `output` option whenever used and must follow the behavior specified for the `output` standard option. The sections below describe each standard option and the behavior of each.

Implementations MUST recognize option names both by the full expression of the option name and by a single letter short form of the option name if one is defined by this specification. Implementations MUST NOT recognize option names of any other length (ie. "partial" or "two letter" option names). For example, the option name for controlling output must be recognized by its full option name form, "-output", and by its single character short form, "-o", and not by any other form, such as "-out".

For each CLP command processed, implementations MUST observe the default setting of standard options unless the user overrides the setting with a session default or by using the standard option explicitly in the command line entered. The order of precedence for option settings is specification default, superseded by the session default, and finally superseded by the explicit command line setting, if present.

Implementations MUST observe the following order of precedence for standard options: `help`, `version`, `examine`, followed by all others. If the `help` option appears anywhere on the command line implementations MUST provide the behavior specified in Section 5.7. If the `version` option appears, but the `help` option does not, implementations MUST provide the behavior specified in Section 5.11. If the `examine` option appears on the command line without either the `help` or `version` option, the implementation MUST provide the behavior described in Section 5.5.

If an option is included more than once in a command, the implementation MUST NOT execute the command. The implementation MUST return Command Response data indicating a COMMAND SYNTAX ERROR.

CLP options either always require an argument or never require an argument. If a CLP option which requires an argument is specified in a command without an argument value, the implementation MUST return Command Status of COMMAND PROCESSING FAILED and a Processing Error of MISSING ARGUMENT.

Implementations MUST only support those options listed in Table 15 Standard Command Options. Implementations MUST support all of the options listed in Table 15 Standard Command Options. Implementations MUST support arguments on all options which specify an argument in Table 15 Standard Command Options Implementations MUST NOT support arguments for those options which do not specify an argument in Table 15 Standard Command Options

For each option listed in Table 15 Standard Command Options which has a value of `yes` in the column labeled *Session Default Supported*, the implementation MUST allow the user to set a default value per session. For each option listed in Table 15 Standard Command Options which does not have a value of `yes` in the column labeled *Session Default Supported*, the implementation MUST NOT allow the user to set a default value per session. Each option with a supported session default value is modeled with properties on the CIM class which represents the CLP session. For information on the CIM class and its properties, see the MAP Profile.

Table 15 Standard Command Options

Option Name	Short Form	Interpretation	Argument	Session Default Supported
`-all`	`-a`	Instructs the verb to perform all of its possible functions.	NONE	
`-default`	<none>	Instructs the verb to select default values. The exact interpretation is dependent on the verb this option is used with.	NONE	
`-destination` *<URI>*	<none>	Indicates the location of a destination for an image or other target data.	URI or SM instance address	
`-display`	`-d`	Selects the data the user wants to display.	List of property names or reserved strings identifying the desired data	
`-examine`	`-x`	Instructs Command Processor to examine the command for syntax and semantic errors but not execute the command.	NONE	
`-force`	`-f`	Instructs verb to ignore warning conditions that would prevent execution by default.	NONE	
`-help`	`-h`	Displays documentation about the command verb.	NONE	
`-keep` *<m[.s]>*	`-k`	Establishes a holding time for command job ID and status.	Amount of time to hold command job ID, status.	Yes
`-level` *<n>*	`-l`	Instructs the Command Processor to execute the command for the current target plus targets contained through the specified level of depth.	Number of levels expressed as a natural number or “all”	
`-output` *<args>*	`-o`	Controls the content and form of the command output.	Many arguments providing control of format, language, level of detail, order, etc. of output data	Yes
`-resetstate`	<none>	Indicates what target specific state to take the target down to.	Target specific state identifier	
`-source` *<URI>*	<none>	Indicates the location of a source image or target.	URI or SM instance address	
`-version`	`-v`	Displays the version of the command verb.	NONE	
`-wait`	`-w`	Instructs the Command Processor to hold Command Response until all spawned jobs have	NONE	Yes

Option Name	Short Form	Interpretation	Argument	Session Default Supported
		completed.		

The following tables specify the requirements for implementations to support each combination of verb and standard option.

Table 16 Verb and Option Support

	cd	create	delete	dump	exit	help	load	reset	set	show	start	stop	version
all										M			
default	M	M							M	M			
destination				M									
display										M			
examine	M	M	M	M	M	M	M	M	M	M	M	M	M
force			S	m			m	S	S	m	S	S	
help	M	M	M	M	M	M	M	M	M	M	M	M	M
keep	M	M	M	M	M	M	M	M	M	M	M	M	M
level										M			
output	M	M	M	M	M	M	M	M	M	M	M	M	M
resetstate								S					
source							M						
state								S			S	S	
version	M	M	M	M	M	M	M	M	M	M	M	M	M
wait	M	M	M	M		M	M	M	M	M	M	M	M

Legend for Table 16 Verb and Option Support

M – If the cell in Table 16 contains an ‘M’, the implementation MUST support usage of the option named in the row with verb named in the column.

m – If the cell in Table 16 contains an ‘m’, the implementation MAY support usage of the option named in the row with verb named in the column.

S – If the cell in Table 16 contains an ‘S’, the implementation SHOULD support usage of the option named in the row with verb named in the column.

If the cell in Table 16 is blank, the implementation MUST NOT support the usage of the option named in the row with the verb named in the column.

5.1 All

Forms
`-all`

```
-a
```

Example use

```
show –a log1
```

Behavior

The `all` option instructs the Command Processor to select all values where appropriate for the command. This option is only supported with the `show` command. For a detailed description of the usage, see Section 4.10.2.

When the `all` option is specified, the implementation will select all values where appropriate for the command.

5.2 Default

Forms

```
-default
```

Example use

```
show –default log1
```

Behavior

The interpretation of the Default option is dependent on the verb with which it is used. For verbs where this option is supported, the behavior is described in the Options section for the verb.

5.3 Destination

Forms

```
-destination
```

Example use

```
dump –destination
ftp://adminstrator:passw0rd@myserver.com/private/administrator/`
memory.dmp /system1/memory1
```

Behavior

The `destination` option is used to specify a destination for the transfer of a binary image. The desired destination is specified as the argument to the `destination` option. The destination may be expressed as a UFiP if it is a Managed Element in the address space of the MAP. The destination may also be expressed as a URI. The desired protocol to use for transferring the image may be specified as part of the URI. This specification does not mandate support for a specific protocol. Thus the protocols supported is implementation specific.

5.4 Display

Forms

```
-d[isplay] <type>*[,<type>]
```

Example use

Show the commands that are supported for use with log1 as a target

```
show –display verbs log1.
```

Show all of the systems in the address space, where one of the values of the Dedicated property is “Router”. No filtering of the results for individual instances is requested.

```
show –level all –display
     properties=dedicated==Router,targets=system /
```

Show all of the cpus and SystemDevice association instances in system1. No filtering of the results for individual instances.

```
show –level all –display targets=cpu,associations=systemdevice
     /system1
```

Show the operationalstatus property for every power supply and fan in system1.

```
show –level all –display
     targets=cpu:powersup,properties=operationalstatus
```

Show the elementname property for every instance that has a bad operationalstatus.

```
show –level all –display
     properties=elementname:operationalstatus==error /
```

Behavior

The `display` option filters the information returned in Command Results. This option provides two levels of filtering. It enables filtering of results for entire instances at the granularity of CIM class or at individual property/value granularity. The `display` option also enables filtering results for an individual instance, allowing a user to restrict the results to the properties (or a subset) of the instance and/or the verbs (or a subset) supported for an instance. For any two commands which are identical except for arguments to the `display` option, the implementation MUST interpret and execute the command such that the set of Managed Elements affected, and the resultant state of those Managed Elements, is identical. The `display` option requires one or more arguments specifying the category of information to include in the Command Results. If the `display` option is specified, the implementation MUST NOT include data in the Command Results returned to the Client unless it is specified by an argument included with the `display` option.

The valid types and formats for the arguments of the Display option are

`verbs` Display the commands that are valid for this target.

`properties[=”(“([name==value,]<name>,<name>,...,<name>)”)”]`

Can be used to filter the Command Results such that information about an instance is returned only if the instance has a property with the specified value or a property with the specified name. Filters the Command Results such that only those properties specified by name are returned. The properties are returned in the order indicated. If no property names are specified, all properties included in the Command Results will be returned.

`targets[="("(UFcT][,<UFcT>,<UFcT>,...,<UFcT>")"]`

Filter the Command Results to only include information about Managed Element instances.

`associations[="("(classname][,<classname>,<classname>,...,<classname>")"]`

Filter the Command Results to only include information about associations.

The `display` option acts as a filter on output of a command. The results of a command are marshaled into operation results data. Arguments to the `display` option are used to select which information is included in the Command Results data. Arguments to the display option are applied in the following order:

- `targets (instance selection by class)`
- `associations (instance selection by class)`
- `properties (propname==propvalue, instance selection by property value)`
- `properties (propname, instance selection by property presence)`
- `properties (propname, restrict properties that are returned for an instance)`
- `verbs`

When the display option is specified with one or more arguments, the implementation MUST apply each of the arguments to the operation results according to the rules for each argument. Implementations MUST apply the arguments (if present) in the following order: targets, associations, properties, verbs. When the diplay option is specified with an argument of `all`, the implementation MUST interpret the display option as if it was specified with an argument of "`targets,associations,verbs,properties`". Prior to processing the argument values, the implementation MUST remove the leading and trailing parenthesis if they are present.

When the `display` option is specified with an argument of `targets` and no value is supplied to the argument, the implementation MUST filter the operation results and add to the Command Results information about Managed Element instances. If the `display` option is specified with an argument of `targets` and a value is supplied to the `targets` argument, the implementation MUST interpret the supplied value as a "," delimited, ordered list of UFcTs. The implementation MUST apply the ordered list of UFcTs as a filter on the instances for which results are returned. The implementation MUST add operation results for an instance if and only if the instance is of one of the types specified by a UFcT in the list. The implementation MUST add the results for instances in the order their corresponding types were specified in the UFcT list.

When the display option is specified with an argument of `associations` and no value is supplied to the argument, the implementation MUST filter the operation results and add to the Command Results information about each association. If the `display` option is specified with an argument of `associations` and a value is supplied to the `associations` argument, the implementation MUST interpret the supplied value as a "," delimited, ordered list of Association Classes. The implementation MUST apply the ordered list of Association Classes as a filter on the instances for which results are returned. The implementation MUST add operation results for an instance if and only if the instance is of one of the types specified by an Association Class in the list. The implementation MUST add the results for instances in the order their corresponding types were specified in the list of class names.

There are three uses for the `properties` argument to the display option. Usage is supported with or without additional values. Implementations MUST apply the filtering of the `properties` argument to the intermediate results that are generated by applying the filtering of the `association` and `targets` arguments to the operation results. If the `display` option is specified with an argument of `properties` and a value is supplied to the `properties` argument, the implementation MUST interpret the value as a "," delimited, ordered list of tokens.

If a token contains the unescaped character sequence "`==`", the implementation MUST interpret the token as a `propertyname==propertyvalue` pair. If a `propertyname==propertyvalue` pair is specified, the implementation MUST filter the set of instances for which results will be returned such that the instance will be returned if and only if the instance has a property named `propertyname` and the value of that property equals `propertyvalue`. The rules for determining equivalency of two values are property specific and defined in the CLP-to-CIM Mapping Specification [13]. If `propertyname` identifies a property which is an array, the implementation MUST search the entire array for a value which is equivalent to `propertyvalue`. If at least one equivalent value is found, the implementation MUST interpret this as a match and return information about the instance. The `propertyname==propertyvalue` syntax is only supported on the `show` command. If the `propertyname==propertyvalue` syntax is specified for a command other than `show`, the implementation MUST NOT execute the command and MUST return a Command Status of COMMAND PROCESSING FAILED and a Processing Error of INVALID OPTION. If there is more than one `propertyname==propertyvalue` pair specified, the implementation MUST NOT execute the command and MUST return a Command Status of COMMAND PROCESSING FAILED and an error of INVALID ARGUMENT.

If the token does not contain the character sequence "==" unescaped, the implementation MUST interpret the token as a property name to apply as a filter to the operation results. The implementation MUST apply the ordered list of property names as a filter after any filtering based on a `propertyname==propertyvalue` pair. The implementation MUST filter the set of instances for which results will be returned such that the instance will be returned if and only if the instance has a property named `propertyname`. The implementation MUST include information about a property of an instance, if the property name appears in the ordered list of property names that were specified as the value of the `properties` argument.

When the `display` option is specified with an argument of `properties` and no value is supplied to the argument, the implementation MUST filter the intermediate results and add to the Command Results for an instance, information about the properties of the instance.

When the `display` option is specified with an argument of `verbs`, the implementation MUST filter the intermediate results and add to the Command Results for an instance information about the verbs that are supported for the instance.

Its possible that applying the filters on information returned for an instance will result in there being no information to return for an instance. After applying the filters, if there is no information for an instance, the implementation MUST remove the instance from the Command Results that are returned.

5.5 Examine

Forms

```
B. -examine
-x
```

Example use

```
stop –force –x /system3
```

Behavior

The `examine` option causes the Command Processor to examine command syntax only and provide feedback to the user on the command's validity and correctness. The Command Processor MUST NOT commit or execute the command. The Command Processor MUST verify that the options, target path, and target properties are valid. The command output MAY describe what action would be taken, if the command were to be entered without the `examine` option. While the Command Processor will not execute the command, applying the examine option to the Command Line is itself an activity of the MAP which will result in a Job being created.

5.6 Force

Forms

```
-force
-f
```

Example use

```
stop –f /system3/blade2
```

Behavior

The `force` option causes the Command Processor to ensure that the command executes, regardless of potential preparation steps that could have been taken, initial ME states recommended, or exceptions that may occur during execution. The `force` option does not override authorization of a user to execute a command for a given target.

When the `force` option is specified, the implementation MUST execute the command.

5.7 Help

Forms
```
-help
-h
```

Example use
```
stop –help
```

Behavior

The `help` option causes the Command Processor to return text describing the proper use of the verb entered as the first term on the command line. The help text includes a description of the verb and it's behavior and descriptions of all options supported by the verb. When the `help` option is specified, the command does not execute or cause any change to the target. It does not alter the state or properties of a target that may appear on the same command line. Help output text is governed by the language option currently in effect.

When the `help` option is specified, the implementation MUST NOT take the action specified by the verb. The implementation MUST return text describing the proper use of the verb entered as the first term on the command line.

5.8 Keep

Forms
```
-keep <m[.s]>
-k <m[.s]>
```
- where 'm.s:' is an amount of time specified as 'minutes.seconds'

Example use

`reset –k 10 system1` Resets 'system1' and retains the Command Status for 10 minutes

Behavior

The `keep` option is used to request the Command Processor to retain status information for the job spawned in response to the command with which the option was included. In order to use the `keep` option to request that status for a command be preserved, it must be included as part of the initial command request.

Implementations MUST keep the CIM_ConcreteJob instance corresponding to the job in the job queue after the job is completed for the time specified by the `keep` option. The implementation MUST set the TimeBeforeRemoval property of the CIM_ConcreteJob instance to the value specified by the `keep` option. Implementations are NOT REQUIRED to maintain Command Results after the command execution completes even if the `keep` option is utilized. The `keep` option only controls how long the implementation is required to maintain the Command Status.

If the CLP session has been ended before the command has completed and Command Status has been returned, the Command Status will be retained either for the amount of time that was specified with the `keep` option or the session default as appropriate.

5.9 Level

Forms

`-level <`*`n`*`>`

`-l <`*`n`*`>`

- where 'n' is the number of levels to include in command scope

Example use

`show –l 2`	Show info about default target and one level of contained MEs
`show –l 3`	Shows info about MEs up to 2 levels deep
`stop –l all`	Stop all contained MEs, then stop command target
`show –l all system3`	Show info about "`system3`" and all contained MEs

Behavior

The `level` option instructs the command verb to include 'n' number of levels in the scope of its execution. A level typically refers to the depth of containment that is to be processed by the verb, following the target addressing syntax described in Section 4.2.3.3.

Any levels specified that extend beyond the containment depth of the command target MUST be ignored. For example, if a command target contains only one level of contained targets and the user attempts to show "n=5" levels, the command is still executed successfully and the one level of containment is returned in the output.

The value of "n" is interpreted as follows:

n=1	Verb is interpreted for the command target only (default)
n=2	Verb acts on the command target and any directly-contained MEs
n=3	Verb acts on the command target, directly-contained MEs, and any MEs contained by those MEs (i.e. Current target and "two down").
n=all	Verb acts on the command target and all target MEs recursively contained in the command target

When the Level option is included with a command, the implementation MUST apply the command to each level of containment up to the minimum of: the level of containment specified by the argument to the level option and the actual containment depth of the command target.

5.10 Output

Forms

`-output <`*`arguments`*`>`

`-o <`*`arguments`*`>`

Table 17 Output Option Arguments

Argument	Value domain (default in **bold**)	Description
`format=<`*`value`*`>`	**`"text"`**, `"clpcsv"`, `"keyword","clpxml"`	Format controls the structure of the output text.
`error`, **`terse`**, `verbose`		Selects the level of detail included in the output.
`language=<`*`value`*`>`	`3-character string identifier of language as specified in ISO 639.2;` **`"eng" is default`**	Language selects the translation of text.
`begin`, `end`		When multiple items are returned in the output, begin/end control where to start in the list.
`order=<`*`value`*`>`	**`"default"`**, `"reverse"`	When multiple items are returned in the output, order controls the order of those items.
`count=<`*`value`*`>`	`<integer string or` **`"all"`**`>`	When multiple items are returned in the output, number controls the number of items returned (e.g. log items); the default is **'all'** items. Maximum value for "number" is determined by the class of the target.
`number=<x-y>`	`[integer string]-[integer string]`	Requests that a range of results be returned.

Example use

```
show –output format=clpcsv,verbose,order=reverse,number=5 /system2/log

show –o verbose /system2/log –display properties=(recordnumber,time,text)

start -o quiet system1
```

Behavior

The `output` option controls the format of output returned by the Command Processor to the Client. This option has no effect on the execution of the command or the Managed Elements affected by the command. For any two commands which are identical except for arguments to the output option, the implementation MUST interpret and execute the command such that the set of Managed Elements affected, and the resultant state of those Managed Elements, is identical. For example, if an implementation receives a command:

```
delete –output count=4 UFcT
```

The implementation would delete all instances of the UFcT in the current container, not just four of them. Prior to processing the value for an argument, the implementation MUST remove the leading and trailing parenthesis if they are present.

By default, all output is returned in free-form English text, which is not suitable for parsing. The user MAY request that output be formatted in a structured form by using the "format" argument and an associated value representing the desired structure. For example, if the user desires the output to be returned in an XML document format, the option form "-output format=clpxml" is used.

If the `output` option with the `format` argument is included in a command, the implementation MUST attempt to interpret the value of the `format` argument as one of the CLP output format identifiers. If the value of the `format` argument is a CLP output format identifier and the implementation supports the indicated output format, the implementation MUST return Command Response data compliant with the format specified. If the value of the `format` argument is a CLP output format identifier and the implementation does not support the indicated output format the implementation MUST NOT execute the command and the implementation MUST return Command Response data compliant with the session default output format indicating OUTPUT FORMAT NOT SUPPORTED. If the argument value is not a CLP output format identifier, the implementation MUST NOT execute the command and the implementation MUST return Command Response data compliant with session default output format indicating a Processing Error of INVALID ARGUMENT.

Implementations MUST recognize the string `text` as identifying the CLP text output format and MUST NOT recognize any other string as identifying the text output format. Implementations MUST recognize the string `clpcsv` as identifying the CLP comma separated value output format and MUST NOT recognize any other string as identifying the comma separated value output format. Implementations MUST recognize the string `keyword` as identifying the CLP keyword/value output format and MUST NOT recognize any other string as identifying the keyword/value output format. Implementations MUST recognize the string `clpxml` as identifying the CLP text output format and MUST NOT recognize any other string as identifying the XML output format.

If the `output` option with the `language` argument is included in a command, the implementation MUST attempt to interpret the value of the `language` argument as a language code defined in accordance with ISO 639-2 [24]. If the implementation recognizes the value of the language argument as identifying an output language it supports, the implementation MUST return Command Response data in the language specified. If the value of the language argument is not recognized by the implementation as identifying an output language it supports, the implementation MUST NOT execute the command, the implementation MUST return Command Response data in the session default language indicating a Processing Error of INVALID ARGUMENT.

The `begin`, `end`, `number`, `order`, and `count` arguments to the `output` option affect the order or number of results displayed. Implementations MUST apply these filters across results for Managed Elements and MUST NOT apply them within the result for an individual Managed Element. The `begin`, `end`, and `number` arguments are mutually exclusive. If the implementation receives a command with the `output` option specified with more than one of `begin`, `end`, and `number` specified, the implementation MUST NOT execute the command

and MUST return Command Status of COMMAND PROCESSING FAILED with Processing Error COMMAND SYNTAX ERROR. The `begin`, `end`, and `order` arguments are mutually exclusive. If the implementation receives a command with the `output` option specified with more than one of `begin`, `end`, and `order` specified, the implementation MUST NOT execute the command and MUST return Command Status of COMMAND PROCESSING FAILED with Processing Error COMMAND SYNTAX ERROR

For each UFcT, the CLP-to-CIM Mapping Specification [13] defines the natural (default) order in which results will be returned. The `begin`, `end,` and `order` arguments allow a user to modify the order in which results are returned for a command. When the `output` option is specified with an argument of `begin`, the implementation MUST select the first element that would be returned according to the natural order of results as the starting point for returning results. When the `output` option is specified with an argument of `end`, the implementation MUST select the last element that would be returned according to the natural order of results as the starting point for returning results for the current command. When the `output` option is specified with an argument of `order` and the value selected for `order` is `default,` the implementation MUST return results in the natural order for the class. When the `output` option is specified with an argument of `order` and the value selected for `order` is `reverse,` the implementation MUST return results in reverse of the natural order for the class. If the `end` argument is not specified, the implementation MUST behave as if the `begin` argument was specified. If the `order=reverse` argument is not specified, the implementation MUST behave as if `order=default` was specified. Thus the default behavior is to return elements according to their natural order.

For example, assume there are ten instances of a class where the natural ordering of their results is UFcT*1*, UFcT*2*, …UFcT*10.* If `end` is the only argument to the `output` option, the implementation will select instance UFcT*10* as the starting point for returning results. Following the natural order of results, there are no other elements after UFcT*10*. Hence the implementation will return UFcT*10* and nothing else. Similiarly, if `order=reverse` is the only argument specified to the `–output` option, the implementation will select UFcT*1* as the starting point for returning elements. Following the reverse of the natural order, the implementation will not find any additional instances occurring prior to UFcT*1*. Hence the instrumentation will return UFcT*1* only and no other elements. Thus, the only way to view the set of elements in reverse order is to specify both the `end` and `–order=reverse` arguments to the `output` option.

The `count` argument allows a user to restrict the number of Managed Elements for which results are displayed. When the `output` option is specified with an argument of `count`, the implementation MUST NOT return results for more Managed Elements than the value specified for `count`.

The number argument allows a user to select a range of Managed Elements. The value of the number argument is in the format:

```
<starting identifier>-<ending identifier>
```

Range selection is inclusive of the start and ending identifier.

The selection of the range by the implementation is class-specific. Implementations MUST follow the rules for range selection specified in CLP-to-CIM Mapping Specification [13]. When the output option is specified with an argument of `number`, the implementation MUST select the range of instances identified by `starting identifier`, through the `ending identifier`, inclusive.

The `terse`, `verbose`, and `error` arguments are used to specify the level of detail an implementation should return when the output format is “text”. If the implementation receives a command with the `output` option specified with more than one of `terse`, `error`, and `verbose` specified, the implementation MUST NOT execute the command and MUST return Command Status of COMMAND PROCESSING FAILED with Processing Error COMMAND SYNTAX ERROR.

When the output format for a command is “text” and the `output` option is specified with an argument of `terse`, the implementation MUST return Command Status which only includes the Status Tag data element, unless a processing error or execution error occurs, in which case the implementation MUST return the Processing Error Tag or Job Error data element as appropriate to the error which occurred.

When the output format for a command is “text” and the `output` option is specified with an argument of `error`, the implementation MUST return Command Status if an error occurs processing the command and MUST NOT return Command Status if an error does not occur processing the command.

When the output format for a command is “text” and the `output` option is specified with an argument of `verbose`, the implementation MUST return Command Status which includes the Status Tag. If a processing error or execution error occurs the implementation MUST return the Processing Error Tag or Job Error data element as appropriate to the error which occurred. The implementation MAY return Message data elements.

Table 18 Using Output option to control Command Status Output

Option	Description / Uses
`-o terse`	A short description of the status is returned.
`-o verbose`	Command Status and Command Results are returned.
`-o error`	Command Status is only returned if an error occurs.

5.11 Resetstate

Forms
`–resetstate`

Example use
`reset –resetstate disabled /system1/service3`

Behavior
The `resetstate` option is used to specify the desired state to take a target Managed Element down to in the course of executing the `reset` command. The domain of values for the argument to the `reset` option is profile specific and specified in the CLP-to-CIM Mapping Specification [13].

5.12 Source

Forms

```
-source
```

Example use

```
load –source `
ftp://adminstrator:passw0rd@myserver.com/private/administrator/`
firmware.img /system1/fw1
```

Behavior

The `source` option is used to specify the source for the transfer of a binary image. The desired source is specified as the argument to the `source` option. The source may be expressed as a UFiP if it is a Managed Element in the address space of the MAP. The source may also be expressed as a URI. The desired protocol to use for transferring the image is specified as part of the URI. This specification does not mandate support for a specific protocol. Thus the protocols supported are implementation specific.

5.13 Version

Forms

```
–version
–v
```

Example use

`start –version` Display version of “start” command verb

Behavior

The `version` option causes the Command Processor to return the version of the command verb that appears as the first term on the command line. CLP command verbs are assigned the version of the CLP in which they were introduced or in which they were last modified. The CLP version tracks the overall version of the syntax and is revised whenever a new CLP verb is added to the standard or when an existing verb’s syntax or semantics are revised. When the `version` option is used, the command verb itself is not executed.

When the `version` option is included in a command, the implementation MUST NOT execute the command. The implementation MUST return Command Response data which identifies the version of the verb supported by the implementation.

5.14 Wait

Forms

```
-wait
-w
```

Example use

```
stop /system3 –w
```

Behavior

When the `wait` option is included in a command, the implementation MUST NOT return control immediately to the Client, instead the implementation MUST withhold all Command Response data and command input control until all jobs related to this command have completed. The Wait option has no effect on the success or failure of the command or spawned jobs.

6 SM CLP Session

6.1 Authentication, Authorization, and Audit

The CLP service relies on user accounts and user groups to control access to Managed Elements through the service. Membership in a user group conveys to the user account the management privileges associated with that user group. User accounts MAY be members of more than one user group. User privileges are additive, membership in a group conveys to the user all of the privileges of that group, not just the privileges which overlap with additional groups to which the user belongs.

The CLP defines three user groups, each with an associated set of privileges. The three CLP defined user groups are Administrator, Operator, and Read Only. Implementations MUST support the Read Only and Administrator groups. Implementations SHOULD support the Operator group. Implementations MAY support additional user groups.

Authorization for the CLP defined user groups is at SM CLP command granularity. If the user account for the session is a member of a group that has permission to execute a command, the implementation MUST attempt to process the command. If the user account for the session is not a member of a group that has permission to execute the command, the implementation MUST NOT complete the requested command and the implementation MUST return Command Status of COMMAND EXECUTION FAILED, Job Error Type of Security Error, and a CIM Status of CIM_ERR_ACCESS_DENIED.

Table 19 Command Authorizations for CLP Groups lists the CLP defined user groups and the associated authorized CLP commands. For each user group listed in the Group column, implementations MUST authorize the CLP commands listed in the SM CLP Commands column and MUST NOT authorize CLP commands which do not appear in the SM CLP Commands column.

Table 19 Command Authorizations for CLP Groups

Group	SM CLP Commands
`Read Only`	cd, exit, help ,show, version
`Operator`	cd, exit, help, show, version, reset, start, stop, set, load, dump
`Administrator`	cd, exit, help, show, version, reset, start, stop, set, load, dump, create, delete

The following table lists the values of the CIM_AuthorizedPrivilege Activity property corresponding to the authorizations for each group.

Table 20 CIM_Privilege Definitions for CLP Groups

Group	CIM_AuthorizedPrivilege.Activity values
`Read Only`	Read
`Operator`	Read, Write, Execute
`Administrator`	Read, Write, Execute, Create, Delete

Prior to allowing a Client to issue any SM CLP commands, implementations MUST have an authenticated session established between the Client and the CLP service. Implementations MUST require that the credentials of the authenticated session identify a recognized user account on the MAP. Establishment of an authenticated session is implementation specific and is outside the scope of this specification.

Implementations MUST support adding, removing, displaying, and modifying user accounts through the CLP. Implementations MUST require that a user belong to the Administrator group in order to create, delete, or modify users and assign users to groups. This is an exception to the rules in Table 19 Command Authorizations for CLP Groups. For more information, see the CLP-to-CIM Mapping Specification [13]. The definition of the initial user account for accessing the CLP is implementation specific and outside the scope of this specification.

6.2 CLP Session

A CLP session is defined as a synchronous text message service exposed by a MAP CLP Service through (or over) an underlying transport protocol. All commands and responses in a CLP session are session data messages from the standpoint of the transport, and there are no required messages in a CLP session except the termination command. A CLP session exists from the time the service is invoked by one of the methods described in the corresponding transport mapping specification and exists until termination is requested by the Client by sending the CLP session termination command (“`exit`”).

The CLP session also ends if the service terminates for any other reason. When a graceful stop of the CLP session is initiated, the implementation MUST NOT accept any additional commands from the user and MUST finish processing the pending command, if there is one. The implementation MUST then end the session as if it was processing an `exit` command from the user. If an immediate stop of the CLP session is initiated, the implementation MUST immediately stop the CLP session and MUST NOT return a Command Response to the user. When terminating the CLP session, the implementation MAY also terminate the transport session.

6.2.1 Session Environment

Interaction with the CLP service is done within the context of a session. For each active session, The CLP service maintains a session context for each active session. The SESSION term (see Section 3.1.7) references the session in which the term is used. This provides a convenient mechanism for a user to access their current session. The CLP service maintains useful information such as the Current Default Target, the default values for options, the default target of the session, and the credentials provided when the session was initiated in the session context. Session default values are only supported on a subset of options. For a complete list of options and session attributes that implementations are required to support, see the CLP-to-CIM Mapping Specification [13]. Implementations MUST allow a user to modify settings for their own session irrespective of which groups the user is a member. This is an exception to the rules in Table 19 Command Authorizations for CLP Groups.

6.2.2 CLP Service

A CLP session is managed by the CLP Service. Within the scope of a single MAP/CLP Service, a user may see other user sessions if the implementation supports multiple, simultaneous sessions, and the user has Administrator authorization.

Implementations MUST require that a user has Administrator authorization in order to show other user sessions, stop other user sessions, or stop the CLP Service.

If a graceful stop of the CLP Service is requested, the implementation MUST gracefully stop each active CLP session as described in Section 6.2 prior to ending the service. If an immediate stop of the CLP Service is requested, the implementation MUST immediately stop each active CLP session as described in Section 6.2 prior to ending the service.

6.2.3 CLP Interaction Model

The CLP observes the following interaction models documented in this section unless otherwise noted:

- CLP Service Discovery (documented in the CLP Service Discovery Specification [14])
- CLP Session Establishment
- CLP Command Interaction
- CLP Session Switching
- CLP Session Termination

The following sections define the normal and exception message sequences of each model. The following conventions are used in each of the following interaction diagrams:

  - Dashed horizontal lines indicate expected behavior. The exact behaviour is outside the scope of this specification.
  - Solid horizontal lines indicate behavior required by this specification.

When a CLP session is established, implementations MUST return the following string followed by the first command prompt:

```
=== SMCLP v<major>.<minor>.<patchlevel> <OEM Data> ===
```

Where <major> is the major revision of the CLP specification supported, <minor> is the minor revision of the CLP specification supported, <patchlevel> is the patch level of the CLP, and <OEM Data> is a vendor defined versioning string. For implementations compliant with this version of the specification, <major> has a value of 1 (one), <minor> has a value of 0 (zero), <patchlevel> has a value of 0 (zero). This version string followed by a command prompt is the initial CLP prompt.

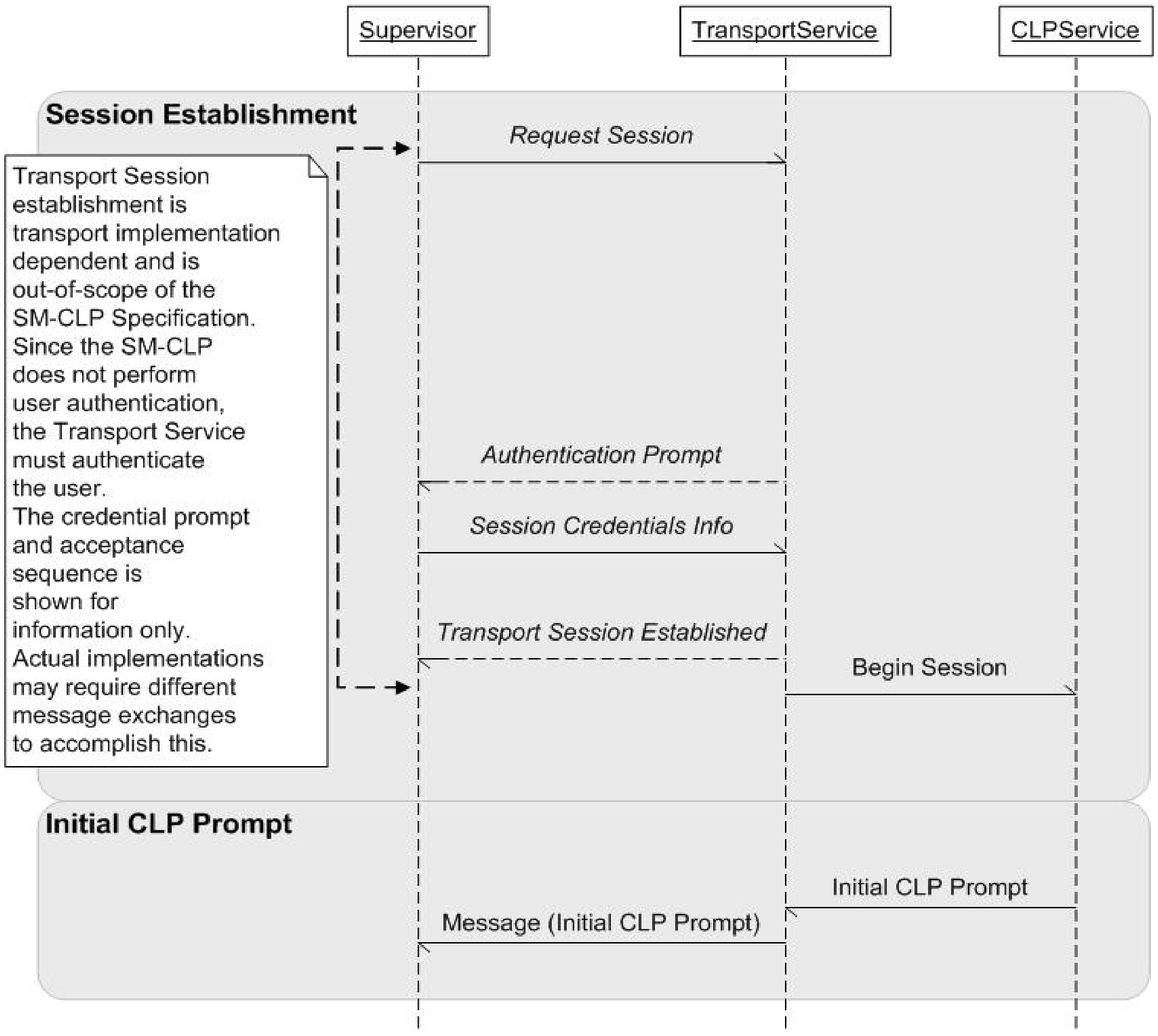

Supervisor
TransportService
CLPService
Session Establishment
Transport Session establishment is transport implementation dependent and is out-of-scope of the SM-CLP Specification. Since the SM-CLP does not perform user authentication, the Transport Service must authenticate the user. The credential prompt and acceptance sequence is shown for information only. Actual implementations may require different message exchanges to accomplish this.
Request Session
Authentication Prompt
Session Credentials Info
Transport Session Established
Begin Session
Initial CLP Prompt
Initial CLP Prompt
Message (Initial CLP Prompt)

Figure 1 Session Establishment Sequence

6.2.3.1 Command Interaction

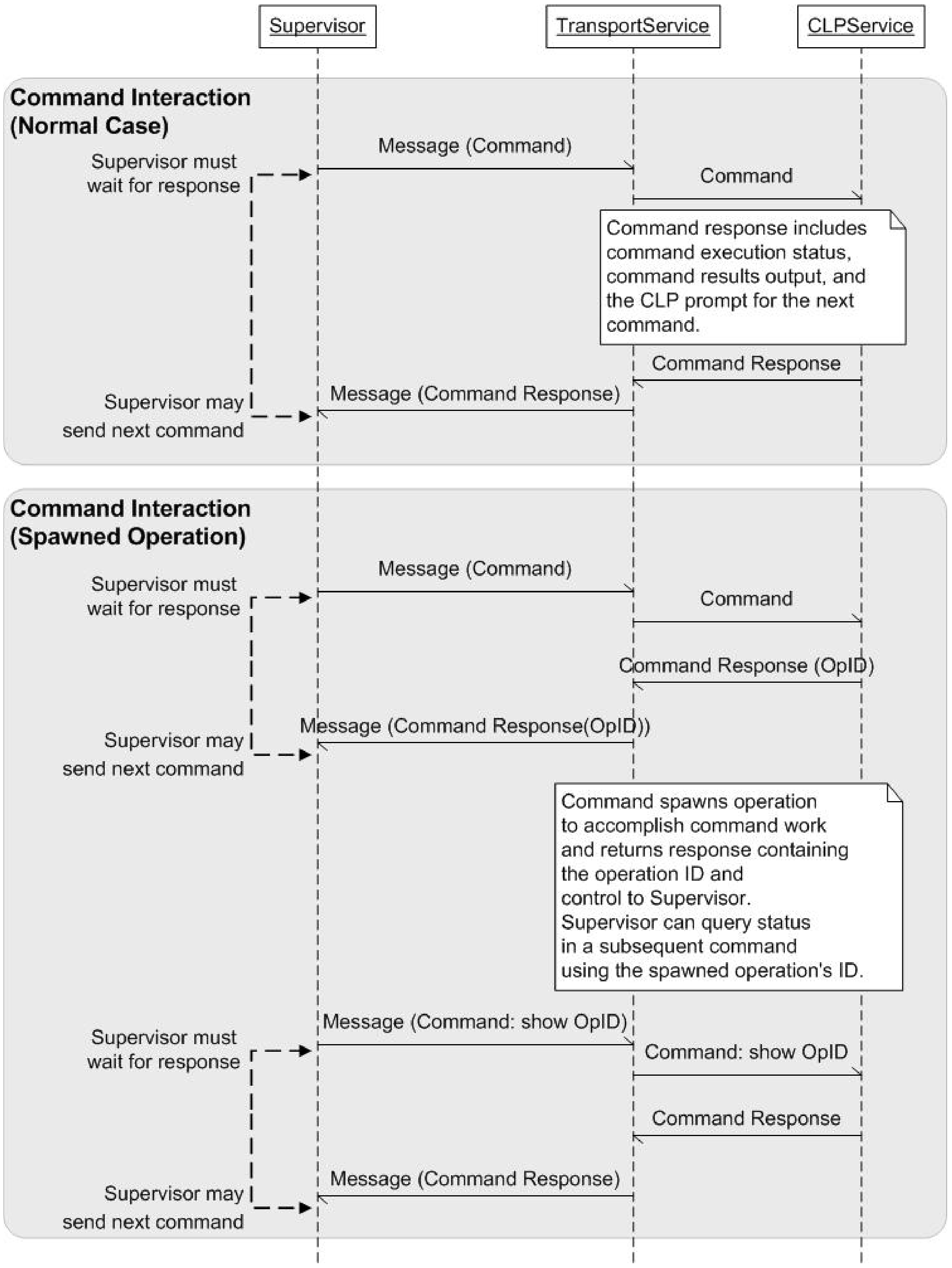

Supervisor
TransportService
CLPService
Command Interaction (Normal Case)
Supervisor must wait for response
Message (Command)
Command
Command response includes command execution status, command results output, and the CLP prompt for the next command.
Command Response
Message (Command Response)
Supervisor may send next command
Command Interaction (Spawned Operation)
Supervisor must wait for response
Message (Command)
Command
Command Response (OpID)
Message (Command Response(OpID))
Supervisor may send next command
Command spawns operation to accomplish command work and returns response containing the operation ID and control to Supervisor. Supervisor can query status in a subsequent command using the spawned operation's ID.
Supervisor must wait for response
Message (Command: show OpID)
Command: show OpID
Command Response
Message (Command Response)
Supervisor may send next command

Figure 2 Command Interaction Sequences

6.2.3.2 Session Switching

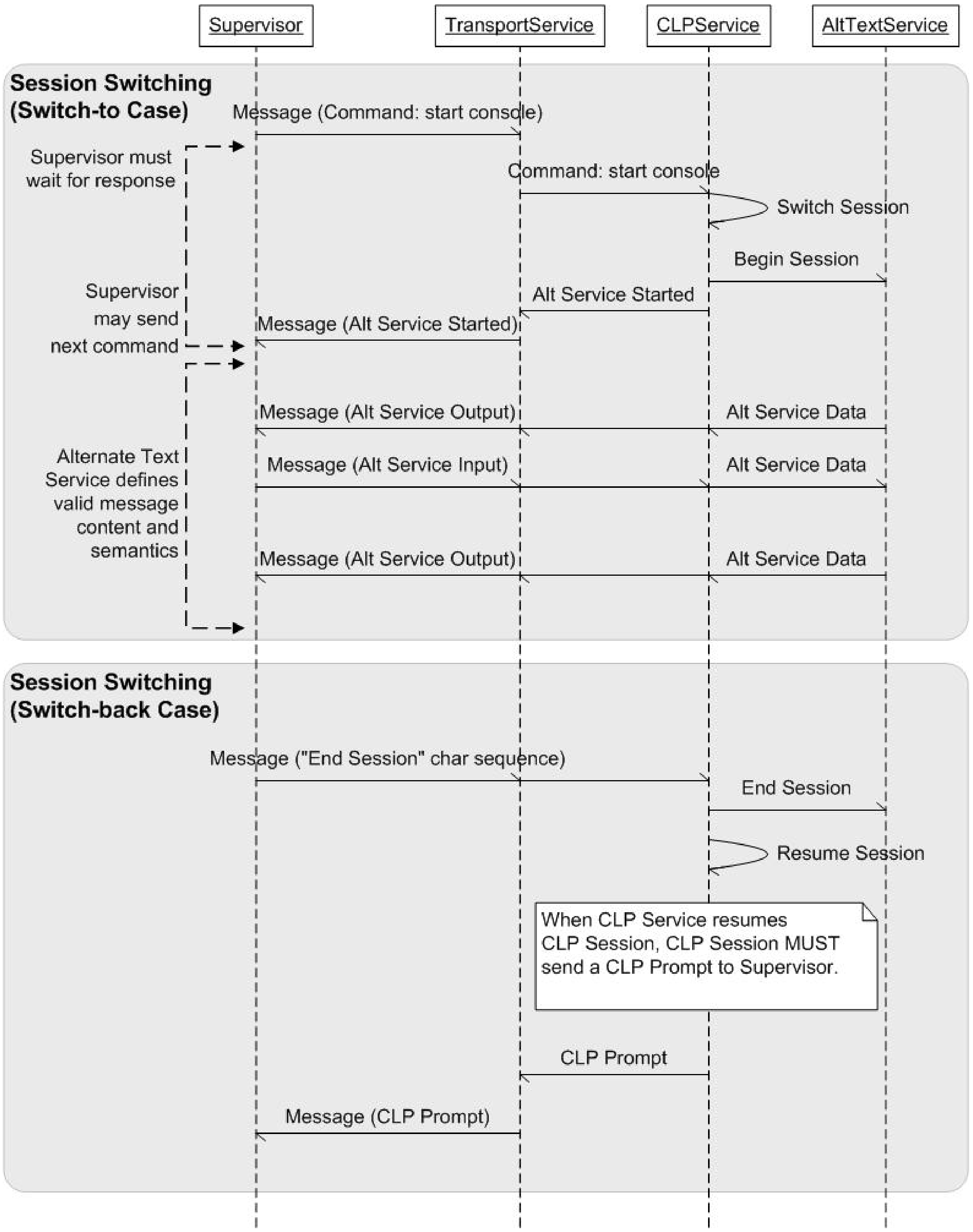
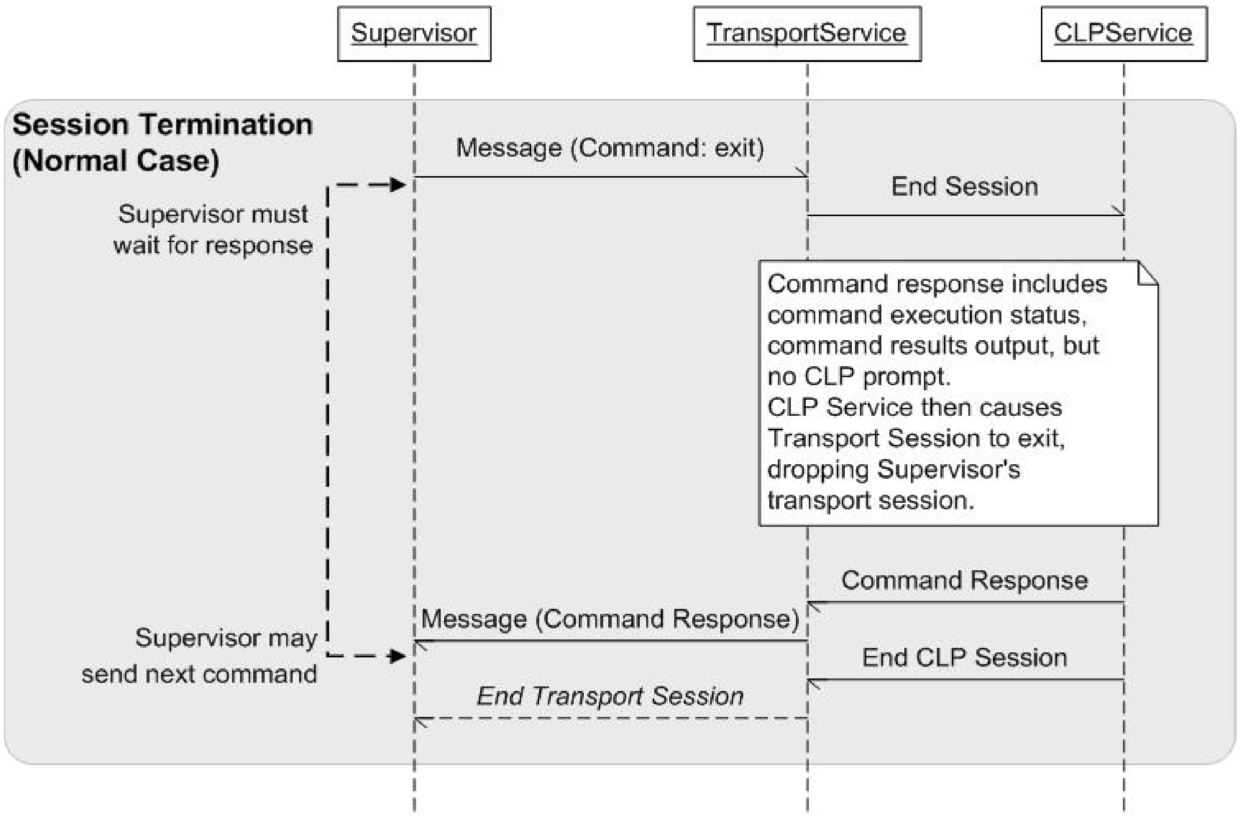

Supervisor
TransportService
CLPService
AltTextService
Session Switching (Switch-to Case)
Message (Command: start console)
Supervisor must wait for response
Command: start console
Switch Session
Begin Session
Supervisor may send next command
Alt Service Started
Message (Alt Service Started)
Message (Alt Service Output)
Alt Service Data
Alternate Text Service defines valid message content and semantics
Message (Alt Service Input)
Alt Service Data
Message (Alt Service Output)
Alt Service Data
Session Switching (Switch-back Case)
Message ("End Session" char sequence)
End Session
Resume Session
When CLP Service resumes CLP Session, CLP Session MUST send a CLP Prompt to Supervisor.
CLP Prompt
Message (CLP Prompt)

Figure 3 Session Switching Sequences

6.2.3.3 Session Termination


Supervisor
TransportService
CLPService
Session Termination (Normal Case)
Message (Command: exit)
End Session
Supervisor must wait for response
Command response includes command execution status, command results output, but no CLP prompt. CLP Service then causes Transport Session to exit, dropping Supervisor's transport session.
Command Response
Message (Command Response)
Supervisor may send next command
End CLP Session
End Transport Session

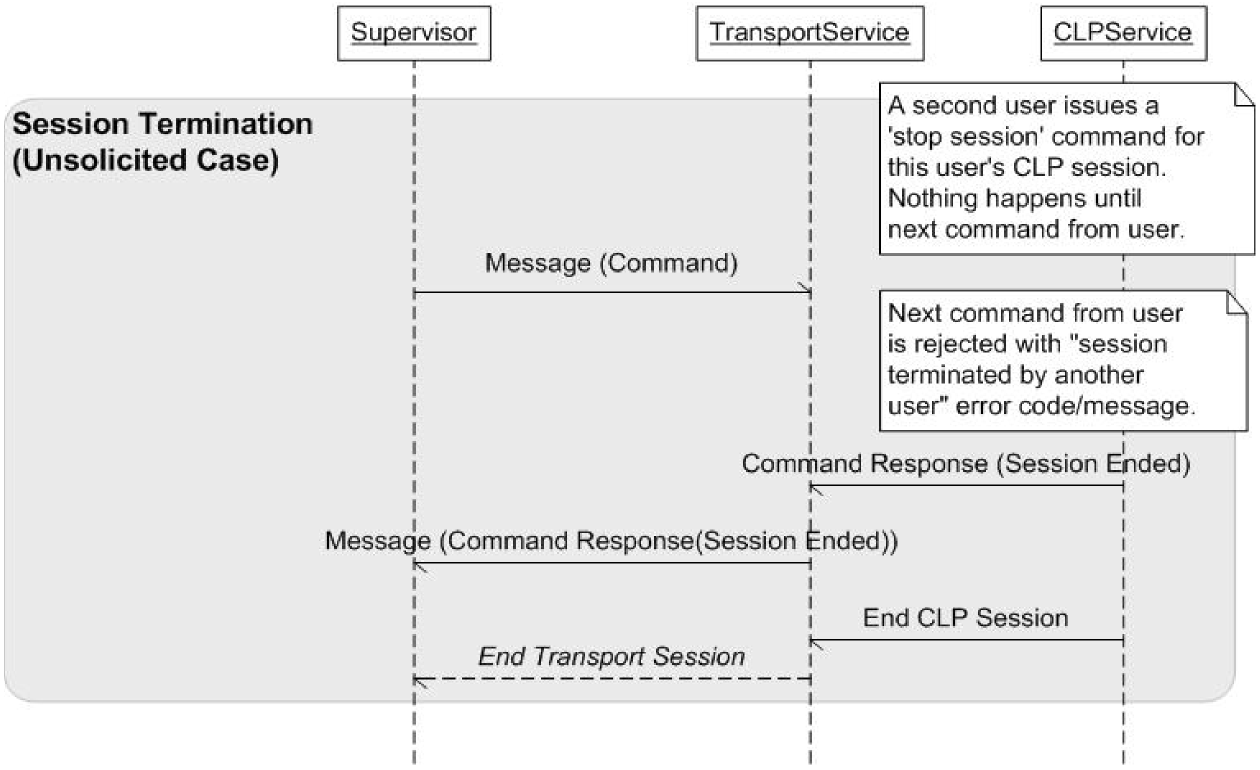

Supervisor
TransportService
CLPService
Session Termination (Unsolicited Case)
A second user issues a 'stop session' command for this user's CLP session. Nothing happens until next command from user.
Message (Command)
Next command from user is rejected with "session terminated by another user" error code/message.
Command Response (Session Ended)
Message (Command Response(Session Ended))
End CLP Session
End Transport Session

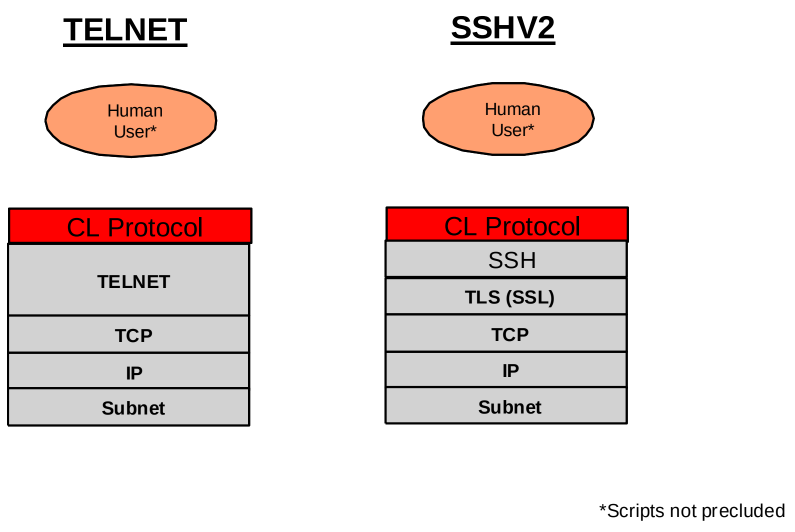
Figure 4 Session Termination Sequences

6.3 Transport

As depicted in the following figure, the CLP is carried over a text message-based protocol. At present, two text message protocol mappings of the CLP are specified: Telnet and the Secure Shell (SSH).

CLP implementations MUST support either Telnet or SSHv2, though support for SSHv2 is RECOMMENDED. Implementations MAY support more than one message passing mechanism for accessing the CLP Service.

Implementations SHOULD provide a method for activating/enabling and deactivating/disabling the supported Telnet or SSHv2 protocol.



The CLP Service exposed by the implementation MUST operate the same whether invoked from an SSH-loaded shell; loaded directly by the SSH daemon; operating as a subsystem; or loaded from a Telnet `pTTY`. This is primarily a test statement because there is nothing in Telnet or SSH that should directly effect the operation of the implementation or its terminal I/O operations as seen from the transport client application. Implementations of the CLP Service MUST ensure that output received by clients of the implementation is consistent irrespective of the transport over which the implementation is accessed. For example, if the CLP Service utilizes a transport which inserts characters into the text stream at one end of the transport, the transport will need to remove the characters prior to presenting the text stream to the Client.

6.3.1 Telnet

Telnet is an IETF-defined Network Virtual Terminal (NVT) protocol that operates over the TCP transport layer. The Telnet protocol provides a standardized interface, through which a program on one host (the Telnet client) MAY access the resources of another host (the Telnet server) as though the client were a local terminal connected to the server.

The basic operational characteristics of an NVT are:

- The data representation is 7-bit ASCII transmitted in 8-bit bytes.
- The NVT provides a local echo function.

A CLP implementation over Telnet MUST support the base Telnet RFC and Standards and MUST support the following IETF RFCs:

Table 21 Telnet Transport Support Requirements

Number	Title	Author or Ed.	Date Released	Status
STD0008 RFC0854	Telnet Protocol Specification [20]	J. Postel, J.K. Reynolds	May-01-1983	STANDARD
STD0008 RFC0855	Telnet Option Specifications [21]	J. Postel, J.K. Reynolds	May-01-1983	STANDARD

Note that, in addition to standard NVT (Network Virtual Terminal) operation as described in RFC 854, implementations MUST respond to requests to support UTF-8.

Each line SHOULD be terminated by a Carriage Return and Line Feed (ISO-10646 UTF-8 [RFC3629]).

A CLP implementation is NOT REQUIRED to listen on any specific predefined TCP port number. CLP TCP port numbers may be administratively set or it may be discovered through the Service Location Protocol (SLP), see SM CLP Discovery Using SLP [14] for more information.

6.3.2 Secure Shell (SSH) Version 2

SSH is a tool for secure remote login over insecure networks. It provides an encrypted terminal session with strong authentication of both the server and client.

Secure Shell provides three main capabilities:

- Secure command-shell

- Secure file transfer

- Port forwarding

SSHv2 is a protocol being defined by the IETF in the SECSH Working Group.

The SSHv2 protocol is described in five core documents.

- SSH Protocol Architecture [15] - describes the overall design of SSH
- SSH Transport Layer Protocol [16] - provides a single, full-duplex, byte-oriented connection between client and server, with privacy, integrity, server authentication, and man-in-the-middle protection.
- SSH Authentication Protocol [17] - identifies the client to the server
- SSH Connection Protocol [18] - provides richer, application-support services over the transport pipe, such as channel multiplexing, flow control, remote program execution, signal propagation, connection forwarding, etc
- SSH Assigned Numbers [19] - this document gathers together and lists various constant assignments made in the other drafts.

A CLP implementation over SSHv2 MUST support the base SSHv2 RFC & Standards and the following IETF RFCs and Internet Drafts:

Table 22 SSHv2 Transport Support Requirements

Name	Title	Status
draft-ietf-secsh-architecture-17	SSH Protocol Architecture	Internet Draft
draft-ietf-secsh-transport-19	SSH Transport Layer Protocol	Internet Draft
draft-ietf-secsh-userauth-22	SSH Authentication Protocol	Internet Draft
draft-ietf-secsh-connect-19	SSH Connection Protocol	Internet Draft
draft-ietf-secsh-assignednumbers-07	SSH Protocol Assigned Numbers	Internet Draft

The Secure Command-shell Mode of operation MUST be supported by a CLP implementation.

A CLP implementation is not required to listen on any specific predefined TCP port #. A CLP TCP Port number may be administratively set or it may be discovered through the SLP), see SM CLP Discovery Using SLP [14] for more information.

No specific encryption algorithms or authentication protocols are specified in this specification. CLP will rely on the referenced SSH specifications to identify REQUIRED techniques.

As with direct Telnet (and for the same reasons), the implementation MUST support a mechanism to restrict access to the implementation using tunneled protocols, and the factory-default setting MUST disable Telnet as a tunneled protocol.

Appendix A – Relationships to Other Standards and Specifications

The Command Line Protocol command syntax blends many relevant command line syntax conventions and common practices in it's design. The following sections describe how the CLP relates to various syntax conventions, guidelines, and specifications.

A.1 W3C Universal Resource Identifiers (URI)

URIs come into consideration under two aspects of the CLP: as targets and as values.

The target specified in the command is intended to be able to be used as the Model Path parameter in an implementation that also supports a structured interface, such as WBEM. This is really more an aspect of the SM Managed Element Addressing Specification [4] combined with the WBEM URI Syntax specification [8].

URIs are expected to be used as values for some keyword=value pairs. For instance, an implementation MAY use a URI as a boot device as the location of the data source for applying a firmware image. The implementation is expected to validate the URI and make sure the schema name included in the URI is valid for the given implementation. The CLP itself does not require any schema or enforce any URIs in specific, but they are expected to adhere to RFC 2396 [9], RFC 2719 [10] and RFC 2717 [11].

A.2 W3C Extensible Markup Language (XML)

The CLP supports generating XML output data [12], as well as CSV (Comma Separated Value) mode and modes for plain text output. XML was chosen as a supported output format due to its acceptance in the industry, establishment as a standard, and need for Clients to import data obtained through the CLP into other applications. For more information on the XML Output mode, see Section 3.2.4.2.3.

A.3 POSIX Utility Conventions

The POSIX Utility Conventions [7] were considered when defining the CLP syntax. The CLP syntax adheres to as many of the POSIX Utility Guidelines as are feasible, but does not conform to the POSIX Utility Argument Syntax.

The POSIX Utility Argument Syntax was found inappropriate for two reasons. First, it was imperative to have the command target be deterministic in order to accommodate low end implementations. Second, in order to provide a consistent, predictable mapping to the CIM Schema, the CLP syntax uses the convention that option terms apply to command verbs and parameter terms apply to the command target, using the "keyword=value" model.

The CLP syntax compares to the thirteen POSIX Utility Guidelines as follows:

- Adhering to Guideline 1 is a goal of the CLP, since it is a desire to keep the verb names short. However, the adopted extensibility conventions imply that it is expected that any extensions will find it problematic to adhere to Guideline 1.

- The CLP syntax currently has no numbers in the commands, nor are verbs required to be entered only in lower case. Therefore, a user or script that adheres to Guideline 2 could find CLP implementations compatible.
- The CLP allows both a short name form and long name form for option names. Therefore any human user or script which is accustomed to one letter options names will find CLP implementations compatible. Allowing whole word options not only allows scripts to be more readable, but allows a shorter learning curve of the CLP. The “W” option is not reserved by the CLP. CLP option names are case insensitive.
- The CLP adheres to Guideline 4: all CLP options MUST be preceded by the ‘-‘ delimiter character.
- The CLP does not allow grouping of options behind a single hyphen and therefore does not adhere to Guideline 5. Most options require a parameter and the decision to allow full length option names eliminated the ability to adhere to this guideline.
- The CLP adheres to Guideline 6 and recognizes the space character as the command line term delimiter.
- The CLP adheres to Guideline 7. Each option either always requires an argument or never requires an argument.
- The CLP recognizes the use of the comma character to separate items in a list in a single argument string for both options and properties and therefore adheres to Guideline 8 with one caveat. A comma character at the beginning or end of the option argument string is not inherently illegal and is command dependent.
- The CLP does adhere to Guideline 9. The command line is REQUIRED to be in the order of Verb Option Target Property.
- The CLP does not recognize the “--“ (hyphen hyphen) term as an “end of options” indicator, nor as a “long option” indicator (as is used in some UNIX utilities. Therefore, the CLP does not adhere to guideline 10.
- The CLP allows options to be specified in any order, but does not allow options to appear twice on any command line, nor does it allow mutually exclusive options or options that do not apply in the current context. Therefore the CLP adheres to part of Guideline 11.
- When examining Guideline 12, the CLP uses keyword=value pairs for operands which require assignment, and just keywords where they do not. Since these are often CIM Schema properties, they are not order dependent. Therefore, operands positions do not matter. This is true regardless of CLP command.
- Guideline 13 is out of scope for the CLP. The CLP does not allow in-stream input and therefore has no need for an input operand.

Appendix B – Command Response schema

The following is the schema defining the XML output for a Command Response.

```
<?xml version="1.0" encoding="UTF-8"?>

<xs:schema xmlns:xs="http://www.w3.org/2001/XMLSchema"
xmlns="http://www.dmtf.org/smash/2005/01/clpxml"
targetNamespace="http://www.dmtf.org/smash/2005/01/clpxml"
elementFormDefault="qualified" attributeFormDefault="unqualified">

	<xs:element name="response">

		<xs:annotation>

			<xs:documentation>Wrapper element containing an entire command
response. This is the Command Response data element.</xs:documentation>

		</xs:annotation>

		<xs:complexType>

			<xs:sequence>

				<xs:element name="command" type="commandinputtype"/>

				<xs:element name="cmdstat" type="commandstattype"/>

				<xs:choice minOccurs="0">

					<xs:element name="cd" type="cdresultstype"/>

					<xs:element name="create" type="createresultstype"/>

					<xs:element name="delete" type="deleteresultstype"/>

					<xs:element name="dump" type="dumpresultstype"/>

					<xs:element name="exit" type="exitresultstype"/>

					<xs:element name="help" type="helpresultstype"/>

					<xs:element name="load" type="loadresultstype"/>

					<xs:element name="reset" type="resetresultstype"/>

					<xs:element name="set" type="setresultstype"/>

					<xs:element name="show" type="showresultstype"/>

					<xs:element name="start" type="startresultstype"/>

					<xs:element name="stop" type="stopresultstype"/>

					<xs:element name="version" type="versionresultstype"/>

					<xs:element name="oemverb"
type="oemverbresultstype"/>

				</xs:choice>

				<xs:element name="oemdata" type="oemdatatype"
minOccurs="0"/>
```

```
                    </xs:sequence>
                </xs:complexType>
        </xs:element>
        <xs:complexType name="messagetype">
                <xs:annotation>
                        <xs:documentation>The Message data element.</xs:documentation>
                </xs:annotation>
                <xs:sequence>
                        <xs:element name="owningentity"/>
                        <xs:element name="messageid"/>
                        <xs:element name="messagetext" minOccurs="0"/>
                        <xs:element name="messagearg" type="messageargtype" minOccurs="0"
maxOccurs="unbounded"/>
                </xs:sequence>
        </xs:complexType>
        <xs:complexType name="messageargtype">
                <xs:annotation>
                        <xs:documentation>Message Argument property</xs:documentation>
                </xs:annotation>
                <xs:sequence>
                        <xs:element name="index"/>
                        <xs:element name="value"/>
                </xs:sequence>
        </xs:complexType>
        <xs:complexType name="messagestype">
                <xs:annotation>
                        <xs:documentation>Collection of Message data
elements</xs:documentation>
                </xs:annotation>
                <xs:sequence>
                        <xs:element name="message" type="messagetype"
maxOccurs="unbounded"/>
                </xs:sequence>
        </xs:complexType>
        <xs:complexType name="instancereferencetype">
                <xs:annotation>
                        <xs:documentation>A complexType that just contains the ufit and ufip
elements of an instance.</xs:documentation>
                </xs:annotation>
                <xs:sequence>
                        <xs:element name="ufit" type="ufittype"/>
                        <xs:element name="ufip" type="ufiptype"/>
                </xs:sequence>
        </xs:complexType>
        <xs:complexType name="instanceassociationtype">
                <xs:annotation>
                        <xs:documentation>A Managed Element instance container for an
Association</xs:documentation>
                </xs:annotation>
                <xs:sequence>
                        <xs:element name="ufit" type="ufittype"/>
                        <xs:element name="ufip" type="ufiptype"/>
                        <xs:element name="associations" type="associationcollectiontype"/>
                </xs:sequence>
        </xs:complexType>
        <xs:complexType name="instancepropertytype">
                <xs:annotation>
                        <xs:documentation>A Managed Element instance and its
properties.</xs:documentation>
                </xs:annotation>
                <xs:sequence>
                        <xs:element name="ufit" type="ufittype"/>
                        <xs:element name="ufip" type="ufiptype"/>
                        <xs:element name="properties" type="propertycollectiontype"/>
                </xs:sequence>
        </xs:complexType>
        <xs:complexType name="instancetype">
                <xs:annotation>
```

```
				<xs:documentation>A complexType that contains the ufit, ufip,
properties, and verbs elements that an instance element can contain.</xs:documentation>
			</xs:annotation>
			<xs:sequence>
				<xs:element name="ufit" type="ufittype"/>
				<xs:element name="ufip" type="ufiptype"/>
				<xs:element name="properties" type="propertycollectiontype"
minOccurs="0"/>
				<xs:element name="associations" type="associationcollectiontype"
minOccurs="0"/>
				<xs:element name="verbs" type="verblisttype" minOccurs="0"/>
			</xs:sequence>
		</xs:complexType>
		<xs:complexType name="commandinputtype">
			<xs:annotation>
				<xs:documentation>Describes the command that was
processed.</xs:documentation>
			</xs:annotation>
			<xs:sequence>
				<xs:element name="inputline">
					<xs:annotation>
						<xs:documentation>Simple text string echoing the input
line.</xs:documentation>
					</xs:annotation>
					<xs:simpleType>
						<xs:restriction base="xs:string">
							<xs:minLength value="0"/>
						</xs:restriction>
					</xs:simpleType>
				</xs:element>
			</xs:sequence>
		</xs:complexType>
		<xs:complexType name="operationtype">
			<xs:annotation>
				<xs:documentation>Job data element.</xs:documentation>
```

```
        </xs:annotation>
        <xs:sequence>
            <xs:element name="job_id" type="jobidtype"/>
            <xs:element name="joberr" type="joberrtype" minOccurs="0"/>
        </xs:sequence>
    </xs:complexType>
    <xs:complexType name="commandstattype">
        <xs:annotation>
            <xs:documentation>The command status element.</xs:documentation>
        </xs:annotation>
        <xs:sequence>
            <xs:element name="status" type="statustype">
                <xs:annotation>
                    <xs:documentation>Command Status data
element</xs:documentation>
                </xs:annotation>
            </xs:element>
            <xs:element name="status_tag" type="statustagtype" minOccurs="0">
                <xs:annotation>
                    <xs:documentation>Command Status Tag data
element</xs:documentation>
                </xs:annotation>
            </xs:element>
            <xs:sequence minOccurs="0">
                <xs:element name="error" type="errortype">
                    <xs:annotation>
                        <xs:documentation>One of error values from the
Processing Error Values and Tags table</xs:documentation>
                    </xs:annotation>
                </xs:element>
                <xs:element name="error_tag" type="errortagtype"
minOccurs="0">
                    <xs:annotation>
```

```
                                        <xs:documentation>The error_tag value from the
Processing Error Values and Tags table which corresponds to the error value
above.</xs:documentation>
                                   </xs:annotation>
                              </xs:element>
                              <xs:element name="messages" type="messagestype"
minOccurs="0"/>
                         </xs:sequence>
                         <xs:element name="job" type="operationtype" minOccurs="0"/>
                    </xs:sequence>
          </xs:complexType>
          <xs:simpleType name="statustype">
                    <xs:restriction base="xs:int">
                         <xs:enumeration value="0"/>
                         <xs:enumeration value="1"/>
                         <xs:enumeration value="2"/>
                         <xs:enumeration value="3"/>
                    </xs:restriction>
          </xs:simpleType>
          <xs:simpleType name="statustagtype">
                    <xs:restriction base="xs:string">
                         <xs:enumeration value="COMMAND COMPLETED"/>
                         <xs:enumeration value="COMMAND SPAWNED"/>
                         <xs:enumeration value="COMMAND PROCESSING FAILED"/>
                         <xs:enumeration value="COMMAND EXECUTION FAILED"/>
                    </xs:restriction>
          </xs:simpleType>
          <xs:simpleType name="errortype">
                    <xs:restriction base="xs:int">
                         <xs:enumeration value="255"/>
                         <xs:enumeration value="254"/>
                         <xs:enumeration value="253"/>
                         <xs:enumeration value="252"/>
                         <xs:enumeration value="251"/>
```

```
                <xs:enumeration value="250"/>
                <xs:enumeration value="249"/>
                <xs:enumeration value="248"/>
                <xs:enumeration value="247"/>
                <xs:enumeration value="246"/>
                <xs:enumeration value="245"/>
                <xs:enumeration value="244"/>
            </xs:restriction>
        </xs:simpleType>
        <xs:simpleType name="errortagtype">
            <xs:restriction base="xs:string">
                <xs:enumeration value="COMMAND ERROR - UNSPECIFIED"/>
                <xs:enumeration value="COMMAND NOT SUPPORTED"/>
                <xs:enumeration value="COMMAND NOT RECOGNIZED"/>
                <xs:enumeration value="COMMAND SYNTAX ERROR"/>
                <xs:enumeration value="INVALID OPTION"/>
                <xs:enumeration value="INVALID ARGUMENT"/>
                <xs:enumeration value="OUTPUT FORMAT NOT SUPPORTED"/>
                <xs:enumeration value="MISSING ARGUMENT"/>
                <xs:enumeration value="OPTION NOT SUPPORTED"/>
                <xs:enumeration value="INVALID TARGET"/>
                <xs:enumeration value="REQUIRED OPTION MISSING"/>
                <xs:enumeration value="QUEUE FULL"/>
            </xs:restriction>
        </xs:simpleType>
        <xs:complexType name="joberrtype">
            <xs:annotation>
                <xs:documentation>The operation error element</xs:documentation>
            </xs:annotation>
            <xs:sequence>
                <xs:element name="errtype">
                    <xs:annotation>
                        <xs:documentation>Execution Error data
element</xs:documentation>
```

```
                                </xs:annotation>
                                <xs:simpleType>
                                        <xs:restriction base="xs:int">
                                                <xs:enumeration value="0"/>
                                                <xs:enumeration value="1"/>
                                                <xs:enumeration value="2"/>
                                                <xs:enumeration value="3"/>
                                                <xs:enumeration value="4"/>
                                                <xs:enumeration value="5"/>
                                                <xs:enumeration value="6"/>
                                                <xs:enumeration value="7"/>
                                                <xs:enumeration value="8"/>
                                                <xs:enumeration value="9"/>
                                                <xs:enumeration value="10"/>
                                        </xs:restriction>
                                </xs:simpleType>
                        </xs:element>
                        <xs:element name="errtype_desc" minOccurs="0">
                                <xs:annotation>
                                        <xs:documentation>Execution Error Tag data
element</xs:documentation>
                                </xs:annotation>
                                <xs:simpleType>
                                        <xs:restriction base="xs:string">
                                                <xs:enumeration value="Unknown"/>
                                                <xs:enumeration value="Other"/>
                                                <xs:enumeration value="Communications Error"/>
                                                <xs:enumeration value="Quality of Service
Error"/>
                                                <xs:enumeration value="Software Error"/>
                                                <xs:enumeration value="Hardware Error"/>
                                                <xs:enumeration value="Environmental Error"/>
                                                <xs:enumeration value="Security Error"/>
                                                <xs:enumeration value="Oversubscription Error"/>
                        <xs:enumeration value="Unavailable Resource
Error"/>
                        <xs:enumeration value="Unsupported Operation
Error"/>
                    </xs:restriction>
                </xs:simpleType>
            </xs:element>
            <xs:element name="cimstat">
                <xs:annotation>
                    <xs:documentation>CIM Status data
element</xs:documentation>
                </xs:annotation>
                <xs:simpleType>
                    <xs:restriction base="xs:int">
                        <xs:enumeration value="1"/>
                        <xs:enumeration value="2"/>
                        <xs:enumeration value="3"/>
                        <xs:enumeration value="4"/>
                        <xs:enumeration value="5"/>
                        <xs:enumeration value="6"/>
                        <xs:enumeration value="7"/>
                        <xs:enumeration value="8"/>
                        <xs:enumeration value="9"/>
                        <xs:enumeration value="10"/>
                        <xs:enumeration value="11"/>
                        <xs:enumeration value="12"/>
                        <xs:enumeration value="13"/>
                        <xs:enumeration value="14"/>
                        <xs:enumeration value="15"/>
                        <xs:enumeration value="16"/>
                        <xs:enumeration value="17"/>
                        <xs:enumeration value="18"/>
                        <xs:enumeration value="19"/>
                        <xs:enumeration value="20"/>
```

```
                                        </xs:restriction>
                                </xs:simpleType>
                        </xs:element>
                        <xs:element name="cimstat_desc" minOccurs="0">
                                <xs:annotation>
                                        <xs:documentation>CIM Status Description data
element</xs:documentation>
                                </xs:annotation>
                                <xs:simpleType>
                                        <xs:restriction base="xs:string">
                                                <xs:enumeration value="CIM_ERR_FAILED"/>
                                                <xs:enumeration
value="CIM_ERR_ACCESS_DENIED"/>
                                                <xs:enumeration
value="CIM_ERR_INVALID_NAMESPACE"/>
                                                <xs:enumeration
value="CIM_ERR_INVALID_PARAMETER"/>
                                                <xs:enumeration
value="CIM_ERR_INVALID_CLASS"/>
                                                <xs:enumeration
value="CIM_ERR_NOT_FOUND"/>
                                                <xs:enumeration
value="CIM_ERR_NOT_SUPPORTED"/>
                                                <xs:enumeration
value="CIM_ERR_CLASS_HAS_CHILDREN"/>
                                                <xs:enumeration
value="CIM_ERR_CLASS_HAS_INSTANCES"/>
                                                <xs:enumeration
value="CIM_ERR_INVALID_SUPERCLASS"/>
                                                <xs:enumeration
value="CIM_ERR_ALREADY_EXISTS"/>
                                                <xs:enumeration
value="CIM_ERR_NO_SUCH_PROPERTY"/>
                                                <xs:enumeration
value="CIM_ERR_MISMATCH"/>
                                                <xs:enumeration
value="CIM_ERR_QUERY_LANGUAGE_NOT_SUPPORTED"/>
```

```
                                                        <xs:enumeration
value="CIM_ERR_INVALID_QUERY"/>
                                                        <xs:enumeration
value="CIM_ERR_METHOD_NOT_AVAILABLE"/>
                                                        <xs:enumeration
value="CIM_ERR_METHOD_NOT_FOUND"/>
                                                        <xs:enumeration
value="CIM_ERR_UNEXPECTED_RESPONSE"/>
                                                        <xs:enumeration
value="CIM_ERR_INVALID_RESPONSE_DESTINATION"/>
                                                        <xs:enumeration
value="CIM_ERR_NAMESPACE_NOT_EMPTY"/>
                                                </xs:restriction>
                                        </xs:simpleType>
                                </xs:element>
                                <xs:element name="severity">
                                        <xs:annotation>
                                                <xs:documentation>Severity data
element</xs:documentation>
                                        </xs:annotation>
                                        <xs:simpleType>
                                                <xs:restriction base="xs:int">
                                                        <xs:enumeration value="0"/>
                                                        <xs:enumeration value="2"/>
                                                        <xs:enumeration value="3"/>
                                                        <xs:enumeration value="4"/>
                                                        <xs:enumeration value="5"/>
                                                </xs:restriction>
                                        </xs:simpleType>
                                </xs:element>
                                <xs:element name="severity_desc" minOccurs="0">
                                        <xs:annotation>
                                                <xs:documentation>Severity Description data
element</xs:documentation>
                                        </xs:annotation>
                                        <xs:simpleType>
                                                <xs:restriction base="xs:string">
                                                        <xs:enumeration value="Unknown"/>
                                                        <xs:enumeration value="Low"/>
                                                        <xs:enumeration value="Medium"/>
                                                        <xs:enumeration value="High"/>
                                                        <xs:enumeration value="Fatal"/>
                                                </xs:restriction>
                                        </xs:simpleType>
                                </xs:element>
                                <xs:element name="probcause" minOccurs="0">
                                        <xs:annotation>
                                                <xs:documentation>Probable Cause data
element</xs:documentation>
                                        </xs:annotation>
                                        <xs:simpleType>
                                                <xs:restriction base="xs:int">
                                                        <xs:minInclusive value="0"/>
                                                        <xs:maxInclusive value="130"/>
                                                </xs:restriction>
                                        </xs:simpleType>
                                </xs:element>
                                <xs:element name="probcause_desc" minOccurs="0">
                                        <xs:annotation>
                                                <xs:documentation>Probable Cause Description data
element</xs:documentation>
                                        </xs:annotation>
                                        <xs:simpleType>
                                                <xs:restriction base="xs:string"/>
                                        </xs:simpleType>
                                </xs:element>
                                <xs:element name="recmdaction" minOccurs="0"
maxOccurs="unbounded">
                                        <xs:annotation>
                                                <xs:documentation>Recommended Action
property</xs:documentation>
```

```
                    </xs:annotation>
                </xs:element>
                <xs:element name="errsource" minOccurs="0">
                    <xs:annotation>
                        <xs:documentation>Error Source
property</xs:documentation>
                    </xs:annotation>
                    <xs:simpleType>
                        <xs:restriction base="xs:string"/>
                    </xs:simpleType>
                </xs:element>
                <xs:element name="errsourceform" minOccurs="0">
                    <xs:annotation>
                        <xs:documentation>Error Source Form
property</xs:documentation>
                    </xs:annotation>
                </xs:element>
                <xs:element name="errsourceform_desc" minOccurs="0">
                    <xs:annotation>
                        <xs:documentation>Error Source Form
property</xs:documentation>
                    </xs:annotation>
                    <xs:simpleType>
                        <xs:restriction base="xs:string"/>
                    </xs:simpleType>
                </xs:element>
                <xs:element name="messages" type="messagestype" minOccurs="0">
                    <xs:annotation>
                        <xs:documentation>Message Data
Element</xs:documentation>
                    </xs:annotation>
                </xs:element>
            </xs:sequence>
        </xs:complexType>
        <xs:simpleType name="jobidtype">
```

```
                <xs:annotation>
                        <xs:documentation>Operation Id data element</xs:documentation>
                </xs:annotation>
                <xs:restriction base="xs:int">
                        <xs:minInclusive value="0"/>
                </xs:restriction>
        </xs:simpleType>
        <!--


        Start per-command results types


        -->
        <xs:complexType name="cdresultstype">
                <xs:annotation>
                        <xs:documentation>The element contained in the response to a cd
command.</xs:documentation>
                </xs:annotation>
                <xs:choice>
                        <xs:sequence>
                                <xs:element name="ufip" type="ufiptype"/>
                        </xs:sequence>
                        <xs:element name="help" type="helptexttype"/>
                        <xs:element name="examine" type="examinetexttype"/>
                </xs:choice>
        </xs:complexType>
        <xs:complexType name="createresultstype">
                <xs:annotation>
                        <xs:documentation>The element contained in the response to a create
command.</xs:documentation>
                </xs:annotation>
                <xs:choice minOccurs="0">
                        <xs:sequence>
                                <xs:element name="instance" type="instancepropertytype"
minOccurs="0"/>
```

```
            </xs:sequence>
            <xs:element name="help" type="helptexttype"/>
            <xs:element name="examine" type="examinetexttype"/>
        </xs:choice>
    </xs:complexType>
    <xs:complexType name="deleteresultstype">
        <xs:annotation>
            <xs:documentation>The element contained in the response to a delete
command.</xs:documentation>
        </xs:annotation>
        <xs:choice minOccurs="0">
            <xs:sequence>
                <xs:element name="target" type="targetreferencetype"
minOccurs="0" maxOccurs="unbounded"/>
            </xs:sequence>
            <xs:element name="help" type="helptexttype"/>
            <xs:element name="examine" type="examinetexttype"/>
        </xs:choice>
    </xs:complexType>
    <xs:complexType name="dumpresultstype">
        <xs:annotation>
            <xs:documentation>The element contained in the response to a dump
command.</xs:documentation>
        </xs:annotation>
        <xs:choice minOccurs="0">
            <xs:sequence>
                <xs:element name="source" type="sourcetype"/>
                <xs:element name="destination" type="destinationtype"/>
            </xs:sequence>
            <xs:element name="help" type="helptexttype"/>
            <xs:element name="examine" type="examinetexttype"/>
        </xs:choice>
    </xs:complexType>
    <xs:complexType name="exitresultstype">
```

```
        <xs:annotation>
            <xs:documentation>The element contained in the response to an exit
command.</xs:documentation>
        </xs:annotation>
        <xs:choice minOccurs="0">
            <xs:sequence>
                <xs:element name="help" type="helptexttype"/>
                <xs:element name="examine" type="examinetexttype"/>
            </xs:sequence>
        </xs:choice>
    </xs:complexType>
    <xs:complexType name="helpresultstype">
        <xs:annotation>
            <xs:documentation>The element contained in the response to a help
command.</xs:documentation>
        </xs:annotation>
        <xs:choice>
            <xs:element name="text" type="texttype"/>
            <xs:element name="examine" type="examinetexttype"/>
        </xs:choice>
    </xs:complexType>
    <xs:complexType name="loadresultstype">
        <xs:annotation>
            <xs:documentation>The element contained in the response to a load
command.</xs:documentation>
        </xs:annotation>
        <xs:choice minOccurs="0">
            <xs:sequence>
                <xs:element name="source" type="sourcetype"/>
                <xs:element name="destination" type="destinationtype"/>
            </xs:sequence>
            <xs:element name="help" type="helptexttype"/>
            <xs:element name="examine" type="examinetexttype"/>
        </xs:choice>
```

```
	</xs:complexType>
	<xs:complexType name="resetresultstype">
		<xs:annotation>
			<xs:documentation>The element contained in the response to a reset
command.</xs:documentation>
		</xs:annotation>
		<xs:choice minOccurs="0">
			<xs:sequence>
				<xs:element name="instance" type="instancereferencetype"/>
				<xs:element name="timestamp" type="datetime"
minOccurs="0"/>
			</xs:sequence>
			<xs:element name="help" type="helptexttype"/>
			<xs:element name="examine" type="examinetexttype"/>
		</xs:choice>
	</xs:complexType>
	<xs:complexType name="setresultstype">
		<xs:annotation>
			<xs:documentation>The element contained in the response to a set
command.</xs:documentation>
		</xs:annotation>
		<xs:choice minOccurs="0">
			<xs:element name="association" type="associationtype"/>
			<xs:element name="instance" type="instancepropertytype"/>
			<xs:element name="help" type="helptexttype"/>
			<xs:element name="examine" type="examinetexttype"/>
		</xs:choice>
	</xs:complexType>
	<xs:complexType name="showresultstype">
		<xs:annotation>
			<xs:documentation>The element contained in the response to a show
command.</xs:documentation>
		</xs:annotation>
		<xs:choice minOccurs="0">
			<xs:sequence>
```

```
                    <xs:element name="target" type="targetfullinstancetype"
minOccurs="0"/>
               </xs:sequence>
               <xs:element name="help" type="helptexttype"/>
               <xs:element name="examine" type="examinetexttype"/>
          </xs:choice>
     </xs:complexType>
     <xs:complexType name="startresultstype">
          <xs:annotation>
               <xs:documentation>The element contained in the response to a start
command.</xs:documentation>
          </xs:annotation>
          <xs:choice minOccurs="0">
               <xs:sequence>
                    <xs:element name="instance" type="instancereferencetype"/>
                    <xs:element name="timestamp" type="datetime"
minOccurs="0"/>
               </xs:sequence>
               <xs:element name="help" type="helptexttype"/>
               <xs:element name="examine" type="examinetexttype"/>
          </xs:choice>
     </xs:complexType>
     <xs:complexType name="stopresultstype">
          <xs:annotation>
               <xs:documentation>The element contained in the response to a stop
command.</xs:documentation>
          </xs:annotation>
          <xs:choice minOccurs="0">
               <xs:sequence>
                    <xs:element name="instance" type="instancereferencetype"/>
                    <xs:element name="timestamp" type="datetime"
minOccurs="0"/>
               </xs:sequence>
               <xs:element name="help" type="helptexttype"/>
               <xs:element name="examine" type="examinetexttype"/>
```

```
        </xs:choice>
    </xs:complexType>
    <xs:complexType name="versionresultstype">
        <xs:annotation>
            <xs:documentation>The element contained in the response to a version
command.</xs:documentation>
        </xs:annotation>
        <xs:choice minOccurs="0">
            <xs:element name="text">
                <xs:simpleType>
                    <xs:restriction base="xs:string">
                        <xs:minLength value="1"/>
                    </xs:restriction>
                </xs:simpleType>
            </xs:element>
            <xs:element name="help" type="helptexttype"/>
            <xs:element name="examine" type="examinetexttype"/>
        </xs:choice>
    </xs:complexType>
    <!--
    Start per-option, option argument results types
    These are for elements that are included in results because the corresponding option or
option arg
    was included on the command line
    -->
    <xs:complexType name="examinetexttype">
        <xs:annotation>
            <xs:documentation>element returned when the examine option is used
with a command.</xs:documentation>
        </xs:annotation>
        <xs:sequence>
            <xs:element name="text" type="texttype"/>
        </xs:sequence>
    </xs:complexType>
```

```
        <xs:complexType name="helptexttype">
                <xs:annotation>
                        <xs:documentation>The element contained in the response to a help
command or use of help option on another command.</xs:documentation>
                </xs:annotation>
                <xs:all>
                        <xs:element name="text" type="texttype"/>
                </xs:all>
        </xs:complexType>
        <xs:simpleType name="texttype">
                <xs:annotation>
                        <xs:documentation>Vendor owns all content contained in this
element.</xs:documentation>
                </xs:annotation>
                <xs:restriction base="xs:string">
                        <xs:minLength value="1"/>
                </xs:restriction>
        </xs:simpleType>
        <xs:complexType name="propertycollectiontype">
                <xs:annotation>
                        <xs:documentation>A group of property elements.</xs:documentation>
                </xs:annotation>
                <xs:sequence>
                        <xs:element name="property" type="propertytype" minOccurs="0"
maxOccurs="unbounded"/>
                </xs:sequence>
        </xs:complexType>
        <xs:complexType name="propertytype">
                <xs:annotation>
                        <xs:documentation>A property of a managed
element.</xs:documentation>
                </xs:annotation>
                <xs:sequence>
                        <xs:element name="name">
                                <xs:annotation>
```

```
                    <xs:documentation>desc:  user friendly property name, 
req:</xs:documentation>
                </xs:annotation>
                <xs:simpleType>
                    <xs:restriction base="xs:string">
                        <xs:minLength value="1"/>
                    </xs:restriction>
                </xs:simpleType>
            </xs:element>
            <xs:choice>
                <xs:element name="value">
                    <xs:complexType>
                        <xs:sequence>
                            <xs:element name="val"/>
                            <xs:element name="valstring" 
minOccurs="0"/>
                        </xs:sequence>
                    </xs:complexType>
                </xs:element>
                <xs:element name="multivalue">
                    <xs:annotation>
                        <xs:documentation>Used to represent 
arrays.</xs:documentation>
                    </xs:annotation>
                    <xs:complexType>
                        <xs:annotation>
                            <xs:documentation>ordering of value 
elements corresponds to ordering in array</xs:documentation>
                        </xs:annotation>
                        <xs:sequence>
                            <xs:element name="value" minOccurs="0" 
maxOccurs="unbounded">
                                <xs:complexType>
                                    <xs:sequence>
```

```
                                                                    <xs:element
name="val"/>
                                                                    <xs:element
name="valstring" minOccurs="0"/>
                                                              </xs:sequence>
                                                        </xs:complexType>
                                                  </xs:element>
                                            </xs:sequence>
                                      </xs:complexType>
                                </xs:element>
                          </xs:choice>
                          <xs:element name="type" type="proptype" minOccurs="0"/>
                          <xs:element name="description" type="xs:string" minOccurs="0"/>
                          <xs:element name="readonly" type="xs:boolean" minOccurs="0"/>
                          <xs:element name="maxlen" type="xs:int" minOccurs="0"/>
                          <xs:element name="maxvalue" type="xs:int" minOccurs="0"/>
                          <xs:element name="minvalue" type="xs:int" minOccurs="0"/>
                          <xs:element name="units" type="xs:string" minOccurs="0"/>
                          <xs:element name="valuemap" minOccurs="0">
                                <xs:simpleType>
                                      <xs:list itemType="xs:string"/>
                                </xs:simpleType>
                          </xs:element>
                          <xs:element name="values" minOccurs="0"/>
                    </xs:sequence>
              </xs:complexType>
              <xs:complexType name="verblisttype">
                    <xs:annotation>
                          <xs:documentation>Defines ability to list standard clp verbs followed by
oem verbs</xs:documentation>
                    </xs:annotation>
                    <xs:sequence>
                          <xs:element name="standardverbs" minOccurs="0">
                                <xs:simpleType>
                    <xs:list itemType="clpverbenum"/>
                </xs:simpleType>
            </xs:element>
            <xs:element name="oemverbs" minOccurs="0">
                <xs:simpleType>
                    <xs:list itemType="xs:string"/>
                </xs:simpleType>
            </xs:element>
        </xs:sequence>
    </xs:complexType>
    <xs:complexType name="targetreferencetype">
        <xs:annotation>
            <xs:documentation>Effectively pointer to an instance and an array of
contained targets.</xs:documentation>
        </xs:annotation>
        <xs:sequence>
            <xs:element name="instance" type="instancereferencetype"/>
            <xs:element name="target" type="targetreferencetype" minOccurs="0"
maxOccurs="unbounded"/>
        </xs:sequence>
    </xs:complexType>
    <xs:complexType name="targetfullinstancetype">
        <xs:annotation>
            <xs:documentation>Effectively pointer to an instance and an array of
contained targets.</xs:documentation>
        </xs:annotation>
        <xs:sequence>
            <xs:element name="instance" type="instancetype"/>
            <xs:element name="target" type="targetfullinstancetype" minOccurs="0"
maxOccurs="unbounded"/>
        </xs:sequence>
    </xs:complexType>
    <xs:complexType name="destinationtype">
        <xs:annotation>
```

```
                    <xs:documentation>identifying element for a
destination</xs:documentation>
                </xs:annotation>
                <xs:choice>
                    <xs:element name="ufip" type="ufiptype"/>
                    <xs:element name="uri" type="uritype"/>
                </xs:choice>
        </xs:complexType>
        <xs:complexType name="sourcetype">
                <xs:annotation>
                    <xs:documentation>identifying element for a source</xs:documentation>
                </xs:annotation>
                <xs:choice>
                    <xs:element name="ufip" type="ufiptype"/>
                    <xs:element name="uri" type="uritype"/>
                </xs:choice>
        </xs:complexType>
        <xs:simpleType name="proptype">
                <xs:annotation>
                    <xs:documentation>xs:simpleType that defines enumeration of values for
a property element</xs:documentation>
                </xs:annotation>
                <xs:restriction base="xs:string">
                    <xs:enumeration value="uint8"/>
                    <xs:enumeration value="sint8"/>
                    <xs:enumeration value="uint16"/>
                    <xs:enumeration value="sint16"/>
                    <xs:enumeration value="uint32"/>
                    <xs:enumeration value="sint32"/>
                    <xs:enumeration value="uint64"/>
                    <xs:enumeration value="sint64"/>
                    <xs:enumeration value="boolean"/>
                    <xs:enumeration value="real32"/>
                    <xs:enumeration value="real64"/>
                    <xs:enumeration value="datetime"/>
                    <xs:enumeration value="string"/>
                    <xs:enumeration value="string array"/>
                </xs:restriction>
        </xs:simpleType>
        <xs:complexType name="oemverbresultstype">
                <xs:annotation>
                    <xs:documentation>placeholder element to allow vendor to insert
responses to an oem verb</xs:documentation>
                </xs:annotation>
                <xs:sequence>
                    <xs:element name="verbname">
                        <xs:annotation>
                            <xs:documentation>force vendor to provide at least the
verbname</xs:documentation>
                        </xs:annotation>
                    </xs:element>
                    <xs:choice>
                        <xs:element name="oemdata" type="oemdatatype"/>
                        <xs:element name="help" type="helptexttype"/>
                        <xs:element name="examine" type="examinetexttype"/>
                    </xs:choice>
                </xs:sequence>
        </xs:complexType>
        <xs:complexType name="assocreferencetype">
                <xs:annotation>
                    <xs:documentation>Reference to a ME in an
assocation.</xs:documentation>
                </xs:annotation>
                <xs:sequence>
                    <xs:element name="name" type="xs:string"/>
                    <!-- AEM I think this should just be a ufip.  The association is the subject
of this element, not an instance -->
                    <xs:element name="instance" type="instancereferencetype"/>
                </xs:sequence>
```

```
	</xs:complexType>
	<xs:complexType name="associationtype">
		<xs:annotation>
			<xs:documentation>Type for an association.</xs:documentation>
		</xs:annotation>
		<xs:sequence>
			<xs:element name="ufct" type="ufcttype"/>
			<xs:element name="reference" type="assocreferencetype" minOccurs="2"
maxOccurs="2"/>
			<xs:element name="properties" type="propertycollectiontype"
minOccurs="0"/>
		</xs:sequence>
	</xs:complexType>
	<xs:complexType name="associationcollectiontype">
		<xs:annotation>
			<xs:documentation>group element for associations</xs:documentation>
		</xs:annotation>
		<xs:sequence>
			<xs:element name="association" type="associationtype" minOccurs="0"
maxOccurs="unbounded"/>
		</xs:sequence>
	</xs:complexType>
	<xs:simpleType name="uritype">
		<xs:annotation>
			<xs:documentation>These values MUST comply with
RFC2396.</xs:documentation>
		</xs:annotation>
		<xs:restriction base="xs:string">
			<xs:minLength value="1"/>
		</xs:restriction>
	</xs:simpleType>
	<xs:simpleType name="ufcttype">
		<xs:annotation>
			<xs:documentation>User Friendly class Tag</xs:documentation>
		</xs:annotation>
		<xs:restriction base="xs:string">
			<xs:minLength value="1"/>
		</xs:restriction>
	</xs:simpleType>
	<xs:complexType name="oemdatatype" mixed="true">
		<xs:annotation>
			<xs:documentation>Completely free-form output.  Vendor owns all
content contained in this element.</xs:documentation>
		</xs:annotation>
		<xs:sequence>
			<xs:any minOccurs="0"/>
		</xs:sequence>
	</xs:complexType>
	<xs:simpleType name="clpverbenum">
		<xs:annotation>
			<xs:documentation>Enumeration of the CLP defined
verbs.</xs:documentation>
		</xs:annotation>
		<xs:restriction base="xs:string">
			<xs:enumeration value="create"/>
			<xs:enumeration value="delete"/>
			<xs:enumeration value="dump"/>
			<xs:enumeration value="show"/>
			<xs:enumeration value="set"/>
			<xs:enumeration value="load"/>
			<xs:enumeration value="help"/>
			<xs:enumeration value="reset"/>
			<xs:enumeration value="stop"/>
			<xs:enumeration value="start"/>
			<xs:enumeration value="version"/>
			<xs:enumeration value="exit"/>
			<xs:enumeration value="oemverb"/>
		</xs:restriction>
	</xs:simpleType>
```

```
        <xs:simpleType name="datetime">
                <xs:annotation>
                        <xs:documentation>Restriction for CIM datetime formatted
property.</xs:documentation>
                </xs:annotation>
                <xs:restriction base="xs:string">
                        <xs:pattern value="[0-9]{14}.[0-9]{6}[+-:][0-9]{3}"/>
                </xs:restriction>
        </xs:simpleType>
        <xs:simpleType name="ufiptype">
                <xs:annotation>
                        <xs:documentation>User Friendly instance Path
element.</xs:documentation>
                </xs:annotation>
                <xs:restriction base="xs:string">
                        <xs:pattern value="((/(([a-z])+([0-9])+)?)|(/(([a-z])+([0-9])+)(/([a-z])+([0-
9])+)*))"/>
                        <!--  -->
                </xs:restriction>
        </xs:simpleType>
        <xs:complexType name="ufittype">
                <xs:annotation>
                        <xs:documentation>Type for the ufit element</xs:documentation>
                </xs:annotation>
                <xs:simpleContent>
                        <xs:extension base="ufitsimpletype">
                                <xs:attributeGroup ref="ufitattrs"/>
                        </xs:extension>
                </xs:simpleContent>
        </xs:complexType>
        <xs:attributeGroup name="ufitattrs">
                <xs:annotation>
                        <xs:documentation>Attribute group for the UFiT
element.</xs:documentation>
                </xs:annotation>
                <xs:attribute name="ufct" use="required">
                        <xs:simpleType>
                                <xs:restriction base="xs:string">
                                        <xs:pattern value="([a-z])+"/>
                                </xs:restriction>
                        </xs:simpleType>
                </xs:attribute>
                <xs:attribute name="instance" use="required">
                        <xs:simpleType>
                                <xs:restriction base="xs:string">
                                        <xs:pattern value="([0-9])+"/>
                                </xs:restriction>
                        </xs:simpleType>
                </xs:attribute>
        </xs:attributeGroup>
        <xs:simpleType name="ufitsimpletype">
                <xs:annotation>
                        <xs:documentation>Force the content of UFiT to be minimum length
one</xs:documentation>
                </xs:annotation>
                <xs:restriction base="xs:string">
                        <xs:minLength value="1"/>
                        <xs:pattern value="([a-z])+([0-9])+"/>
                </xs:restriction>
        </xs:simpleType>
</xs:schema>
```