Document Number: DSP0214

Date: 2007-03-07

Version: 1.0.2

Server Management Command Line Protocol (SM CLP) Specification

Document Type: Specification

Document Status: Final Standard

Document Language: E

CONTENTS

Figures

Tables

Foreword

The *Server Management Command Line Protocol (SM CLP) Specification* (DSP0214) was prepared by the Server Management Working Group. This document was prepared in accordance with *ISO/IEC Directives, Part 2: Rules for the structure and drafting of International Standards*.

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

DMTF is a not-for-profit association of industry members dedicated to promoting enterprise and systems management and interoperability.

Introduction

This section contains an overview of the Command Line Protocol (CLP). This includes the goals behind creating the CLP and the specific problems that it attempts to resolve. In addition, this section lays 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.

Problem Statement

The fundamental problem that 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 an interoperable, heterogeneous, enterprise-wide management solution. By extending the DMTF specifications to include a CIM-based command line protocol for managing systems and devices, the DMTF comes closer to realizing its vision of enabling end-to-end, multi-vendor interoperability in management systems.

Principal Goals

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

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 *Simple Mail Transfer Protocol* (RFC2821).

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 because they can be in any specific language. In addition, the CLP input (commands and syntax) is not translated because CLP syntax is itself its own language.

The CLP allows for extensibility through four different mechanisms: verbs, options and option argument terms, command target terms, and target property terms. The conventions contained herein allow for implementers to extend the interface in a non-conflicting mechanism that allows for differentiation and experimentation without encroaching on the standard CLP syntax and semantics.

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 Clause 6. 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`", and "`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 *Server Management Managed Element (SM ME) Addressing Specification* (DSP0215). 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. It can also be an Association. 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 shall be preceded by a hyphen ("-"). They provide features such as changing the output format, allowing the command to apply to nested levels, requesting that 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 Clause 7.

Command target properties are attributes that may contain values associated with a target that are needed to process the command. Command target properties identify properties of the target's class that are to be retrieved or modified by the command. Valid command target property names are documented in the MOF file that defines the class. Implementations shall 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 Clause 6. If a value is to be assigned to a property, the syntax shall be of the form "<property name>=<value>". There may be zero or more property terms per command.

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* (DSP0216) 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 DSP0216.

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 the *SM Architecture White Paper* (DSP2001).

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 DSP2001. An implementation of the CLP service is modeled using the *Command Line Protocol (CLP) Service Profile* (DSP1005).

Physical Connection

The physical connection and media between the Client and the MAP are outside the scope of this specification. Any physical connection that is capable of running one of the supported transports is assumed to be able to support the CLP. Because 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 that support IP. In fact, it is reasonable for an implementation to be created in which the protocol never leaves the managed server.

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 the scope of this specification.

The Client is the terminus of one end of the connection; the CLP Service implementation is the other. CLP Service implementations manage the transport and the sessions that occur over the transport as required by DSP1005.

Authentication, Authorization, and Encryption

The CLP Service does not perform any authentication or encryption. It relies entirely on the transport to perform these functions. Session transport requirements are documented in 8.3.

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 8.3.

The CLP Service authorizes commands through the use of authorization groups. Each CLP User shall be a member of at least one CLP Group. For more information, see 8.1.

The CLP contains commands for the creation, removal, and modification of user accounts, including authorization and access rights. For more information, see Clause 6.

Sessions

Sessions between a Client and a CLP Service are established over a transport protocol. After 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 will maintain a session context and session characteristics as required by DSP1005. Examples of these characteristics include the Current Default Target, currently selected output mode, current output language, and the current user and session identifier. Commands for manipulating the session characteristics are included in the CLP. For more information, see 8.2.2.

Input Editing

The CLP is a command/response protocol. CLP implementations shall receive and parse an entire Command Line, complete with verb, command target term, options, and properties. The CLP Service shall 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 this specification.

Command Line Protocol Service

The CLP Service is responsible for providing and enforcing the syntax and semantics of the CLP. Implementations will support being managed as required by DSP1005. This includes starting, stopping, and changing the attributes of the service.

CLP Service Access Point

The transport session is established to the CLP Service Access Point (SAP). 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 SAP. Implementations will support managing the supporting protocol stacks as required by DSP1005.

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 shall 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 5.1.6.

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, and 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, provisioning, or both. It is not intended that the CLP Service be the primary interface for managing mass storage, because these standards and access points exist in the industry.

The CLP also includes the ability to select, control, and initiate the transfer of images. Also provided is 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 shut down a server), and mechanisms to initiate a dump of the operating system.

Access to some system resources is also included in the CLP. This includes access and manipulation of any accessible logs; the ability to view and set remote status displays, LEDs, 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.

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, 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 use to manage hardware. Consequently, the format of the commands and their responses, as well as the CIM methods, properties, and operations supported, are not always sufficient for the CLP Service Access Point to be the primary interface for advanced server management software.

Server Management Command Line Protocol (SM CLP) Specification

1 Scope

This document lays out the general framework for the Server Management Command Line Protocol (SM CLP). This specification is intended to guide developers of implementations of the SM CLP and optionally be used as a reference by system administrators and other users of SM CLP implementations.

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

2 Normative References

The following referenced documents are indispensable for the application of this document. For dated references, only the edition cited applies.

DMTF, *Common Information Model (CIM) Schema, version 2.12*, April 20, 2006

DMTF, DSP0215, *Server Management Managed Element (SM ME) Addressing Specification v1.0*, November 10, 2006

DMTF, DSP0224, *Server Management Command Line Protocol (SM CLP) Command Response XML Schema, v1.0*, 2006

DMTF, DSP1005, *Command Line Protocol (CLP) Service Profile, v1.0*, October 10, 2006

IETF, RFC2234, *Augmented BNF for Syntax Specifications: ABNF*, November 1997

IETF, RFC2396, *Uniform Resource Identifiers (URI): Generic Syntax*, August 1998

ISO 639-2, *Codes for the Representation of Names of Languages Part 2: Alpha-3 Code*, March 2002

ISO, *ISO/IEC Directives, Part 2: Rules for the structure and drafting of International Standards*, Fifth edition, 2004

3 Terms and Definitions

For the purposes of this document, the following terms and definitions apply.

3.1
Absolute Target Address
a designation of target address that begins at the root of the containment hierarchy

3.2
Addressing Associations
an association instance that is used by the MAP to construct the UFiP to an instance referenced by the association instance

3.3
Admin Domain
the set of Managed Elements, Logical Devices, and Services for which the MAP has management responsibilities

3.4
Association
a relationship between two Managed Elements, which is itself a manageable entity within the Admin Domain

3.5
Association Class
identifies the CIM Class of an Association

3.6
Client
any system that acts in the role of a client to a MAP

3.7
CLP Service
the logical entity within a MAP that implements the CLP

3.8
CLP Target
a Managed Element or Association whose properties, behavior, UFcT, and so on are wholly defined by the profiles approved for use with the CLP

3.9
Command Line Protocol
CLP
the human-oriented command line protocol defined by the System Management Architecture for Server Hardware, used for managing systems

3.10
Command or Command Line
a text message containing the complete expression of a management action, including a command verb, an optional command target, options and option arguments, and properties and values

366 **3.11**
367 **Command Processor**
368 the logical entity within a MAP responsible for parsing, interpreting, and executing incoming commands
369 and returning responses

370 **3.12**
371 **Command Response**
372 response returned by the CLP Service to a Client when a Command is issued
373 The Command Response consists of Command Status and Command Results.

374 **3.13**
375 **Command Results**
376 the actual results of a successful command returned as part of the Command Response

377 **3.14**
378 **Command Status**
379 information returned by the CLP Service to a Client describing the overall status of a Command

380 **3.15**
381 **Command Status Data**
382 detailed information returned by the CLP Service to a Client describing the status of the Command as
383 part of the Command Response

384 **3.16**
385 **Core Properties**
386 properties for which the profile that owns the definition of the class does not stipulate any behavioral
387 requirements
388 However, the properties are defined in the MOF.

389 **3.17**
390 **Current Default Target**
391 **CDT**
392 the CLP session environment setting that establishes a default base address for all command targets that
393 are expressed as a Relative Target Address and is used as the command target if a target term is not
394 specified in a command entered

395 **3.18**
396 **Implicit Command Target**
397 target acted upon that is inherent to the command being executed
398 The command does not act upon either the Current Default Target or a target specified as part of the
399 command. The `cd` command is an example of a command that acts upon an Implicit Command Target.

400 **3.19**
401 **Keyword**
402 a text-string token that is recognized and reserved by the CLP to have a specified meaning when used in
403 command output and input

404 **3.20**
405 **Local Addressing Service**
406 the entity responsible for discovering, enumerating, and determining the addresses of Managed Elements
407 and Associations within the Admin Domain

3.21
Manageability Access Point
MAP
a service of a system that provides management in accordance with specifications of the DMTF System Management Architecture for Server Hardware

3.22
Managed Element
ME
the finest granularity of addressing, which can be the target of commands or messages, or a collection thereof

3.23
Managed Element Access Method
the method by which a Managed Element performs a unit of work

3.24
Managed System
a collection of Managed Elements that compose a computer system for which the MAP has management responsibilities

3.25
Management Service Core
the logical entity that contains the core services set of the MAP

3.26
Non-addressing Association
an association instance that is not used by the MAP in constructing the UFiP to any instances referenced by the association instance

3.27
OEM Properties
properties added to instances of a class by an OEM vendor

3.28
OEM Target
a Managed Element or Association whose properties, behavior, UFcT, and so on are outside the scope of this specification and are vendor dependent

3.29
OEM Verbs
verbs defined by an OEM vendor that are outside the scope of this specification

3.30
Operation
an identifiable activity of a MAP

3.31
Option
a term of the Command Line that selects a particular behavior of a command verb

3.32
Option Argument
input data passed to the command verb in relation to an option that selects a particular value for that option

3.33
Property
an attribute of the command target

3.34
Relative Target Address
a designation of target address in relation to the Current Default Target as opposed to an Absolute Target Address

3.35
Required Properties
properties for which the profile that owns the definition of the class requires that instances of the class have values

3.36
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

3.37
Reserved Target
a valid value for a target term that has a special meaning defined by this specification
The meaning of a Reserved Target cannot be changed by a user nor can additional Reserved Targets be created.

3.38
Resultant Address
the Target Address after applying the target address precedence rules

3.39
Resultant Target
the effective target for a command after applying the target address precedence rules

3.40
Service Access Point
SAP
the representation of the endpoint or interface into a service, such as the CLP

3.41
SM CLP Verb
a verb defined by this specification

3.42
Target
the Managed Element or Association upon which a command acts

3.43
Target Address
a string value used as the target term in a Command Line to identify the target for a command

3.44
Target Class Addressing
a method of selecting Managed Elements within a container based on the UFcT of the element

3.45
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 are Telnet and Secure Shell.

3.46
Transport
the layers of the communication stack responsible for reliable transportation of commands and messages between the Client and the MAP

3.47
User-Friendly instance Tag
UFiT
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.

3.48
User-Friendly selection Tag
UFsT
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 *.

3.49
User-Friendly class Tag
UFcT
a short, user-friendly alias for a CIM class name
It has the same properties and methods as the CIM class it represents.

3.50
User-Friendly instance Path
UFiP
the unique path to an instance formed by concatenating the UFiTs of each instance from the root instance to the terminating instance

3.51
Verb
the string name of a command, used as the first term of a Command Line

4 Symbols and Abbreviated Terms

The following symbols and abbreviations are used in this document.

4.1
CDT
Current Default Target

4.2
CIM
Common Information Model

4.3
CLP
Command Line Protocol

4.4
MAP
Manageability Access Point

4.5
ME
Managed Element

4.6
NIC
Network Interface Card

4.7
NVT
Network Virtual Terminal

4.8
OEM
Original Equipment Manufacturer

4.9
SMASH
System Management Architecture for Server Hardware

4.10
SSHv2
Secure Shell Version 2

4.11
UFcT
User-Friendly class Tag

4.12
UFiT
User-Friendly instance Tag

4.13
UFiP
User-Friendly instance Path

4.14
UFsT
User-Friendly selection Tag

4.15
WBEM
Web Based Enterprise Management

5 Server Management Command Line Protocol (SM CLP) Specification

The following clauses detail the requirements for the SM CLP.

5.1 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.

5.1.1 Command Verb

A CLP command verb retrieves information about a target or initiates a state change of the target. A CLP interaction shall consist of one and only one command verb.

5.1.2 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 shall not redefine the usage of a CLP option name across different CLP command verbs.

5.1.3 Command Target

This clause details requirements related to the usage and interpretation of a command target.

5.1.3.1 General

The command target 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 either is dictated by the command verb itself or is referenced from the session environment variable "Current Default Target". Implementations shall interpret command verbs submitted to the CLP only for the Resultant Target. 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 shall not define Reserved Targets beyond the ones defined in this specification. Implementations shall interpret Reserved Targets in accordance with the meaning assigned to them by this specification.

This version of the SM CLP Specification supports the *Server Management Managed Element (SM ME) Addressing Specification v1.0* (DSP0215).

5.1.3.2 Current Default Target

A Current Default Target address shall always be in effect during a CLP session. This 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 5.1.3.3.

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 shall set the Current Default Target to the instance address of the root of the address space of the MAP upon CLP session activation. Implementations shall use the same initial value for the CDT for all users and shall 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 implementation shall set the CDT to the UFiP of the Managed Element that represents the root of address space of the MAP.
- The Command Processor shall 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 the 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, and the implementation shall not change the Current Default Target and shall return a Command Status of COMMAND EXECUTION FAILED and a CIM Status of CIM_ERR_NOT_FOUND.
- If a Managed Element becomes unresponsive at some point after it has been set as the Current Default Target, then the implementation shall return an appropriate error code and shall 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. Because unpredictable, undesired results would occur if the Current Default Target is automatically changed, the Command Processor shall not automatically change the value of the Current Default Target, for any reason.

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 with a command, 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.

5.1.3.3 Target Address Precedence

The implementation shall 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 shall be selected as the Resultant Target.
2) If a command target term is specified in the Command Line, the implementation shall apply the Target Address Evaluation Rules to derive the Resultant Target. These rules are detailed in 5.2.1.3.6.
3) If the Command Line did not include a command target term, the implementation shall select the CDT as the Resultant Target.

If the Resultant Target is determined by the implementation to be invalid, then the implementation shall not execute the command and shall 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 (for example, the `cd` command) or may not accept a command target term (for example, the `exit` command). Each command's use of the command target term is documented in the subclause of Clause 6 devoted to the command.

5.1.3.4 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 *4HCommon Information Model (CIM) Schema, version 2.12*.

The CLP defines a set of general command verbs 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", and so on. 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 Clause 6 describe each CLP command verb in detail.

DSP0216 describes the full mapping of the CLP to the CIM. For each CIM class, DSP0216 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 5.1.4.

- Contained Targets

  This is the set of Managed Elements immediately contained in the Managed Element according to the rules of instance containment described in 5.1.3.5.

- Associations

  This is the set of associations that reference the Managed Element. They are described in more detail in 5.1.5.

- Verbs

  This is the set of commands that are applicable to the Managed Element. The SM CLP verbs are described in Clause 6.

5.1.3.5 Target Addressing

CLP target addressing is defined by DSP0215. CLP implementations shall operate only on command target terms that adhere to DSP0215 or to the rules for identifying OEM targets described in 5.2.6.

The specific arrangements of Managed Elements that a MAP may expose are documented in DSP0215 and SMASH Implementation Requirements (DSP0217). 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, and so on are wholly defined by the profiles approved for use with the CLP. OEM Targets are Managed Elements whose properties, behavior, UFcT, and so on are outside the scope of the profiles approved for use with the CLP and are vendor dependent.

5.1.3.6 Aggregated Targets

Command targets may be an aggregation of underlying components. These underlying components may be visible in the address space of the MAP. When the command target is composed of aggregated parts, the Command Processor shall interpret the command for the aggregated target as a single job and return a Command Response accordingly.

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 shut down 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.

5.1.3.7 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 targets with a single command is Target Class Addressing.

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

Implementations shall support Target Class Addressing for the `show` command. Implementations may support Target Class Addressing for the `create` and `delete` commands. Implementations shall not support Target Class Addressing for CLP commands other than the `show`, `create`, and `delete` commands. When Target Class Addressing is utilized for a command, the implementation shall select the instances to be the target of the command by using the Rules for Selecting Instances by UFcT defined in DSP0215, where the Selection UFcT is the UFcT identified by the UFsT specified in the command target term.

EXAMPLE The command target term "`/system3/disk*`" instructs the Command Processor to issue the command for all targets with an UFcT of "`disk`" in the container "`/system3`".

5.1.4 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 5.2.6 of this specification. The SM CLP recognizes three categories of properties:

- Required Properties are properties that the profile defining the class of the target deems required for compliance with the profile. These properties will be present for the instance across implementations.
- Core Properties are properties that are defined for the class of the target in the CIM schema. The profile that defines the class does not require these properties; however, they may be used because they are defined in the MOF. Note that 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 properties will not be consistent across different vendors' implementations.

5.1.5 Associations

DSP0215 specifies the Association Classes that may be used to construct paths to address any Managed Element appearing within the scope of the MAP. DSP0215 identifies these as Addressing Associations. Additional associations that are not used for addressing may exist and express relationships between Managed Elements. DSP0215 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. 5.2.1.3.5 describes how to use the association separator "=>" to address association instances. 6.10 and 6.11 illustrate the use of the `set` and `show` commands, respectively. Associations have properties which follow the rules for Command Target Properties as identified in 5.1.4.

5.1.6 Command Processing

This clause states the requirements for the processing of a CLP Command Line.

5.1.6.1 General

Implementations of the CLP shall 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 shall validate every Command Line against the `clp-command-line` production of the grammar specified in Annex A. When a Command Line does not comply with the `clp-command-line` production of the grammar defined in Annex A, the implementation shall return a Command Status of COMMAND PROCESSING ERROR and a Processing Error of COMMAND SYNTAX ERROR. There are error conditions identified by this specification that can be detected by validation against the grammar for which this specification identifies specific values for Processing Error that are required to be returned instead. When specified, the requirement to return specific values supersedes the requirement to return the general COMMAND SYNTAX ERROR.

EXAMPLE The `create` command requires a command target term to be specified. The specific Processing Error of MISSING REQUIRED TARGET is required to be returned if the command target term is not included. The grammar also requires that a command target term be specified in the `create-cmd` production. Thus, if a command target term is not specified, the Command Line will fail validation against the grammar and a COMMAND SYNTAX ERROR would be appropriate. However, this general Processing Error is superseded by the specific Processing Error specified for the error condition.

5.1.6.2 Job Visibility

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. 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 shall not expose these jobs through the CLP. The command job shall 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 Time Before Removal has not expired.

By default, the implementation shall 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 if a command takes 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 are implementation specific.

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

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

The implementation shall 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 shall 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 shall be the Instance Suffix for the UFiT of the instance of CIM_ConcreteJob used to represent the job in the job queue. After the Command Status holding time has expired, the corresponding job is deleted and the Job Identifier is released. Implementations shall also implement a user-controlled holding time for Command Status, controlled by a per-command option. Use of this option is documented in **Error! Reference source not found.**. Every command job has a Time Before Removal associated with it. The Time Before Removal indicates the amount of time that a command job is managed by the MAP after completion. The command job itself is

795 represented with an instance of CIM_ConcreteJob. The Time Before Removal of the command job is
796 modeled with the TimeBeforeRemoval property of the CIM_ConcreteJob instance.

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

802 #### 5.1.6.3 Error Handling

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

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

811 Implementations shall include all detected syntax errors first in the Command Status data.
812 Implementations shall include all detected semantic errors in the Command Status data.

813 If the Command Processor detects a syntax error, the implementation shall report an error and the
814 implementation shall not alter the state of the Target. If the Command Processor detects a semantic error,
815 the implementation shall report an error and should not alter the state of the Target.

816 Command Status output is defined by the CLP and is documented in 5.2.2.

817 Command Results output is defined for each CLP command and is documented in Clause 6.

818 ### 5.1.7 SESSION Reserved Target

819 Sessions with the CLP Service are represented as Managed Elements within the address space of the
820 MAP. This enables users of the CLP to manage attributes of the session using standard CLP commands.
821 Users will frequently wish to manage attributes of their own session with the CLP Service. To simplify
822 accessing the Managed Element that represents the user's session, the CLP defines a reserved keyword
823 "SESSION". Implementations shall interpret the keyword "SESSION" as the fully qualified path to the
824 Managed Element that represents the session of the user issuing the command.

825 ### 5.1.8 UFiT Assignment

826 Individual Managed Elements within the address space of the map are identified by a UFiT. The UFiT is
827 constructed by concatenating an integer suffix to the UFcT for the class of the Managed Element. The
828 rules for assigning and maintaining UFiTs are defined in DSP0215.

829 ### 5.1.9 Input Data

830 This clause states requirements for the handling of input data.

831 #### 5.1.9.1 General

832 Implementations of the CLP shall not allow inclusion of input data embedded in the text session
833 (sometimes referred to as "streaming"). A data file or stream can be selected for input to a CLP command
834 by reference only using a command-specific option.

835 EXAMPLE To input a firmware image for reloading firmware, the option `-source <URI>` is used.

5.1.9.2 Data Passed in on Command Line

Data may be provided as input to a command through an option argument, option argument value, 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. Implementations shall 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 shall not execute the command and shall return a Command Status of COMMAND PROCESSING FAILED and a Processing Error of COMMAND SYNTAX ERROR.

5.1.10 Output Data

This clause states requirements related to output data.

5.1.10.1 General

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 shall return a Command Response data element as the response before any other text that is appended. Implementations shall include a Command Status data element in the Command Response data element and may include a Command Results data element. Implementations shall 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 5.2.4 and 5.2.1.1. 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, selection of one of the CLP's structured output formats is recommended. These formats are "keyword" and "clpxml". Implementations shall 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 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 is likely to 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 target after 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 shall 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 5.2.2.

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

5.1.10.2 Command Status Data Elements

This clause states requirements related to Command Status Data Elements.

5.1.10.2.1 General

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 shall return the Command Status data element as the first data element in a Command Response. Implementations shall 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 are listed in Table 4.

Processing Errors are conditions detected by the Command Processor prior to creating a job to execute the command. These are generally syntax-related errors. Two properties are 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 is in Table 6.

Implementations may support the Status Tag and Processing Error Tag properties.

5.1.10.2.2 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 shall include the Message, Owning Entity, and Message Id properties, and may include one or more Message Argument properties. Implementations shall 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.

To guarantee uniqueness, the Owning Entity property shall include a prefix string that uniquely identifies the entity that owns and defines the Owning Entity value. The prefix string shall 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 shall not return a status message that exceeds 256 characters. Implementations may support the Message data element.

5.1.10.2.3 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 shall 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 shall 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 5.2.3.2.

The implementation shall 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 shall 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 shall include the Processing Error property in the Command Status data element.

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

When the Command Status is COMMAND PROCESSING FAILED, the implementation shall not include the Job data element in the Command Status data element.

When the Command Status is COMMAND SPAWNED, the implementation shall include the Job data element in the Command Status data element and shall not include the Job Error data element in the Job data element.

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

5.1.10.3 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 Clause 6 for command specifications and their corresponding Command Result schema.

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

See DSP0216 for the keywords and value specifications that are relevant to each class of the target Managed Element.

5.1.11 Session Prompt

CLP session output shall 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 shall be of the following form:

```
<OUTPUT><IMP PROMPT><CLP PROMPT>
```

where

- **OUTPUT** is the Command Response. The Command Response shall be formatted according to one of the CLP-defined output formats.
- **IMP PROMPT** is the implementation's optional prompt string, which shall not contain the CLP PROMPT string.
- **CLP PROMPT** is the CLP standard prompt string, "`-> `". The CLP prompt string shall appear after each Command Response.

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

Implementations may omit an implementation-provided prompt string. If the implementation provides an implementation prompt string, the implementation shall insert the prompt string 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. 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 shall return a response to each command presented by the Client. Implementations of the CLP Service shall not accept any further commands from the Client until after the implementation returns a Command Response for the currently outstanding command.

Because the CLP is primarily for use by a human user, the CLP Service shall return a Command Response within a "reasonable amount of time". The CLP Service shall be capable of spawning any commands that it determines to be long running. When commands are spawned, the CLP Service shall 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.

5.1.12 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 5.2.6 so that the user is aware that the use of the extension is non-standard. The syntax clause of this document defines areas of vendor extensibility and the requirements in effect for those extensions.

5.2 Syntax

The CLP implements a small and easily remembered set of verbs (Clause 6) that apply across the entire target address space (5.1.3). This allows users 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 (5.1.10), which further simplifies understanding by both the new and experienced user as they move from implementation to implementation and from simple to complex behaviors. As users become 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 (Clause 7).

5.2.1 Basic Command Syntax

The SM CLP basic command syntax is described in the following clauses.

5.2.1.1 Character Set, Delimiters, Special, and Reserved Characters

All implementations of the CLP shall interpret the characters provided by the transport as UTF8 representation of the characters, including those in Table 1, and shall 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), used in front of reserved characters to instruct the command parser to use the reserved character without special meaning. When the escape character is not followed by a reserved character, it is treated as a normal character in the string that contains it.
<cr> <lf> <cr><lf>	end-of-line	Each of these sequences is accepted as an end-of-line indicator.

Character or Sequence	Name	Description / Uses
<escape character><end-of-line>	line continuation	An escape character placed immediately before the end-of-line sequence indicates that the current line is continued to the following line. The following line will be appended to the current line.
,	comma	Delimits items in an option argument term that is to be interpreted as a list of option arguments. Also delimits values for an option argument.
=	assignment operator	A single equal sign '=' is used to separate a property name from a desired value for the property when used with verbs that modify or create an instance. It will not have a space before or after it in an expression of a property and its value.
==	equivalence operator	Two consecutive equal signs "==" without white space between them are used to separate a property name from a desired value when filtering instances for which results are returned.
-	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 command target term that identify the instances the association references.
.	dot	Recognized as a special target address token meaning "this container".
..	dot-dot	Recognized as a special target address token meaning "the container of this container".
( )	parentheses	In an option argument term that 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.

5.2.1.2 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 uppercase and lowercase characters. Implementations shall accept command verb, option, option argument, target, and property names expressed in any combination of uppercase and lowercase characters.

EXAMPLE "show", "Show", and "SHOW" are all valid expressions of the CLP verb "show". "-o Format=clpxml", "-O format=clpxml", and "-OUTPUT FORMAT=clpxml" are all valid expressions of the output option and format argument.

For readability, this specification documents all verb, option, target, and property names in lowercase.

The CLP places no restrictions on case sensitivity for the interpretation of property values. Interpretation of property values (including case sensitivity) is determined by the profile that defines the target to which the property belongs. Requirements for the input format of common property types are specified in DSP0216.

5.2.1.3 Command Line Terms

This clause details the requirements for Command Line Terms.

5.2.1.3.1 General

A CLP Command Line consists of four types of terms: verbs, options and option argument terms, command target terms, and target property terms. Each term on the Command Line is separated from other terms by the CLP command term separator character, " " (space). (See 5.2.1.1 for the list of CLP special and reserved characters.) Implementations shall recognize the CLP command term separator character, beginning of line, and end of line as delimiting terms on the Command Line.

Implementations shall 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. The implementation shall not initiate command processing until after the complete command has been received by the CLP. If an implementation receives a Command Line that contains zero characters or consists entirely of the command term separator character, implementations shall return a Command Response that contains exactly zero characters. The effect of this requirement is such that a blank line is not reported as an error and instead results in a CLP prompt being returned.

5.2.1.3.2 Verb

The implementation shall expect the command verb to be the first term in a command. The implementation will expect the command verb to be one of the specified CLP command verbs listed in Clause 6 or an OEM command line extended form as defined in 5.2.6.3.3. Implementations shall not support any verbs other than the CLP verbs defined in this specification and any command verbs identified as OEM verbs according to 5.2.6.3.3. When the first term in a Command Line is not a CLP verb and is not identified as an OEM verb according to the rules in 5.2.6.3.3, the implementation shall not execute the command and shall return a Command Status of COMMAND PROCESSING FAILED and a Processing Error of COMMAND NOT RECOGNIZED.

5.2.1.3.3 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 shall occur immediately after the command verb and before the optional command target term and target properties.

Command options either require an argument or require no argument.

Options that require no argument are separated from the subsequent options, command target term, or target property names by the command term separator character. Options that require arguments are separated from their option argument term by the command term separator character. An option argument will be one or more argument names or argument/value pairs. The comma is used to delimit arguments and argument/value pairs within the option argument term. The comma is also used to delimit values within an argument value. Using the comma for two types of tokenization within the option argument term could result in ambiguity when parsing an option argument term. To eliminate the potential for ambiguity, parentheses are used to enclose argument values which can be comma-delimited lists. When processing an option argument term, implementations will tokenize the option argument term using the comma as a delimiter unless the comma is enclosed in parentheses, in which case the implementation will ignore it. Implementations shall interpret a "," (comma) as delimiting arguments in the option argument term unless the comma is preceded by a left parenthesis "(", in which case the implementation shall ignore any commas that occur prior to a matching right parenthesis ")". If while parsing a Command Line an implementation encounters an option it does not recognize, the implementation shall not execute the command and shall return a Command Status of COMMAND PROCESSING FAILED and a Processing Error of INVALID OPTION.

5.2.1.3.4 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 5HDSP0215. Instance addressing is extended to include support for Relative Target Addresses. It adds support for addressing instances of a particular class and support for addressing an instance or instances of an association relative to target instances that the association references.

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

The implementation shall observe the following ordered rules for determining if the first non-option or option argument term after the verb is treated as a command target term:

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

These rules enable a Client wishing to ensure consistent results when specifying a command target term to do so through the inclusion of one of the CLP reserved addressing terms. When the implementation determines that a command target term has been specified, the implementation shall validate the command target term against the all-legal-targets production of the grammar specified in Annex A. If the command target term is invalid, the implementation shall return a Command Status of COMMAND PROCESSING FAILED and a Processing Error of INVALID TARGET.

The order of precedence for determining the Resultant Target of a command is defined in 5.1.3.

5.2.1.3.5 Target Address Syntax

Target addresses in the SM CLP are 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 command target 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 Class identifies 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 that 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 Annex A.

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 command target term and performing any substitutions necessary, it is possible to create a UFiP identifying the Managed Element that is the 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 as follows:

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

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

```
[[<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 precede 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 second occurrence of "=>" and following additional target address terms are optional. The second "=>" 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 second set of address terms is omitted, the implementation will return all instances of the Association Class that reference the instance addressed on the left-hand side.

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

```
/
/system1
\system1
system1
/system1/alarm3
alarm3
../rack3
hw1/./../../rack3
..
.
/system1=>AssociatedPowerManagementService=>/system1/service24
../system1/cpu1=>SystemDevice=>/system1
```

```
../system1/cpu1=>SystemDevice
```

5.2.1.3.6 Target Address Evaluation

CLP commands fall into two categories: commands that accept a command target term and commands that do not. Not all commands that accept a command target require one to be included. The rules of precedence that govern choosing among an Implicit Command Target, the CDT, and a command target term are detailed in 5.1.3.3. This clause describes the rules for evaluating the command target term if it is specified.

The command target term can be either a Relative Target Address or an Absolute Target Address. The command 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.

5.2.1.3.6.1 Rules for Addressing a Specific Association

If the command target term includes exactly two occurrences of the association separator, the implementation shall evaluate the command target term according to the following rules:

- The implementation shall interpret the characters between the two occurrences of the association separator as identifying the Association Class.
- The implementation shall interpret all characters prior to the first occurrence of the association separator as a single token. The implementation shall evaluate the token according to the "Rules for Addressing a Target Instance" (5.2.1.3.6.4).
- The implementation shall interpret all characters after the second occurrence of the association separator as a single token. The implementation shall evaluate the token according to the "Rules for Addressing a Target Instance" (5.2.1.3.6.4).

The implementation shall set the Resultant Target for the command to the association instance of the type specified by the Association Class such that the association references the Managed Element identified by the instance address preceding the first association separator and the association references the Managed Element identified by the instance address following the second occurrence of the association separator.

5.2.1.3.6.2 Rules for Addressing Instances of an Association

If the command target term includes exactly one occurrence of the association separator, the command target term is assumed to be targeting all instances of a particular Association Class that reference a Managed Element instance, and the implementation shall evaluate the command target term according to the following rules:

- The implementation shall interpret the characters between the occurrence of the association separator and the Command Line term separator as identifying the Association Class.
- The implementation shall interpret all characters prior to the first occurrence of the association separator as a single instance address. The implementation shall evaluate the instance address according to the "Rules for Addressing a Target Instance" (5.2.1.3.6.4).

The implementation shall 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 preceding the first occurrence of the association separator in the command target term.

5.2.1.3.6.3 Rules for Addressing a Target Instance/Class

If the command target term does not include any occurrences of the association separator, the implementation shall evaluate the command target term according to the "Rules for Addressing a Target Instance" (5.2.1.3.6.4).

5.2.1.3.6.4 Rules for Addressing a Target Instance

Implementations shall 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 shall interpret an Absolute Target Address as relative to the Managed Element instance that 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 shall interpret a Relative Target Address as relative to the Current Default Target and shall 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 shall 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 preceding UFiT, if a preceding 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 preceding rules, the instance address term will be a UFiP and is the Resultant Address produced by applying the rules for addressing a target instance. Note that after applying these rules, it is possible that the Resultant Address will consist of a single address term separator character. Implementations shall interpret this Resultant Address as equivalent to the UFiP of the Managed Element that is the root of the address space.

5.2.1.3.7 Target Properties

This clause specifies constraints for target properties.

5.2.1.3.7.5 General

Many CLP verbs accept target property terms as input to the command. Target property terms always contain a target property name and optionally contain the assignment or equivalence operator followed by a property value. Implementations shall interpret terms appearing in the Command Line after the command target term as target property terms. Implementations shall interpret target property names in a case-insensitive manner.

When the command target term is omitted, implementations shall interpret any non-option name terms as target property terms.

When a structured output is specified, the implementation shall 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 "Rules for Specifying Target Property Values" (5.2.1.3.7.1). There are three types of target property terms: terms that include the assignment operator, terms that include the equivalence operator, and terms that do not include either. Terms that include the assignment operator are used to indicate a desired value to assign to a property and are interpreted according to "Using the Assignment Operator" (5.2.1.3.7.3). Terms that include an equivalence operator are used to indicate a property name and desired value for the property when filtering for an instance with that property and are interpreted according to the rules in "Using the Equivalence Operator" (5.2.1.3.7.4).

5.2.1.3.7.1 Rules for Specifying Target Property Values

A CLP implementation will accept target property values as part of a target property term. They can be used with some CLP verbs (`create` and `set`) to specify a value to assign to a property or with some

CLP verbs and options (`show` and `display`) to filter results based on a property/value match. When a user specifies a target property value on the Command Line, the implementation shall enforce the following syntax:

> If the property value contains a CLP reserved character, the value is enclosed in quotes. If the property value includes a " (double quote) character, the " (double quote) is escaped using the CLP escape character.

The specific format of the value for a property is defined in DSP0216. Note that in the case of a property that is a Value/ValueMap, the string supplied as a value to the property for assignment could be the string representation of the numeric value or the actual value mapped string constant.

5.2.1.3.7.2 Rules for Specifying Array Properties

Some properties on Managed Elements are arrays. The CLP provides two methods for dealing with array properties. Implementations shall support both methods. The first method allows individual positions within an array property to be addressed by index using bracket notation. Bracket notation consists of a property name followed by an opening bracket ('['), followed by one or more characters specifying the desired index, followed by a closing bracket (']'). Note that no white space occurs anywhere between the property name and the closing bracket. When a property target term includes a '[' character, followed by a ']' character, the implementation shall interpret the characters that occur between the two brackets as specifying the index of the position within the array property that is being addressed. For each array property, legal values for the index are defined by the MOF (*4HCommon Information Model (CIM) Schema, version 2.12*) that defines the class to which the property belongs. The syntax for addressing a position within an array property is as follows:

```
<property name>"["<index>"]"
```

This syntax is supported wherever a property name/value is accepted by the CLP.

The alternate approach is supported only for the assignment of values to an array property. It is documented in the following clause. If a client uses array notation with a property that is not an array property, the implementation will return an error.

5.2.1.3.7.3 Using the Assignment Operator

The assignment operator is used to indicate a desired value to be assigned to a property. The syntax for using the assignment operator in a target property term is as follows:

```
<property name>=<property value>
```

When the property name contains the bracket notation defined in "Rules for Specifying Array Properties" (5.2.1.3.7.2), the implementation shall assign the property value to the array position identified by the index delimited by the brackets.

If the property is multi-valued (an array), multiple array positions can be assigned using a comma-delimited list of values. When the target property value of a target property term is a comma-delimited list, the implementations shall interpret each comma-delimited token in the target property value as the value to be assigned to the corresponding array position of the property. When the comma-delimited token is a zero-length string, the implementation shall not assign a value to the corresponding array position. When `<property name>` identifies an array property, and neither of the two methods for managing array properties is used, the implementation shall attempt to assign `<property value>` to the first position in the array.

5.2.1.3.7.4 Using the Equivalence Operator

The equivalence operator is used to indicate that an implementation filters results for instances that have a property with the specified name and value. The syntax for using the equivalence operator is as follows:

```
<property name>==<property value>
```

When the property name contains the bracket notation defined in "Rules for Specifying Array Properties" (5.2.1.3.7.2), the implementation shall compare the property value to the value of the array position identified by the index delimited by the brackets. When `<property name>` identifies an array property, and the bracket notation defined in "Rules for Specifying Array Properties" is not used, the implementation shall compare `<property value>` to all array positions in the property.

5.2.2 Output Data Schema

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

5.2.2.1 Command Response Organization

In the absence of communication errors or session termination, every CLP command will result in 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 (for example, successful or errors/exceptions)
- Command Results data (for example, information generated from/by the command)

See 5.1.10 for full descriptions of the data elements. CLP command options described in Clause 7 control the content and format of output data.

5.2.2.2 Common Output Keywords

Common output keywords (Table 2) are those data elements that may appear in any response data. Common keywords are used for data organization purposes—identification, grouping, sorting, and so on.

Table 2 – Common Output Keywords

Keyword	Definition
`association`	Indicates that the data is a target association.
`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.
`property, properties`	Indicates that the data is a target property name.
`target`	Indicates that the data is a Managed Element that may contain other Managed Elements.
`targets`	Indicates that the data lists targets contained in an element.
`ufct`	Indicates that the data is a User-Friendly class Tag.
`ufit`	Indicates that the data is a User-Friendly instance Tag.
`ufip`	Indicates that the data is a User-Friendly instance Path (fully qualified path).
`verb, verbs`	Indicates that the data is a command verb name.
`endoutput`	Indicates the end of a "keyword" Command Response.

5.2.3 Command Status Data Elements

The following clauses describe the constraints on each of the Command Status data elements.

5.2.3.1 Command Status Keywords

A Command Response includes Command Status data elements.

Table 3 lists the keywords used to identify properties of the Command Status data elements.

1347 **Table 3 – Command Status Keywords**

Keyword	Definition
status	The Status property. This is one of the values defined in Table 4.
status_tag	The status_tag property. This is one of the values defined in Table 4.
error	The Processing Error property detected by the CLP Service. This is one of the integer values in Table 6.
error_tag	The error_tag property. This is the string value in Table 6.

1348 The Command Status indicates the processing disposition of the command entered. When the `status`
1349 keyword is returned, implementations shall assign it one of the values listed in Table 4. When the
1350 `status_tag` keyword is returned, implementations shall assign it the value in Table 4 that corresponds
1351 to the value of the `status` keyword.

1352 **Table 4 – 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 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.

1353 Table 5 lists the keywords used to identify properties of the Message data element. Each `message_arg`
1354 identifies a string value for insertion into the message text. If an implementation supports message
1355 argument insertion into message text, the implementation shall identify each insertion location in the
1356 message text using the character sequence `{n}`. Implementations shall interpret the value of *n* as
1357 identifying the index of the message argument to insert.

1358 **Table 5 – 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 6 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 shall assign them values from Table 6.

Table 6 – Processing Error Values and Tags

error	error_tag	Description
255	COMMAND ERROR – UNSPECIFIED	Unspecified command error; used only when other command errors are not applicable.
254	COMMAND NOT SUPPORTED	The command is recognized as a CLP command verb but is not supported by this implementation.
253	COMMAND NOT RECOGNIZED	The command is syntactically correct, but the implementation does not recognize the first term in the command as a verb (that is, cannot report "not supported" because the verb is unknown to the implementation).
252	COMMAND SYNTAX ERROR	The command is recognized as a CLP command verb, but the syntax has not been correctly followed.
251	INVALID OPTION	The command is recognized as a CLP command verb, the syntax is correct, but an option is not valid.
250	INVALID ARGUMENT	The 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	The user selected an output format that is not supported by this implementation.
248	MISSING ARGUMENT	The command is recognized as a CLP command verb, the syntax is correct, but an argument value for an option is missing.
247	OPTION NOT SUPPORTED	The 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 cannot be started to execute the command.
243	UNRECOGNIZED OEM EXTENSION	The Command Line includes an OEM Extension Name String that is unrecognized by the implementation.
242	MISSING REQUIRED TARGET	The command verb requires that a command target term be specified to identify a specific target for the command, and a command target term was not included in the Command Line.
241	FUNCTION NOT SUPPORTED	The command syntax is valid but included a request for optional behavior that is not supported by this implementation.

When an error occurs processing a command prior to creating a job to execute the command and this specification does not identify a specific Processing Error to use to indicate the error condition, the implementation shall return a Command Status of COMMAND PROCESSING FAILED and a Processing Error of COMMAND ERROR – UNSPECIFIED.

5.2.3.2 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. Table 7 defines the valid keywords and values for the Job data element. The accepted values and corresponding descriptions for each value are provided in the tables that follow. For each keyword, implementations shall return data that conforms to the restrictions specified for the keyword in Table 7.

1373 **Table 7 – Job Error Keywords**

Keyword	Definition
job_id	Job Identifier—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 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.

1374 The Execution Error property communicates the primary category of the Job Error. If an implementation
1375 includes the errtype keyword in a Command Response, the implementation shall assign the keyword
1376 one of the values from Table 8. If an implementation includes the errtype_desc keyword in a
1377 Command Response, the implementation shall assign the keyword the value from Table 8 that
1378 corresponds to the value assigned to the errtype keyword. The values for errtype and
1379 errtype_desc correspond to the ValueMap and Values for the ErrorType property of CIM_Error.

1380 **Table 8 – Error Type Values and Descriptions**

errtype	errtype_desc	Description
0	Unknown	None
1	Other	None
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	None
4	Software Error	None
5	Hardware Error	None
6	Environmental Error	None
7	Security Error	None
8	Oversubscription Error	None
9	Unavailable Resource Error	CLP Service cannot acquire needed internal resources to process the command.
10	Unsupported Operation Error	None

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 9 being returned. If an implementation includes the `cimstat` keyword in a Command Response, the implementation shall assign the keyword one of the values from Table 9. If an implementation includes the `cimstat_desc` keyword in a Command Response, the implementation shall assign the `cimstat_desc` keyword the value from Table 9 that 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 9 – CIM Status Code Values and Descriptions

cimstat	cimstat_desc	Description
1	CIM_ERR_FAILED	A general, unspecified error occurred.
2	CIM_ERR_ACCESS_DENIED	The user does not have proper authorization to use the command. or The command was authorized for the user, but the user is not authorized to perform a resulting operation on this or a dependent target ME. or The user does not have access to a CIM resource.
3	CIM_ERR_INVALID_NAMESPACE	The target namespace does not exist.
4	CIM_ERR_INVALID_PARAMETER	The verb is recognized, the command syntax is correct, option names are correct, but an option argument value is not valid. One or more target property values or option argument values that specify target properties are 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	The command target is not found. The requested UFiT could not be found or was unresponsive.
7	CIM_ERR_NOT_SUPPORTED	The command is valid but the target specified does not support the necessary operation or operations needed to carry out the command. OR The user has requested an operation that is not supported by this target ME.
8	CIM_ERR_CLASS_HAS_CHILDREN	The operation cannot be carried out on this class because it has subclasses with instances.
9	CIM_ERR_CLASS_HAS_INSTANCES	The operation cannot be carried out on this class because it has instances.
10	CIM_ERR_INVALID_SUPERCLASS	The operation cannot be carried out because the superclass does not exist.
11	CIM_ERR_ALREADY_EXISTS	The operation cannot be carried out because the specified UFiT already exists.

cimstat	cimstat_desc	Description
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 currently executing on the Target is performing an internal function such that the requested operation cannot be concurrently executed. or The target ME is in use by another session.
17	CIM_ERR_METHOD_NOT_FOUND	The 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 target was unexpected. or The job spawned by the 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 shall assign the keyword one of the values from Table 10. If an implementation includes the `severity_desc` keyword in a Command Response, the implementation shall assign the `severity_desc` keyword the value from Table 10 that 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 10 – 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, and 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 11. If an implementation includes the `probcause` keyword in a Command Response, the implementation shall assign the keyword one of the values from Table 11. If an implementation includes the `probcause_desc` keyword in a Command Response, the implementation shall assign the `probcause_desc` keyword the value from Table 11 that 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 11 – Probable Cause Values and Descriptions

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

probcause	probcause_desc
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
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

probcause	probcause_desc
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
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

probcause	probcause_desc
120	Delayed Information
121	Duplicate Information
122	Information Missing
123	Information Modification
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

5.2.4 Output Data Formats

The CLP specifies the following named, selectable formats for output data: "text", "keyword", and "clpxml". These data formats are defined in the following clauses.

5.2.4.1 General

Implementations shall support "text" format and shall provide "text" format output as the default output setting. When other output data formats are supported, implementations shall allow the user to override the format on a per-command basis using the command option `output`. Implementations shall 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 shall support the "clpxml" output format.

5.2.4.2 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.

5.2.4.3 Structured Outputs

This clause details requirements related to structured output.

5.2.4.3.1 General

The CLP specification defines two structured output formats: "keyword" and "clpxml". When returning a Command Response formatted according to a structured output, the implementation shall use the specific keywords and values identified in 5.2.3.1 and 5.2.3.2 for each data element included in the Command Status data element.

For information about the rules governing the use of the `output` option to select an output format, see **Error! Reference source not found.**.

5.2.4.3.2 Keyword=Value Format

The "keyword" output format requests the command to format the output in a "keyword=value" format. To select "keyword" format explicitly, the implementations shall 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. Implementations shall use double quotes around any value that contains the end-of-line sequence.

Implementations shall specify a group of "keyword" items that are to be interpreted as a single item or collection by using the "begingroup" keyword and a value identifying the type of data. Implementations shall terminate a group by using the "endgroup" keyword. When a "begingroup" keyword appears in the output, all keywords that follow are interpreted as part of a group until the next "endgroup" keyword. Implementations shall indicate the end of the Command Response by using the "endoutput" keyword. Implementations may include blank lines or lines that have an # character (octothorp) in the first character position in the output. If an implementation includes blank lines or lines that have an # character (octothorp) in the first character position in the output, the implementation shall 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 the 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 the 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 shall 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 shall define each keyword using the convention for OEM name extensions defined in 5.2.6.2.

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

5.2.4.3.3 XML Format

The output format "clpxml" requests the command to format the output in an XML document format. To select "clpxml" format explicitly, implementations shall 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 and so on) is defined per command. An outline of the XML document specific to each CLP verb is located in the "XML Output" subclause for that verb in Clause 6.

The XML schema defining the Command Response data element is defined using XSD in *Server Management Command Line Protocol (SM CLP) Command Response XML Schema v1.0* (DSP0224). 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 shall ensure that the document default namespace is:

"http://schemas.dmtf.org/smash/commandresponse/1.0.0/dsp0224.xsd"

OEM vendors may extend the Command Response schema. If OEM vendors extend the Command Response schema, the implementation shall place the OEM extensions into a distinct namespace and shall define a namespace prefix that follows the convention for OEM name extensions defined in 5.2.6.2.

5.2.5 Internationalization/Localization

This clause 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 locale.

5.2.5.1 Command Input

CLP implementations shall not provide support for internationalization of Command Line terms. CLP implementations may provide support for localization of input data. Furthermore, a CLP implementation shall not support alternative strings for CLP command verbs, option names, and target property names except as OEM extensions (see 5.2.6). Commands written using alternative strings for these Command Line terms will not be portable from implementation to implementation.

5.2.5.2 Command Output

This clause details requirements related to localized command output.

5.2.5.2.1 CLP Service-Side Localization

CLP implementations may support localized CLP command output. If the implementation supports localized output, the implementation shall support the `language` session setting and follow the described use of the setting as given in **Error! Reference source not found.**.

5.2.5.2.2 Client-Side Localization

It is possible that a CLP implementation will support localization of CLP command output by the Client. To support localization of output data at the Client, a CLP implementation could support the following capabilities:

- at least one of the structured output modes (see 5.2.4.3)
- capability to report a Message Owner and Message Identifier for each translatable message when a structured output mode is selected (see 5.2.2)

5.2.5.3 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 and time, and so on) through Managed Element settings.

Implementations shall 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 clause.

5.2.6 OEM Extensions

The CLP allows an OEM to add support for vendor-unique commands and output data. This clause details requirements related to OEM extensions to the CLP.

5.2.6.1 General

A vendor may extend the CLP in the following ways:

- by providing OEM commands that conform to one of the specified CLP Extended Forms
- by providing OEM output keywords

5.2.6.2 OEM Extension Name Strings

Implementations shall identify any OEM Extension Name Strings used for CLP Command Line terms with 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 shall 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.

Implementations shall use a value for the OEM string portion of the prefix that uniquely identifies the entity that owns and defines the command. The string shall include a copyrighted, trademarked, or otherwise unique name that is owned by the business entity or standards body defining the command.

Implementations shall interpret and recognize the standard portion of the string, "OEM", and the vendor identifier string as case insensitive.

EXAMPLE ("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".

5.2.6.3 Command Extension Forms

The CLP recognizes two forms of command extension:

- CLP Verb Extended Form
- OEM Command Line Extended Form

5.2.6.3.1 General

If a Command Line contains an OEM Extension Name String that is not supported by the implementation, the implementation shall return a Command Status of COMMAND PROCESSING FAILED and a Processing Error of UNRECOGNIZED OEM EXTENSION.

5.2.6.3.2 CLP Verb Extended Form

The CLP Verb Extended Form requires that a standard CLP command verb appear as the first term on the Command Line.

5.2.6.3.2.1 General

A vendor may define vendor-specific Command Line terms for options, option arguments, option argument values, target addresses, and target properties, as long as those terms follow the semantics defined in the CLP specification.

5.2.6.3.2.2 Terms

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

5.2.6.3.2.3 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>)
```

5.2.6.3.2.4 Rules

The CLP specification defines the behavior of the verb and associated CLP options when the option is specified with a CLP-defined argument.

- OEM Arguments to SM CLP options and property names and values shall observe CLP syntax and delimiter rules.
- The implementation shall not support OEM option arguments or option argument values 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 cannot be used to modify the targets of a command.
- Implementations shall not accept a short form for an OEM Option.
- Implementations of OEM targets/properties shall observe/adhere to the specified CLP verb/option behaviors.

- When defining OEM target name addresses, implementations shall observe the CLP command delimiter characters and may 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, and so on. Therefore, when an OEM-defined option is included in a command, it is likely to be necessary to have a priori knowledge of the option in order to deterministically parse the Command Line.

5.2.6.3.3 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, and so on 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.

5.2.6.3.3.1 Terms

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

5.2.6.3.3.2 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.

5.2.6.3.3.3 Rules

The vendor completely defines the command syntax, behavior, target addressing, and so on that appear after the first term, where the first term is prefixed by "OEM".

5.2.6.4 Output Extensions

This clause details requirements related to vendor extensions to the CLP output.

5.2.6.4.1 Vendor-Specific Keywords

A vendor may supply additional output data elements in the response to any CLP command. An implementation may support vendor-supplied keyword names. The implementation shall define any vendor-supplied keyword names such that they comply with the rules for defining OEM Extension Name Strings defined in 5.2.6.2.

EXAMPLE If vendor "ZYX" introduced an output data element keyword "foobar", the resulting keyword would be "OEMZYXfoobar".

5.2.6.4.2 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 5.2.2.

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.

6 SM CLP Verbs

This clause 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 subclauses 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 Clause 7.

Examples are for information only. If an example contradicts specification text elsewhere in the document, the specification text is the authority. General rules and requirements for the Command Response are specified in 5.1.10. See Annex E for the conventions used in the examples.

6.1 Verb Support Requirements

The requirements in Table 12 are interpreted as follows:

shall If the "Requirement" column in Table 12 contains "shall", implementations shall support the verb specified in the "Command" column of that row.

PROFILE If the "Requirement" column in Table 12 contains "PROFILE", implementations will support the verb specified in the "Command" column of that row when the implementations support a profile that defines a mapping for the verb. Specific by-target requirements are found in 32HDSP0216.

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 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 needs to know the mapping of CLP verb option/target/property combinations to the CIM Schema. This mapping information is not included in this specification but is collected in a separate specification as noted above.

If a CLP verb is specified as the first term of a Command Line and the implementation does not support any profiles which require that the verb be supported, the implementation shall not execute the command and the implementation shall return a Command Status of COMMAND PROCESSING FAILED and a Processing Error of COMMAND NOT SUPPORTED.

Table 12 – Verb Support Requirements

Command	Requirement	Definition and Usage
`cd`	shall	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 command is allowed only 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 command is allowed only for specific target object types as defined by the profiles and specific MAP implementation.
`dump`	PROFILE	Used to move a binary image from the MAP to a URI.
`exit`	shall	Used to terminate a CLP session.
`help`	shall	Used to get context-sensitive help.
`load`	PROFILE	Used to move a binary image to the MAP from a URI.

Command	Requirement	Definition and Usage
reset	PROFILE	Used to cause a target with power/process control to cycle states from enabled to disabled and back to enabled.
set	shall	Used to set a property or set of properties to a specific value.
show	shall	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	shall	Used to query the version of the CLP implementation (by default) and other CLP elements (when specified).

6.2 cd

The general form of the cd command is:

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

6.2.1 General

For the cd command, implementations shall support the syntax defined for the cd-cmd term in the CLP grammar defined in Annex A.

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. An implementation shall accept a command target term that is either an Absolute Target Address or a Relative Target Address. As a result, the command supports both relative and absolute path changes. If a command target term is specified, implementations of the cd command shall 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. If the command target term does not evaluate to a UFiP according to the rules defined in 5.1.3.5, implementations shall not change the Current Default Target and shall return a Command Status of COMMAND PROCESSING ERROR and a Processing Error of INVALID TARGET.

Implementations of the cd command shall support usage without an argument. If no arguments appear on the Command Line, the command shall not change the Current Default Target, the implementation shall not attempt to validate that the CDT addresses a valid Managed Element, and the implementation shall return the Current Default Target path as output.

Because 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 command target term evaluation rules defined in 5.2.1.3.6 will result in the command target term being resolved to the root of the session's address space. 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.

6.2.2 Valid Targets

Implementations of the cd command will 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.

6.2.3 Options

Implementations of the cd command support the options specified in **Error! Reference source not found.**.

6.2.4 Output

This clause details the requirements for output of the cd verb.

6.2.4.1 Text Format

The Command Results data shall 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 shall return text indicating that the Current Default Target is invalid.

6.2.4.2 Structured Format

This clause details requirements for structured output formats for the cd verb.

6.2.4.2.1 General

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

6.2.4.2.2 XML Output

The implementation shall return the cd element in the `response` element as defined in the Command Response schema in DSP0224.

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

6.2.4.2.3 Keyword

Implementations shall 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
```

6.2.5 Examples

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

EXAMPLE 1: Returns the Current Default Target and does not change it. No validation of the CDT is performed by the implementation.

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

EXAMPLE 2: Returns the Current Default Target and does not change it. The CDT is validated and appropriate Command Response data is returned if it is no longer valid.

```
-> cd .
/system1/cpu2
```

EXAMPLE 3: Moves to the parent (container) of the Current Default Target (up the tree) one level.

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

EXAMPLE 4: Moves up the containment tree two levels.

```
-> cd ../..
/
```

EXAMPLE 5: Changes the Current Default Target to the (absolute) target `/system1/cpu1`.

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

EXAMPLE 6: Changes the Current Default Target to `/system1/cpu2/temp1` (assuming a target named with a UFiT of `temp1` exists within `/system1/cpu2`).

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

EXAMPLE 7: Moves to the session's Current Default Target's parent and then to `cpu1`.

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

EXAMPLE 8: Returns the error (the target does not resolve) and then returns the current target.

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

EXAMPLE 9: Returns the error (the target does not resolve) and then returns the current target.

```
-> cd –o format=clpxml /cpu1
<?xml version="1.0" encoding="UTF-8"?>
<response
xmlns="http://schemas.dmtf.org/smash/commandresponse/1.0.0/dsp0224.xsd"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xsi:schemaLocation="http://schemas.dmtf.org/smash/commandresponse/1.0.0
/dsp0224.xsd 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>
```

EXAMPLE 10: Changes Current Default Target to /system1/cpu3.

```
-> cd –o format=clpxml /system1/cpu3
<?xml version="1.0" encoding="UTF-8"?>
<response
xmlns="http://schemas.dmtf.org/smash/commandresponse/1.0.0/dsp0224.xsd"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xsi:schemaLocation="http://schemas.dmtf.org/smash/commandresponse/1.0.0
/dsp0224.xsd 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>
```

EXAMPLE 11: Changes Current Default Target 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
```

EXAMPLE 12: Attempts to change Current Default Target to system3, and 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
```

EXAMPLE 13: Bad syntax on command 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
```

EXAMPLE 14: Changes Current Default Target to SESSION.

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

6.3 create

The general form of the `create` command is:

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

6.3.1 General

For the `create` command, implementations shall support the syntax defined for the `create-cmd` term in the CLP grammar defined in Annex A.

The `create` command is used to create new target objects in the target address space of the implementation. The `create` command is supported only in certain specific target profiles. This command will be supported on any implementation capable of creating new target objects. An example is 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 final term of the Resultant Address is a specific UFiT, the implementation shall 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 shall 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.

The `create` command does not support usage without Command Line parameters. When a command target term is not included in the Command Line, the implementation shall not execute the command and shall return a Command Status of COMMAND PROCESSING ERROR and a Processing Error of MISSING REQUIRED TARGET. The `create` command can either be used with property name and value pairs or with the `source` option. When property name and value pairs are specified as target property terms, the implementation will attempt to assign each named property the associated value. If there are additional properties on the instance, the implementation will provide default values. If the implementation cannot assign default values to unspecified properties, it will return an error. The required properties, default values, and so on are documented in the CLP-to-CIM mapping for the profile that defines the target. When the `create` verb is specified without the `source` option, or at least one target property term, the implementation will return a Processing Error of COMMAND SYNTAX ERROR. When the `source` option and at least one target property term are both specified in a command, the implementation will return a Processing Error of COMMAND SYNTAX ERROR. 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 DSP0216.

6.3.2 Valid Targets

This command is supported when it is specified in a target mapping (DSP0216) for a profile that the implementation supports. For all targets that do not support the use of the `create` command, implementations shall 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 will accept an Absolute or a Relative Target Address for the command target term. If the Resultant Address is not a UFiP and does not terminate in a UFsT, implementations shall return a Command Status of COMMAND PROCESSING ERROR and a Processing Error of INVALID TARGET.

6.3.3 Options

Implementations of the `create` command support the options specified in **Error! Reference source not found.**. The use of the `source` option is as follows:

`-source` The `source` option is used to identify a Managed Element or other data source that will provide initial values for the instance to be created. Examples include a Managed Element that will be cloned or settings to use. When this option is supported for use with a particular target, it is defined in the CLP-to-CIM mapping for that target.

6.3.4 Output

This clause details the requirements for output of the `create` verb.

6.3.4.1 Text Format

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

6.3.4.2 Structured Format

This clause details requirements for structured output formats for the `create` verb.

6.3.4.2.1 General

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

6.3.4.2.2 XML Output

The implementation shall return the `create` element in the `response` element in the returned XML document as defined in the Command Response schema in DSP0224.

```
<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>
```

6.3.4.2.3 Keyword

Implementations shall 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
```

6.3.5 Examples

The following examples try to create a log entry in a log.

EXAMPLE 1: Successfully creates a record log.

```
-> 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
```

EXAMPLE 2: Fails to create a log record due to missing properties.

```
-> create log1/record*
Create failed—-missing required properties.
```

EXAMPLE 3: Fails 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://schemas.dmtf.org/smash/commandresponse/1.0.0/dsp0224.xsd"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xsi:schemaLocation="http://schemas.dmtf.org/smash/commandresponse/1.0.0
/dsp0224.xsd
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>
```

EXAMPLE 4: Fails 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
```

EXAMPLE 5: Successfully creates 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://schemas.dmtf.org/smash/commandresponse/1.0.0/dsp0224.xsd"
xmlns:xsi=http://www.w3.org/2001/XMLSchema-instance"
xsi:schemaLocation="http://schemas.dmtf.org/smash/commandresponse/1.0.0
/dsp0224.xsd
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>
```

EXAMPLE 6: Successfully creates 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
```

6.4 delete

The general form of the `delete` command is:

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

6.4.1 General

For the `delete` command, implementations shall support the syntax defined for the `delete-cmd` term in the CLP grammar defined in Annex A.

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

This command can be used 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 shall 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 shall 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.

6.4.2 Valid Targets

This command is supported when it is specified in a target mapping (DSP0216) for a profile that the implementation supports. For all targets that do not support the use of the `delete` command, implementations shall 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 will accept an Absolute or a Relative Target Address for the command target term. If the Resultant Address is not a UFiP and does not terminate in a UFsT, implementations shall return a Command Status of COMMAND PROCESSING ERROR and a Processing Error of INVALID TARGET.

6.4.3 Options

Implementations of the `delete` command will support the options specified in **Error! Reference source not found.**. The implementation will support other options for the command as specified in the specific target profile.

`-f, -force` The `force` option allows objects to be deleted, ignoring any policy that might cause the implementation to normally not execute the command. When this option is used, the implementation shall execute this deletion if at all possible, without regard to consequences.

6.4.4 Output

This clause details the requirements for output of the `delete` verb.

6.4.4.1 Text Format

Implementations shall return Command Result data that 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 shall indicate that no targets were deleted. For example, the implementation could return the string "No targets deleted" or an appropriate translation.

6.4.4.2 Structured Format

This clause details requirements for structured output formats for the `delete` verb.

6.4.4.2.1 General

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

6.4.4.2.2 XML Output

The implementation shall return the `delete` element in the `response` element as defined in the Command Response schema in DSP0224.

```
<delete>
   <target>
      <instance>
         <ufip>
            UFiP identifying target.
   </ufip>
</instance>
   <target>
          Recursive target elements representing Managed Elements contained in
          the initial target.
   </target>
      .
      .
      .
</target>
   .
   .
   .
</delete>
```

6.4.4.2.3 Keyword

Implementations shall 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
```

6.4.5 Examples

The following examples delete a single log entry on a MAP that supports log deletion. For additional examples, see the target profiles that 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 format is text.

EXAMPLE 1: Deletes the specific log record `record1` contained in `log1`.

```
-> delete log1/record1
log1/record1 deleted.
```

EXAMPLE 2: Deletes all instances of record contained in `log1` (that is, clears the event log).

```
-> delete –o verbose log1/record*
record1 deleted.
record2 deleted.
record3 deleted.
record4 deleted.
record5 deleted.
5 records successfully deleted.
```

EXAMPLE 3: Deletes all instances of record contained in `log1` (that is, clears the event log). Displays 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.
```

EXAMPLE 4: Deletes all instances of record contained in `log1`.

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

EXAMPLE 5: Successfully deletes `record3` in `log1` in `system34`. Requests error-only output, so nothing is returned.

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

EXAMPLE 6: Deletes `record3` in `log1` in `system34`.

```
-> delete –o error,format=clpxml /system34/log1/record3
<?xml version="1.0" encoding="UTF-8"?>
<response
xmlns="http://schemas.dmtf.org/smash/commandresponse/1.0.0/dsp0224.xsd"
```

```
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xsi:schemaLocation="http://schemas.dmtf.org/smash/commandresponse/1.0.0
/dsp0224.xsd 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>
```

EXAMPLE 7: Deletes `record3` in `log1` in `system34`.

```
-> 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
```

EXAMPLE 8: Attempts to delete a record that does not exist.

```
-> delete –o format=clpxml log1/record334
<?xml version="1.0" encoding="UTF-8"?>
<response
xmlns="http://schemas.dmtf.org/smash/commandresponse/1.0.0/dsp0224.xsd"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xsi:schemaLocation="http://schemas.dmtf.org/smash/commandresponse/1.0.0
/dsp0224.xsd 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>
```

EXAMPLE 9: Attempts to delete individual records, but this operation is not supported by this implementation.

```
-> delete –o format=clpxml log3/record334
<?xml version="1.0" encoding="UTF-8"?>
<response
xmlns="http://schemas.dmtf.org/smash/commandresponse/1.0.0/dsp0224.xsd"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xsi:schemaLocation="http://schemas.dmtf.org/smash/commandresponse/1.0.0
/dsp0224.xsd 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>
```

EXAMPLE 10: Attempts to delete individual records, but this operation 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
```

EXAMPLE 11: Deletes all of the users in the current target.

```
-> delete user*
user[1-20] successfully deleted.
```

EXAMPLE 12: Deletes all records in `log1` (only four exist).

```
-> delete –o format=clpxml log1/record*
<?xml version="1.0" encoding="UTF-8"?>
<response
xmlns="http://schemas.dmtf.org/smash/commandresponse/1.0.0/dsp0224.xsd"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xsi:schemaLocation="http://schemas.dmtf.org/smash/commandresponse/1.0.0
/dsp0224.xsd 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>
```

EXAMPLE 13: Deletes 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
```

6.5 dump

The general form of the dump command is:

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

6.5.1 General

For the dump command, implementations shall support the syntax defined for the dump-cmd term in the CLP grammar defined in Annex A.

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 supported only on certain specific target profiles. This command shall be supported on any implementation that 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 shall not execute the command and shall return a Command Status of COMMAND PROCESSING FAILED and a Processing Error of REQUIRED OPTION MISSING.

6.5.2 Valid Targets

This command is supported when it is specified in a target mapping (DSP0216) for a profile that the implementation supports. For all targets that do not support the use of the dump command, implementations shall not show the dump command in a command listing as being available. Implementations of the dump command will accept an Absolute or a Relative Target Address for the command target term. If the Resultant Address is not a UFiP, implementations shall return a Command Status of COMMAND PROCESSING ERROR and a Processing Error of INVALID TARGET.

6.5.3 Options

Following are valid options for the dump command in addition to those specified in **Error! Reference source not found.**:

```
-destination <URI>
```

The destination option tells the implementation the target to which it will transfer the binary image. The destination option is required on the Command Line every time this verb is executed.

The URI specified can contain a scheme that indicates the explicit service and location to be used to capture the dumped data.

6.5.4 Output

This clause details the requirements for output of the dump verb.

6.5.4.1 Text Format

Implementations shall include the source and target addresses in the Command Results data and shall indicate whether the operation was successful.

EXAMPLE 1: If the command was successful, an implementation could return the following string:

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

EXAMPLE 2: If the file is not transferred, the implementation could return the following string:

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

6.5.4.2 Structured Format

This clause details requirements for structured output formats for the `dump` verb.

6.5.4.2.1 General

The returned data shall include any status data in the standard format at the top of the response. The `dump` command shall 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 shall identify the destination using the keyword `ufip`. If the destination is a URI, the implementation shall identify the destination using the keyword `uri`. The Client will need to check the Command Status to determine whether the transfer was successful.

6.5.4.2.2 XML Output

The implementation shall return the `dump` element in the `response` element as defined in the Command Response schema in DSP0224. A portion of the schema is illustrated below. Note that an implementation will return either the <uri> or <ufip> element as appropriate for the format of the source and destination of the command.

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

6.5.4.2.3 Keyword

Implementations shall 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 shall be used instead of the `uri` keyword.

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

6.5.5 Examples

This clause details examples of the use of the `dump` verb.

EXAMPLE 1: Transfers 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.
```

EXAMPLE 2: Attempts to transfer a binary image of `memory1` to an FTP site but the transfer fails because the userid/password for the destination is invalid.

```
-> dump –destination `
   ftp://administrator:passw0rd@myserver.com/private/administrator/
   memory.dmp –o format=clpxml memory1
<?xml version="1.0" encoding="UTF-8"?>
<response
xmlns="http://schemas.dmtf.org/smash/commandresponse/1.0.0/dsp0224.xsd"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xsi:schemaLocation="http://schemas.dmtf.org/smash/commandresponse/
1.0.0/dsp0224.xsd
smclp_command_response.xsd">
   <command>
      <inputline>dump -destination
   ftp://administrator:passw0rd@myserver.com/private/
      administrator/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>
```

EXAMPLE 3: Transfer fails because the userid/password for the destination is invalid.

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

EXAMPLE 4: Requests help for the dump command.

```
-> dump –help –o format=clpxml
<?xml version="1.0" encoding="UTF-8"?>
<response
xmlns="http://schemas.dmtf.org/smash/commandresponse/1.0.0/dsp0224.xsd"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xsi:schemaLocation="http://schemas.dmtf.org/smash/commandresponse/
1.0.0/dsp0224.xsd
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 specified 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>
```

EXAMPLE 5: 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 specified 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
```

EXAMPLE 6: Transfers memory1.

```
-> dump –destination `
ftp://administrator:passw0rd@myserver.com/private/administrator/memory.
dmp –o format=clpxml memory1
<?xml version="1.0" encoding="UTF-8"?>
<response
xmlns="http://schemas.dmtf.org/smash/commandresponse/1.0.0/dsp0224.xsd"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xsi:schemaLocation="http://schemas.dmtf.org/smash/commandresponse/1.0.0
/dsp0224.xsd smclp_command_response.xsd">
   <command>
```

```
      <inputline>dump –destination
      ftp://administrator:passw0rd@myserver.com/private/
      administrator/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://administrator:passw0rd@myserver.com/private/
         administrator/memory.dmp</uri>
      </destination>
   </dump>
</response>
```

EXAMPLE 7: Command is missing the required destination option.

```
-> dump –o format=clpxml memory1
<?xml version="1.0" encoding="UTF-8"?>
<response
xmlns="http://schemas.dmtf.org/smash/commandresponse/1.0.0/dsp0224.xsd"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xsi:schemaLocation="http://schemas.dmtf.org/smash/commandresponse/1.0.0
/dsp0224.xsd 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>
```

EXAMPLE 8: Spawns a job to transfer `memory1`.

```
-> dump –destination `
ftp://administrator:passw0rd@myserver.com/private/administrator/memory.
dmp –o format=clpxml memory1
<?xml version="1.0" encoding="UTF-8"?>
<response
xmlns="http://schemas.dmtf.org/smash/commandresponse/1.0.0/dsp0224.xsd"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xsi:schemaLocation="http://schemas.dmtf.org/smash/commandresponse/1.0.0
/dsp0224.xsd smclp_command_response.xsd">
   <command>
      <inputline>dump -destination

   ftp://administrator:passw0rd@myserver.com/private/administrator/
         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://administrator:passw0rd@myserver.com/private/
         administrator/memory.dmp</uri>
      </destination>
   </dump>
</response>
```

EXAMPLE 9: Spawns a job to transfer `memory1`.

```
-> dump –destination `
ftp://administrator:passw0rd@myserver.com/private/administrator/memory.
dmp –o format=keyword memory1
commandline=dump -destination
ftp://administrator:passw0rd@myserver.com/private/administrator/memory.
dmp memory1
status=1
job_id=385
command=dump
begingroup=source
ufip=/system1/memory1
endgroup
begingroup=destination
uri=ftp://administrator:passw0rd@myserver.com/private/administrator/mem
ory.dmp
endgroup
endoutput
```

6.6 exit

The general form of the `exit` command is:

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

6.6.1 General

For the `exit` command, implementations shall support the syntax defined for the `exit-cmd` term in the CLP grammar defined in Annex A.

The `exit` command terminates the user's current CLP session. This command shall be supported. When this command is received, implementations shall initiate a graceful shutdown of the underlying transport. Prior to initiating the shutdown, implementations shall return Command Response data indicating that the shutdown has been initiated and should wait for the message to be received by the Client prior to ending the session.

6.6.2 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 a command target term, the implementation shall not execute the command and shall return a Command Status of COMMAND PROCESSING FAILED and a Processing Error of COMMAND SYNTAX ERROR in the Command Response data.

6.6.3 Options

Implementations of the `exit` command will support the options specified in **Error! Reference source not found.**.

6.6.4 Output

This clause describes requirements for CLP output for the `exit` verb.

6.6.4.1 Text Format

The Command Response data shall include Command Status.

6.6.4.2 Structured Format

This clause details requirements for structured output formats for the `exit` verb.

6.6.4.2.1 General

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

6.6.4.2.2 XML Output

The implementation shall return the `exit` element in the `response` element as defined in the Command Response schema in DSP0224.

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

6.6.4.2.3 Keyword

Implementations shall use the following form when returning Command Results for the `exit` command in "keyword" format:

```
command=exit
endoutput
```

6.6.5 Examples

This clause provides examples of the use of the `exit` verb.

EXAMPLE 1: Examines the effect of the exit command.

```
-> exit –x

   If run without the examine option, this command will exit the
current CLP session.
```

EXAMPLE 2: Displays help for the `exit` command.

```
-> exit -help

   The exit command is used to exit a CLP session.
```

EXAMPLE 3: Exits the current session.

```
-> exit –output format=clpxml
<?xml version="1.0" encoding="UTF-8"?>
<response
xmlns="http://schemas.dmtf.org/smash/commandresponse/1.0.0/dsp0224.xsd"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xsi:schemaLocation="http://schemas.dmtf.org/smash/commandresponse/1.0.0
/dsp0224.xsd 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>
```

6.7 help

The general form of the `help` command is:

```
help [<options>] [<help topics>]
```

6.7.1 General

For the `help` command, implementations shall support the syntax defined for the `help-cmd` term in the CLP grammar defined in Annex A.

The `help` command is used to request information related to the use of the CLP. The text that is returned by this command can be defined by an OEM as required for its specific market. The `help` command accepts zero or more options. The `help` command can be specified with zero or more tokens identifying topics for which the user is requesting help. Examples of possible tokens that an implementation could recognize include verb names, option names, target addresses, UFcTs, and target property names. The topics recognized and supported by an implementation are implementation specific. If the implementation recognizes a token as identifying a topic for which it can provide specific help text, the implementation may return help text specific to the topic identified by the token. Implementations of the `help` command shall implement the rules for recognizing and using option terms as specified in 5.2.1.3.3. The `help` command is an exception to the general rules regarding recognizing command target terms and target property terms.

6.7.2 Valid Targets

The `help` command is unique in that it does not operate against a target. An implementation may recognize a token as a target address and provide help specific to that target.

6.7.3 Options

Implementations of the `help` command will support the following options, in addition to those specified in **Error! Reference source not found.**. These option arguments 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.

6.7.4 Output

This clause states requirements for CLP output for the `help` verb..

6.7.4.1 Text Format

The implementation should return text providing help to the user.

6.7.4.2 Structured Format

This clause details requirements for structured output formats for the `help` verb.

6.7.4.2.1 General

The returned data shall include any status data in the standard format at the top of the response. The `help` command shall then return an OEM-defined set of text describing the help for the target or command specified on the Command Line. The keyword "helptext" shall be used when returning this text.

6.7.4.2.2 XML Output

The implementation shall return the `help` element in the `response` element as defined in the Command Response schema in DSP0224.

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

6.7.4.2.3 Keyword

Implementations shall use the following form when returning Command Results for the `help` command in "keyword" format:

```
command=help
help=The help text.
Endoutput
```

6.7.5 Examples

This clause provides examples of the use of the `help` verb.

EXAMPLE 1: Displays help for the log target.

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

EXAMPLE 2: Uses the `examine` option with the `help` command.

```
-> help –x –o format=clpxml /system1
<?xml version="1.0" encoding="UTF-8"?>
<response
xmlns="http://schemas.dmtf.org/smash/commandresponse/1.0.0/dsp0224.xsd"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xsi:schemaLocation="http://schemas.dmtf.org/smash/commandresponse/1.0.0
/dsp0224.xsd 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>
```

EXAMPLE 3: Uses the `examine` option 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
```

6.8 load

The general form of the `load` command is:

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

6.8.1 General

For the `load` command, implementations shall support the syntax defined for the `load-cmd` term in the CLP grammar defined in Annex A.

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 dictates 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 is implementation dependent whether additional actions are required to complete the installation process. This command is supported only on certain specific target profiles. The `load` command will be supported on implementations that manage 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 shall not execute the command and shall return a Command Status of COMMAND PROCESSING FAILED and a Processing Error of REQUIRED OPTION MISSING.

6.8.2 Valid Targets

This command is supported when it is specified in a target mapping (DSP0216) for a profile that the implementation supports. For all targets that do not support the use of the `load` command, implementations shall not show the `load` in a command listing as being available. Implementations of the `load` command will accept an Absolute or a Relative Target Address for the command target term. If the Resultant Address is not a UFiP, implementations shall return a Command Status of COMMAND PROCESSING ERROR and a Processing Error of INVALID TARGET.

6.8.3 Options

Implementations of the `load` command support the following option, in addition to those specified in **Error! Reference source not found.**:

`-source <URI>` This option tells the implementation the target from which it will transfer the binary image.

The URI specified can contain a scheme that indicates the explicit service and location to be used to retrieve the binary image.

6.8.4 Output

This clause states the requirements for CLP output for the `load` verb.

6.8.4.1 Text Format

The Command Result data shall include the source URI and the target instance address, and shall indicate whether the command was successful.

EXAMPLE 1: If the command was successful, an implementation could return the following string:

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

EXAMPLE 2: If the image is not transferred successfully, the implementation could return the following string:

```
<URI> not transferred
```

6.8.4.2 Structured Format

This clause details requirements for structured output formats for the `load` verb.

6.8.4.2.1 General

The returned data shall include any status data in the standard format at the top of the response. The `load` command shall 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 return only the URI address (with the `source` keyword). If the source is an address within the MAP address space, the implementation shall identify the source using the keyword `ufip`. If the destination is a URI, the implementation shall identify the source using the keyword `uri`.

6.8.4.2.2 XML Output

The implementation shall return the `load` element in the `response` element as defined in the Command Response schema in DSP0224. A portion of the schema is illustrated below. Note that an implementation will either return the <uri> or <ufip> element as appropriate for the format of the source and destination of the command.

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

6.8.4.2.3 Keyword

Implementations shall 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
```

6.8.5 Examples

This clause provides examples of the use of the `load` verb.

EXAMPLE 1: Loads firmware image from FTP site.

```
-> load –source ftp://myserver.com/pub/fwimage.img ` firmwareimage
   ftp://myserver.com/pub/firmwareimage.img is transferred to
firmwareimage.
```

EXAMPLE 2: Loads firmware image from an authenticated FTP site.

```
-> load –source ` ftp://administrator:passw0rd@myserver.com/private/ `
administrator/firmware.img –o format=clpxml softwareid1
<?xml version="1.0" encoding="UTF-8"?>
<response
xmlns="http://schemas.dmtf.org/smash/commandresponse/1.0.0/dsp0224.xsd"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xsi:schemaLocation="http://schemas.dmtf.org/smash/commandresponse/1.0.0
/dsp0224.xsd smclp_command_response.xsd">
   <command>
      <inputline>load -source
   ftp://administrator:passw0rd@myserver.com/private/
      administrator/firmware.img -o format=clpxml
         softwareid1</inputline>
   </command>
   <cmdstat>
      <status>0</status>
      <job>
         <job_id>385</job_id>
      </job>
   </cmdstat>
   <load>
      <source>
      <uri>ftp://administrator:passw0rd@myserver.com/private/
      administrator/firmware.img</uri>
      </source>
      <destination>
         <ufip>/system1/softwareid1</ufip>
      </destination>
   </load>
</response>
```

EXAMPLE 3: Loads firmware image from an authenticated FTP site.

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

```
commandline=load -source
ftp://administrator:passw0rd@myserver.com/private/administrator/
firmware.img -o format=keyword softwareid1
status=0
job_id=385
command=load
begingroup=source
uri=ftp://administrator:passw0rd@myserver.com/private/administrator/
firmware.img
endgroup
begingroup=destination
ufip=/system1/softwareid1
endgroup
endoutput
```

EXAMPLE 4: Fails to load firmware image from an authenticated FTP site due to bad credentials.

```
-> load –source ` ftp://administrator:passw0rd@myserver.com/private/`
administrator/firmware.img –o format=clpxml softwareid1
<?xml version="1.0" encoding="UTF-8"?>
<response
xmlns="http://schemas.dmtf.org/smash/commandresponse/1.0.0/dsp0224.xsd"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xsi:schemaLocation="http://schemas.dmtf.org/smash/commandresponse/1.0.0
/dsp0224.xsd
smclp_command_response.xsd">
   <command>
     <inputline>load -source
   ftp://administrator:passw0rd@myserver.com/private/
     administrator/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>
```

6.9 reset

The general form of the `reset` command is:

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

6.9.1 General

For the `reset` command, implementations shall support the syntax defined for the `reset-cmd` term in the CLP grammar defined in Annex A.

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 can be used with and without Command Line options.

6.9.2 Valid Targets

This command is supported when it is specified in a target mapping (DSP0216) for a profile that the implementation supports. For all targets that do not support the use of the `reset` command, implementations shall not show the `reset` command in a command listing as being available. Implementations of the `reset` command will accept an Absolute or a Relative Target Address for the command target term. If the Resultant Address is not a UFiP, implementations shall return a Command Status of COMMAND PROCESSING ERROR and a Processing Error of INVALID TARGET.

The behavior of state-change commands for each UFcT is defined in DSP0216.

6.9.3 Options

Following are valid options for the `reset` command in addition to those specified in **Error! Reference source not found.**:

`-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 shall execute this reset if at all possible, without regard to consequences.

6.9.4 Output

This clause states requirements for output for the `reset` verb.

6.9.4.1 Text Format

Implementations shall return Command Result data that 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 shall indicate this result.

6.9.4.2 Structured Format

This clause details requirements for structured output formats for the `reset` verb.

6.9.4.2.1 General

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

6.9.4.2.2 XML Output

The implementation shall return the `reset` element in the `response` element as defined in the Command Response schema in DSP0224.

```
<reset>
   <ufip> Target address the command was invoked against </ufip>
```

```
   <timestamp> Time reset occurred if completed synchronously, returned in CIM
   datetime format </timestamp>
</reset>
```

6.9.4.2.3 Keyword

Implementations shall 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
```

6.9.5 Examples

This clause provides examples of the use of the `reset` verb.

EXAMPLE 1: Resets the operating system on `system1`.

```
-> reset /system1

   /system1 reset at 10:40am 1/1/01.
```

EXAMPLE 2: Fails to reset the operating system on `system1`.

```
-> reset –o format=clpxml /system1/os1
<?xml version="1.0" encoding="UTF-8"?>
<response
xmlns="http://schemas.dmtf.org/smash/commandresponse/1.0.0/dsp0224.xsd"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xsi:schemaLocation="http://schemas.dmtf.org/smash/commandresponse/1.0.0
/dsp0224.xsd 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>
```

EXAMPLE 3: Fails to reset the operating system on `system1`.

```
-> 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
```

EXAMPLE 4: Resets the operating system on `system1`.

```
-> reset –w –o format=clpxml /system1/os1
<?xml version="1.0" encoding="UTF-8"?>
<response
xmlns="http://schemas.dmtf.org/smash/commandresponse/1.0.0/dsp0224.xsd"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xsi:schemaLocation="http://schemas.dmtf.org/smash/commandresponse/1.0.0
/dsp0224.xsd 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>
```

EXAMPLE 5: Resets the operating system on `system1`.

```
-> 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
```

6.10 set

The general form of the `set` command is:

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

6.10.1 General

For the `set` command, implementations shall support the syntax defined for the `set-cmd` term in the CLP grammar defined in Annex A.

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

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 shall 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. It is necessary for the user to check the command output or the target itself for the value of each property to determine which values were set. Applying properties one command at a time is a deterministic way to determine which properties get applied for any given command.

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.

6.10.2 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 use of the `set` command, the `set` command shall not show up in a command listing as being available. Implementations of the `set` command will accept an Absolute or a Relative Target Address for the command target term. If the Resultant Address is not a UFiP, implementations shall return a Command Status of COMMAND PROCESSING ERROR and a Processing Error of INVALID TARGET.

6.10.3 Options

Implementations of the `set` command will support the options specified in **Error! Reference source not found.**.

6.10.4 Output

This clause states requirements for output for the `set` verb.

6.10.4.1 Text Format

Implementations shall return Command Result data that 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.

6.10.4.2 Structured Format

This clause details requirements for structured output formats for the `set` verb.

6.10.4.2.1 General

The returned data shall include any status data in the standard format at the top of the response. The `set` command shall 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.

6.10.4.2.2 XML Output

The implementation shall return the `set` element in the `response` element as defined in the Command Response schema in DSP0224.

```
<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>
```

6.10.4.2.3 Keyword

Implementations shall use the following form when returning Command Results for the `set` 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
   .
   Additional property groups
   .
endgroup
endoutput
```

6.10.5 Examples

This clause provides examples of the use of the `set` verb.

EXAMPLE 1: Sets the system's name to `sam`.

```
-> set /system1 name=sam
name=sam
```

EXAMPLE 2: Sets password to `12345`.

```
-> set /map1/user3 password=12345
password=12345
```

EXAMPLE 3: Successfully sets a new userid and password.

```
-> set /map1/user3 userid=joesmith password=passw0rd
userid=joesmith
password=passw0rd
```

EXAMPLE 4: The password fails. Whether the userid is updated is implementation-specific behavior. 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=
```

EXAMPLE 5: The password fails. Whether the userid is updated is implementation-specific behavior. 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=
```

EXAMPLE 6: Sets name to a string that contains spaces.

```
-> set /system1 name="Human Resources Server"
name="Human Resources Server"
```

EXAMPLE 7: Fails to enable load balancing on Ethernet port 2 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 enetport2 enabledcapabilities=loadbalancing
<?xml version="1.0" encoding="UTF-8"?>
<response
xmlns="http://schemas.dmtf.org/smash/commandresponse/1.0.0/dsp0224.xsd"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xsi:schemaLocation="http://schemas.dmtf.org/smash/commandresponse/1.0.0
/dsp0224.xsd smclp_command_response.xsd">
   <command>
      <inputline>set -o format=clpxml enetport2
      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">enetport2</ufit>
      <ufip>/system78/enetport2</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>
```

EXAMPLE 8: Fails to enable load balancing on Ethernet port 2 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 enetport2 enabledcapabilities=loadbalancing
commandline=set –o format=keyword enetport2
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/enetport2
begingroup=property
property_name=enabledcapabilities
property_val=3
property_valstring=wakeonlan
property_val=2
property_valstring=alertonlan
endgroup
endgroup
endoutput
```

EXAMPLE 9: Fails to enable load balancing on Ethernet port 2 because teamed NIC is not configured. Notice that this property currently does not have any values assigned.

```
-> set –o format=clpxml enetport2 enabledcapabilities=loadbalancing
<?xml version="1.0" encoding="UTF-8"?>
<response
xmlns="http://schemas.dmtf.org/smash/commandresponse/1.0.0/dsp0224.xsd"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xsi:schemaLocation="http://schemas.dmtf.org/smash/commandresponse/1.0.0
/dsp0224.xsd smclp_command_response.xsd">
   <command>
      <inputline>set -o format=clpxml enetport2
      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">enetport2</ufit>
        <ufip>/system78/enetport2</ufip>
        <properties>
           <property>
              <name>enabledcapabilities</name>
              <multivalue></multivalue>
           </property>
        </properties>
     </instance>
  </set>
</response>
```

EXAMPLE 10: Fails to enable load balancing on Ethernet port 2 because teamed NIC is not configured. Notice that this property currently does not have any values assigned.

```
-> set –o format=keyword enetport2 enabledcapabilities=loadbalancing
commandline=set –o format=keyword enetport2
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/enetport2
begingroup
property_name=enabledcapabilities
endgroup
endoutput
```

EXAMPLE 11: Successfully enables failover and WakeOnLAN support on Ethernet port 2. Notice that multiple values are assigned.

```
-> set –o format=clpxml enetport2
enabledcapabilities=failover,wakeonlan
<?xml version="1.0" encoding="UTF-8"?><response
xmlns="http://schemas.dmtf.org/smash/commandresponse/1.0.0/dsp0224.xsd"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xsi:schemaLocation="http://schemas.dmtf.org/smash/commandresponse/1.0.0
/dsp0224.xsd smclp_command_response.xsd">
   <command>
      <inputline>set -o format=clpxml /system78/enetport2
      enabledcapabilities=failover,wakeonlan</inputline>
         </command>
         <cmdstat>
            <status>0</status>
            <job>
               <job_id>7623</job_id>
            </job>
         </cmdstat>
         <set>
            <instance>
               <ufit instance="2" ufct="port">enetport2</ufit>
               <ufip>/system78/enetport2</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>
```

EXAMPLE 12: Successfully enables failover and WakeOnLAN support on Ethernet port 2. Notice that multiple values are assigned.

```
-> set –o format=keyword enetport2
enabledcapabilities=failover,wakeonlan
commandline=set /system78/enetport2
enabledcapabilities=failover,wakeonlan
status=0
job_id=7632
command=set
begingroup=instance
ufip=/system78/enetport2
begingroup=property
property_name=enabledcapabilities
property_val=4
property_valstring=failover
property_val=3
property_valstring=wakeonlan
endgroup
endgroup
endoutput
```

EXAMPLE 13: Sets the default output format to XML.

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

EXAMPLE 14: Changes the CDT by using the back door method.

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

EXAMPLE 15: Sets the OEM property `OEMxyzpropertyA`.

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

EXAMPLE 16: Sets a property. The vendor returns data outside the specification.

```
-> 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=/system1
begingroup=property
property_name=PrimaryOwnerName
property_val=TheOwner
endgroup
endgroup
oemxyz_message=The Contact information may need to be updated.
endoutput
```

EXAMPLE 17: Enables `software2` for `memory1`. The association is explicitly identified.

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

   software2 is now active for memory1
```

EXAMPLE 18: Fails to change which `software2` is current on `memory1` because multiple instances of ElementSoftwareIdentity reference `memory1`.

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

   Failed to set property "IsCurrent". Both references to an Association
are required when it is the target of a set command.
```

EXAMPLE 19: Enables `software2`. Specifies both references.

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

               software2 is now active for memory1.
```

EXAMPLE 20: Attempts to suspend a currently running job.

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

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

EXAMPLE 21: Stops traffic over `nic1`.

```
-> set /system4/nic1 enabledstate=quiesce

   Traffic over nic1 has been quiesced.
```

EXAMPLE 22: Enables SSHv1 for a session.

```
-> set /system1/sshprotoendpt1 enabledsshversions[1]=sshv1
   /system1/sshprotoendpt1
     enabledsshversions[0] = SSHv2
     enabledsshversions[1] = SSHv1
```

6.11 show

The general form of the `show` command is:

```
show [<options>] [<target>] [<properties>] [<propertyname>== <propertyvalue>]
```

6.11.1 General

For the `show` command, implementations shall support the syntax defined for the `show-cmd` term in the CLP grammar defined in Annex A.

The `show` command is used to display information about Managed Elements or Associations. 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 that 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 that 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 Command Results for the `show` command is determined in the following order:

1) Determine the Resultant Target for the command.
2) Select Managed Element and Associations for which results will be returned from the containment hierarchy below the Resultant Target based on the value of the `level` option.
3) Retrieve results for the selected Managed Elements and Associations.
4) If the command target term terminated in a UFsT, apply the UFcT as a filter against the Managed Elements.
5) If one or more property target terms that include the equivalence operator were specified, filter Managed Elements and Associations based on the property/value.
6) If one or more target property terms that are property names were specified, filter the results to include only Managed Elements and Associations that have the properties, and for each Managed Element and Association, filter the results to include only the specified properties.
7) If the `display` option was specified, apply its arguments to filter the results.

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. If the Resultant Address contains a single term that is a UFsT, the Resultant Target of the command will be the Managed Element that is the root of the address space. The implementation shall return each instance that is contained within the Resultant Target of the type specified by the UFcT specified in the UFsT. The implementation shall 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 shall 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.

The `show` command can be specified with target property terms. Each target property term will either be a property name or contain a property name, equivalence operator, and property value. When specified, the target property terms affect the results returned by the `show` command. The `show` command can be used with target property terms as follows:

- without target property terms
- with target property terms that are property names
- with target property terms that contain the equivalence operator
- with a combination of target property terms that are just property names and others that contain the equivalence operator

When the `show` command is used with one or more target property terms that are property names, the implementation shall restrict the results shown to include only Managed Elements or Associations that have the specified property and restrict the results for the Managed Element or Association to include only properties where the property name was specified as a target property term.

Implementations shall accept at least one target property term that contains the equivalence operator with the `show` command to use as a filter on the Managed Elements or Associations for which results are returned. Implementations may accept more than one target property term that contains an equivalence operator with the `show` command to use as a filter on the Managed Elements or Associations for which results will be returned. When an implementation accepts exactly one occurrence of a target property term that includes the equivalence operator and more than one occurrence of a target property term that includes the equivalence operator is specified on the Command Line, the implementation shall return a Command Status of COMMAND PROCESSING FAILED and a Processing Error of FUNCTION NOT SUPPORTED. For each target property term that includes the equivalence operator specified with the `show` command, the implementation shall restrict the Managed Elements or Associations for which results are shown to include only those Managed Elements or Associations that have a property with the specified name and the value of the property is equivalent to the value specified in the target property term.

The `display` option can be used with the `show` command to control 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 by using the `targets` and `associations` arguments respectively. For more information on using the `display` option, see **Error! Reference source not found.**.

If the `show` command is specified without the `display` option, implementations will use the session default value for the `display` option. Use of the `level` option is supported only when the Resultant Address of the command is a UFiP. Regardless of whether the implementation supports the `level` option, when the `show` command is specified without the `level` option, the implementation shall behave as if the `level` option was specified with an argument of `'1'`. Note that this has no effect when the Resultant Target is an Association Class or Association instance.

The command can be used with and without Command Line options.

6.11.2 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, the behavior can be slightly different as defined in that profile. Implementations of the `show` command will accept an Absolute or a Relative Target Address for the command target term.

6.11.3 Options

Following are valid options for the `show` command in addition to those specified in **Error! Reference source not found.**:

`-l, -level <value>`

Controls the target depth level for the containment hierarchy retrieval. The default for <*value*> is "`1`" (that is, "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.

`-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`. OEMs may also add values using the OEM_ namespace as defined in the OEM extensions clause 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` Instructs the implementation to return all data element types subject to any filtering of categories by the `display` option.

Table 13 lists each type of data element that is returned by the `show` command and indicates whether the type requires the `all` option to 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 be returned unless the `all` option is included with the `show` command. For each data element type listed in Table 13 for which the column labeled "-all Required" includes a value of "yes", implementations shall 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 13 for which the column labeled "-all Required" is blank, implementations shall return elements of the specified data element type irrespective of whether the `all` option is specified.

Table 13 – Data Element Types and `all` Option

Data Element Type	Corresponding Display Argument	`-all` Required
Required properties	properties	
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

An implementation shall include a verb supported by the implementation in the list of verbs for a Managed Element if a command that includes the verb will complete successfully when the Resultant Target of the command is the Managed Element. An implementation shall not include a verb in the list of verbs for a Managed Element if a command that includes the verb will not successfully complete when the Resultant Target of the command is the Managed Element.

6.11.4 Output

This clause states requirements for CLP output for the `show` verb.

6.11.4.1 Text Format

When contained targets are returned, the Command Results output shall include a list of contained targets. If the Command Results contain multiple targets, the implementation shall return results such that the target containment hierarchy is unambiguous. If properties are returned, the implementation shall 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.

6.11.4.2 Structured Format

This clause details requirements for structured output formats for the `show` verb.

6.11.4.2.1 General

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

6.11.4.2.2 XML Output

The XML document fragment indicates the general form of XML-encoded Command Results for the `show` command. It is possible that multiple instances of the <target> element will be returned. Target containment hierarchy is explicitly indicated through <target> element nesting. The implementation shall return the `show` element in the `response` element as defined in the Command Response schema in DSP0224.

```
<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>
```

6.11.4.2.3 Keyword

It is possible that results for multiple Managed Elements will be returned for the `show` command. Each Managed Element is returned as a block starting with:

```
begingroup=instance
```

The target containment hierarchy is not indicated in the Command Results structure. Instead, the `begingroup=instance` keyword/value pair identifies that a listing of UFiPs of contained Managed Elements will follow if this target contains other targets.

Implementations shall use the following form when returning Command Results for the `show` 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
```

6.11.5 Examples

This clause provides examples of the use of the `show` verb.

EXAMPLE 1: Shows all of the targets in the root of the address space.

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

EXAMPLE 2: Shows the commands that apply to the root of the address space.

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

EXAMPLE 3: Shows the value of the name property on `system1`.

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

EXAMPLE 4: Displays 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://schemas.dmtf.org/smash/commandresponse/1.0.0/dsp0224.xsd"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xsi:schemaLocation="http://schemas.dmtf.org/smash/commandresponse/1.0.0
/dsp0224.xsd 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>
```

EXAMPLE 5: Displays the associations that reference the target.

```
-> show –display associations –o format=clpxml /system1/powersup3
<?xml version="1.0" encoding="UTF-8"?>
<response
xmlns="http://schemas.dmtf.org/smash/commandresponse/1.0.0/dsp0224.xsd"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xsi:schemaLocation="http://schemas.dmtf.org/smash/commandresponse/1.0.0
/dsp0224.xsd smclp_command_response.xsd">
   <command>
      <inputline>show -display associations -o format=clpxml
            /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>
```

EXAMPLE 6: Displays the SuppliesPower and Realizes associations that reference the target.

```
-> show –display associations=(SuppliesPower,Realizes) –o format=clpxml
/system1/powersup3
<?xml version="1.0" encoding="UTF-8"?>
<response
xmlns="http://schemas.dmtf.org/smash/commandresponse/1.0.0/dsp0224.xsd"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xsi:schemaLocation="http://schemas.dmtf.org/smash/commandresponse/1.0.0
/dsp0224.xsd
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>
```

EXAMPLE 7: Views the status of the spawned job above. In this example, the spawned job failed to run to completion.

```
-> show –o format=clpxml –d properties /map1/job1/job385
<?xml version="1.0" encoding="UTF-8"?>
<response
xmlns="http://schemas.dmtf.org/smash/commandresponse/1.0.0/dsp0224.xsd"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xsi:schemaLocation="http://schemas.dmtf.org/smash/commandresponse/1.0.0
/dsp0224.xsd
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://administrator:passw0rd@myserver.com/
                        private/administrator/memory.dmp memory1
                        </val>
                    </value>
                </property>
            </properties>
        </instance>
    </target>
  </show>
</response>
```

EXAMPLE 8: Queries the value of the PrimaryOwnerName and ResetCapability properties by using `-display properties=(ResetCapability,PrimaryOwnerName)` as a filter on these results.

```
-> show -display ` properties=(ResetCapability,PrimaryOwnerName)
/system1
   /system1
     properties
        ResetCapability = Disabled (3)
        PrimaryOwnerName = Some guy
```

EXAMPLE 9: Queries the value of the PrimaryOwnerName and ResetCapability properties using `-display properties=(ResetCapability,PrimaryOwnerName)` 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://schemas.dmtf.org/smash/commandresponse/1.0.0/dsp0224.xsd"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xsi:schemaLocation="http://schemas.dmtf.org/smash/commandresponse/1.0.0
/dsp0224.xsd 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>
```

EXAMPLE 10: 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://schemas.dmtf.org/smash/commandresponse/1.0.0/dsp0224.xsd"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xsi:schemaLocation="http://schemas.dmtf.org/smash/commandresponse/1.0.0
/dsp0224.xsd 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>
```

EXAMPLE 11: Displays information about `system1` and everything immediately contained within it.

```
-> show –d all –level 2 –o format=clpxml /system1
 <?xml version="1.0" encoding="UTF-8"?>
<response
xmlns="http://schemas.dmtf.org/smash/commandresponse/1.0.0/dsp0224.xsd"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xsi:schemaLocation="http://schemas.dmtf.org/smash/commandresponse/1.0.0
/dsp0224.xsd smclp_command_response.xsd">
   <command>
      <inputline>show –d all -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 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="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>
```

EXAMPLE 12: Displays information about `system1` and everything immediately contained within it.

```
-> show –d all –level 2 –o format=keyword /system1
commandline=show –d all -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=009923652930591037628501948339 20
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
```

EXAMPLE 13: Shows 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
```

EXAMPLE 14: Shows all of the CPUs that have a bad operational status.

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

EXAMPLE 15: Shows all of the components that have a bad operational status.

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

EXAMPLE 16: Shows all of the components that are named "main processor 4".

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

EXAMPLE 17: Tries to filter using two property values.

```
-> show -level 2 –display properties=(Name) /system1 'Name=="Main
processor 4",OperationalStatus==Degraded
   Filtering with multiple properties is not supported.
```

EXAMPLE 18: Shows just the operational status of the processors in `system1`.

```
-> show /system1/cpu* OperationalStatus
/system1/cpu1
   OperationalStatus = Degraded
/system1/cpu2
   OperationalStatus = Ok
```

EXAMPLE 19: Assuming the same `system1` as the preceding example, shows the ElementName of all degraded processors in `system1`.

```
-> show /system1/cpu* ElementName OperationalStatus==Degraded
/system1/cpu1
   ElementName = "processor one"
```

EXAMPLE 20: Assuming the same `system1` as the preceding example, shows the OperationalStatus of all degraded processors in `system1`.

```
-> show /system1/cpu* OperationalStatus ' OperationalStatus==Degraded
/system1/cpu1
   OperationalStatus = Degraded
```

EXAMPLE 21: Shows the values for options that can have default values.

```
-> show –display properties –o format=clpxml SESSION
<?xml version="1.0" encoding="UTF-8"?>
<response
xmlns="http://schemas.dmtf.org/smash/commandresponse/1.0.0/dsp0224.xsd"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xsi:schemaLocation="http://schemas.dmtf.org/smash/commandresponse/1.0.0
/dsp0224.xsd 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="clpendpt" instance="5">clpendpt5</ufit>
            <ufip>/map1/sessions1/clpendpt5</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>
```

EXAMPLE 22: Shows information about the processors in `system1`.

```
-> show –display properties=(Name),targets=cpu Name=="Main processor 4"
   /system1/cpu4
      Name is main processor 4
```

6.12 start

The general form of the `start` command is:

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

6.12.1 General

For the `start` command, implementations shall support the syntax defined for the `start-cmd` term in the CLP grammar defined in Annex A.

The `start` command starts the target. If the target is already started, an error might or might not be returned. The precise behavior is profile specific.

This command can be used with and without Command Line options.

6.12.2 Valid Targets

This command is supported when it is specified in a target mapping (DSP0216) for a profile that the implementation supports. For all targets that do not support the use of the `start` command,

implementations shall not show the `start` command in a command listing as being available. Implementations of the `start` command will accept an Absolute or a Relative Target Address for the command target term. If the Resultant Address is not a UFiP, implementations shall return a Command Status of COMMAND PROCESSING ERROR and a Processing Error of INVALID TARGET.

The behavior of state-change commands for each UFcT is defined in DSP0216.

6.12.3 Options

Following are valid options for the `start` command in addition to those specified in **Error! Reference source not found.**:

`-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 shall execute this start if at all possible, without regard to consequences.

6.12.4 Output

This clause details requirements for CLP output for the `start` verb.

6.12.4.1 Text Format

Implementations shall return Command Result data that 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 their needs. If the implementation did not start the target, the implementation shall return Command Response data indicating that no target was started.

6.12.4.2 Structured Format

This clause details requirements for structured output formats for the `start` verb.

6.12.4.2.1 General

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

6.12.4.2.2 XML Output

The implementation shall return the `start` element in the `response` element as defined in the Command Response schema in DSP0224.

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

6.12.4.2.3 Keyword

Implementations shall use the following form when returning Command Results for the `start` 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
```

6.12.5 Examples

This clause provides examples of the use of the `start` verb.

EXAMPLE 1: Sends `system1` to its default enabled state.

```
-> start /system1
/system1 started at 10:40am 1/1/01
```

EXAMPLE 2: Fails to start `cpu2` 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://schemas.dmtf.org/smash/commandresponse/1.0.0/dsp0224.xsd"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xsi:schemaLocation="http://schemas.dmtf.org/smash/commandresponse/1.0.0
/dsp0224.xsd 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>
```

EXAMPLE 3: Fails to start `cpu2` 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
```

EXAMPLE 4: Starts `system1` and waits for the job to complete.

```
-> start –o format=clpxml –w /system1
<?xml version="1.0" encoding="UTF-8"?>
<response
xmlns="http://schemas.dmtf.org/smash/commandresponse/1.0.0/dsp0224.xsd"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xsi:schemaLocation="http://schemas.dmtf.org/smash/commandresponse/1.0.0
/dsp0224.xsd 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>
```

EXAMPLE 5: Starts `system1` and waits 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
```

6.13 stop

The general form of the `stop` command is:

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

6.13.1 General

For the `stop` command, implementations shall support the syntax defined for the `stop-cmd` term in the CLP grammar defined in Annex A.

The `stop` command stops the target. If the target is already stopped, an error might or might not be returned. The precise behavior is profile specific.

This command can be used with and without Command Line options.

6.13.2 Valid Targets

This command is supported when it is specified in a target mapping (DSP0216) for a profile that the implementation supports. For all targets that do not support the use of the `stop` command, implementations shall not show the `stop` command in a command listing as being available. Implementations of the `stop` command will accept an Absolute or a Relative Target Address for the command target term. If the Resultant Address is not a UFiP, implementations shall return a Command Status of COMMAND PROCESSING ERROR and a Processing Error of INVALID TARGET.

The behavior of state-change commands for each UFcT is defined in DSP0216.

6.13.3 Options

Following are valid options for the `stop` command in addition to those specified in **Error! Reference source not found.**:

`-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 shall execute this stop if at all possible, without regard to consequences.

6.13.4 Output

This clause details requirements for CLP output for the `stop` verb.

6.13.4.1 Text Format

Implementations shall return Command Result data that includes the target address that was stopped (if any) and the time and date when the stop was initiated. Implementations may return the time/date information in any applicable format that meets their needs. If the implementation did not stop the target, the implementation shall return Command Response data indicating that no targets were stopped.

6.13.4.2 Structured Format

This clause details requirements for structured output formats for the `stop` verb.

6.13.4.2.1 General

Implementations shall include any status data in the standard format at the top of the response. If the target was successfully stopped, the implementation shall then return the target that was stopped and the time the stop completed.

6.13.4.2.2 XML Output

The implementation shall return the `stop` element in the `response` element as defined in the Command Response schema in DSP0224.

```
<stop>
   <ufip> Target address of Managed Element to stop </ufip>
   <timestamp>Time stop occurred if completed synchronously, returned in CIM
         datetime format</timestamp>
</stop>
```

6.13.4.2.3 Keyword

Implementations shall 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
```

6.13.5 Examples

This clause provides examples of the use of the `stop` verb.

EXAMPLE 1: Sends `system1` to its default disabled state.

```
-> stop /system1
   /system1 stopped at 2:59:04 January 30, 2005.
```

EXAMPLE 2: Stops `system1`.

```
-> stop –o format=clpxml /system1
<?xml version="1.0" encoding="UTF-8"?>
<response
xmlns="http://schemas.dmtf.org/smash/commandresponse/1.0.0/dsp0224.xsd"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xsi:schemaLocation="http://schemas.dmtf.org/smash/commandresponse/1.0.0
/dsp0224.xsd 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>
```

EXAMPLE 3: Stops `system1`.

```
-> stop –o format=keyword /system1
commandline=stop /system1
status=0
job_id=9834
command=stop
ufip=/system1
timestamp=20050130145904.000000-300
endoutput
```

EXAMPLE 4: Specifies the `Examine` option and stops `system1`.

```
-> stop -x –o format=clpxml /system1
<?xml version="1.0" encoding="UTF-8"?>
<response
xmlns="http://schemas.dmtf.org/smash/commandresponse/1.0.0/dsp0224.xsd"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
```

```
xsi:schemaLocation="http://schemas.dmtf.org/smash/commandresponse/1.0.0
/dsp0224.xsd 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>
```

EXAMPLE 5: Specifies the `Examine` option and stops `system1`.

```
-> 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
```

EXAMPLE 6: Attempt to stop `system1` does not complete synchronously.

```
-> stop /system1
Stopping system1, job id is 9834.
```

EXAMPLE 7: Attempt to stop `system1` does not complete synchronously.

```
-> stop –o format=clpxml /system1
<?xml version="1.0" encoding="UTF-8"?>
<response
xmlns="http://schemas.dmtf.org/smash/commandresponse/1.0.0/dsp0224.xsd"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xsi:schemaLocation="http://schemas.dmtf.org/smash/commandresponse/1.0.0
/dsp0224.xsd 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>
```

EXAMPLE 8: 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
```

6.14 version

The general form of the `version` command is:

```
version [<options>]
```

6.14.1 General

For the `version` command, implementations shall support the syntax defined for the `version-cmd` term in the CLP grammar defined in Annex A.

The `version` command is used to display the version of the SM CLP that this implementation supports. The implementation shall return the version of the SM CLP with which it is compatible.

6.14.2 Valid Targets

The `version` command does not accept a command target term. Implementations shall not accept non-option terms for the `version` command. If the implementation receives a command with non-option terms specified, the implementation shall return a Command Status of COMMAND PROCESSING ERROR and a Processing Error of COMMAND SYNTAX ERROR.

6.14.3 Options

Valid options for the `version` command are specified in **Error! Reference source not found.**.

6.14.4 Output

This clause details requirements for CLP output for the `version` verb.

6.14.4.1 Text Format

The implementation shall include the string "`Version 1.0.0`" clearly identified as the CLP version supported by the implementation. The implementation shall include the version of the *Server Management Managed Element (SM ME) Addressing Specification* with which it is conformant.

6.14.4.2 Structured Format

This clause details requirements for structured output formats for the `version` verb.

6.14.4.2.1 General

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

6.14.4.2.2 XML Output

The implementation shall return the `version` element in the `response` element as defined in the Command Response schema in DSP0224. The implementation shall return the exact string "`v1.0.0`" as the data contained in the `clpversion` element within the `version` element.

6.14.4.2.3 Keyword

Implementations shall use the following form when returning Command Results for the `version` command in "keyword" format:

```
command=version
clpversion=v1.0.0
addressingversion=v1.0.0
endoutput
```

6.14.5 Examples

This clause provides examples of the use of the `version` verb.

EXAMPLE 1: Runs the `version` command the way it is meant to be (that is, without targets).

```
-> version
SM CLP Version 1.0.0
SM ME Addressing Version 1.0.0
```

EXAMPLE 2: Attempts to include targets with the `version` command, but the `version` command does not support targets.

```
-> version /system1
   Invalid syntax; the version command does not support a target.
```

EXAMPLE 3: Specifies invalid syntax for the `version` command.

```
-> version –o format=clpxml /system1
<?xml version="1.0" encoding="UTF-8"?>
<response xmlns="http://schemas.dmtf.org/smash/commandresponse/1.0.0/
dsp0224.xsd" xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xsi:schemaLocation="http://schemas.dmtf.org/smash/commandresponse/1.0.0
/dsp0224.xsd 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>
```

EXAMPLE 4: Successfully runs the `version` command.

```
-> version –o format=clpxml
<?xml version="1.0" encoding="UTF-8"?>
<response
xmlns="http://schemas.dmtf.org/smash/commandresponse/1.0.0/dsp0224.xsd"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xsi:schemaLocation="http://schemas.dmtf.org/smash/commandresponse/1.0.0
/dsp0224.xsd 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>
      <clpversion>v1.0.0</clpversion>
      <addressingversion>v1.0.0</addressingversion>
   </version>
</response>
```

EXAMPLE 5: Successfully runs the `version` command.

```
-> version –o format=keyword
command=version –o format=keyword
clpversion=v1.0.0
addressingversion=v1.0.0
endoutput
```

7 Standard Command Options

This clause describes the requirements for supporting and implementing standard command options.

7.1 General

Every CLP option is defined to have the same name and behavior across the CLP command verb set. For each CLP verb, implementations shall 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.

EXAMPLE 1: Use of "-o" always indicates the `output` option whenever used and follows the behavior specified for the `output` standard option.

The clauses below describe each standard option and its behavior.

Implementations shall 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 shall not recognize option names of any other length. That is, "partial" or "two letter" option names are not allowed.

EXAMPLE 2: The option name for controlling output is 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 shall 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. For each CLP Command processed, implementations shall apply every standard option that is supported for the verb. Implementations shall observe the following order of precedence for determining the value of a standard option: specification default, superseded by the session default, and finally superseded by the explicit Command Line setting, if present.

Implementations shall observe the following order of precedence for the following standard options: `help`, `version`, and `examine`. If the `help` option appears on the Command Line, implementations shall provide the behavior specified in **Error! Reference source not found.**. If the `version` option appears, but the `help` option does not, implementations shall provide the behavior specified in **Error! Reference source not found.**. If the `examine` option appears on the Command Line without either the `help` or `version` option, the implementation shall provide the behavior described in **Error! Reference source not found.**. These precedence rules do not affect the processing of standard options or other options specified in the Command Line.

If an option is included more than once in a command, the implementation shall not execute the command. The implementation shall 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 that requires an argument is specified in a command without an argument value, the implementation shall return a Command Status of COMMAND PROCESSING FAILED and a Processing Error of MISSING ARGUMENT. If a CLP option that does not require an argument is specified in a command with an argument value, the implementation shall return a Command Status of COMMAND PROCESSING FAILED and a Processing Error of COMMAND SYNTAX ERROR. If a CLP option is specified with an argument that is not defined by the CLP as a legal argument and the argument is not identified as an OEM-defined argument according to the rules in 5.2.6.2, the implementation shall return a Command Status of COMMAND PROCESSING FAILED and a Processing Error of INVALID ARGUMENT.

Implementations shall support only those options listed in Table 14. Implementations shall support all of the options listed in Table 14. Implementations shall support arguments on all options that specify an argument in Table 14. Implementations shall not support arguments for those options that do not specify an argument in Table 14.

For each option listed in Table 14 that has a value of "yes" in the column labeled "Session Default Supported", the implementation shall allow the user to set a default value per session. For each option listed in Table 14 that does not have a value of "yes" in the column labeled "Session Default Supported", the implementation shall 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 that represents the CLP session. For information on the CIM class and its properties, see the *CLP Service Profile* (DSP1005).

Table 14 – 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	
`-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.	Multiple arguments controlling the type of information returned about a target.	yes
`-examine`	`-x`	Instructs the Command Processor to examine the command for syntax and semantic errors but not execute the command.	None	
`-force`	`-f`	Instructs the 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"	

Option Name	Short Form	Interpretation	Argument	Session Default Supported
`-output <args>`	`-o`	Controls the content and form of the command output.	Many arguments providing control of format, language, level of detail, order, and so on of output data.	yes
`-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 the Command Response until all spawned jobs have completed.	None	yes

Table 15**Error! Reference source not found.** specifies the requirements for implementations to support each combination of verb and standard option. Except where noted, a cell with an *X* indicates that the implementation shall support use of the option named in the row with the verb named in the column.

Table 15 – Verb and Option Support

CLP Option	CLP Verb												
	cd	create	delete	dump	exit	help	load	reset	set	show	start	stop	version
all										X			
destination				X									
display										X			
examine	X	X	X	X	X	X	X	X	X	X	X	X	X
force			X^2	X^1			X^1	X^2	X^2	X^1	X^2	X^2	
help	X	X	X	X	X	X	X	X	X	X	X	X	X
keep	X	X	X	X	X	X	X	X	X	X	X	X	X
level										X			
output	X	X	X	X	X	X	X	X	X	X	X	X	X
source							X						
version	X	X	X	X	X	X	X	X	X	X	X	X	X
wait	X	X	X	X		X	X	X	X	X	X	X	X

Notes: X^1 – The implementation *may* support use of the `force` option with the `dump`, `load`, and `show` verbs.
X^2 – The implementation *should* support use of the `force` option with the `delete`, `reset`, `set`, `start`, and `stop` verbs.

If the cell in Table 15**Error! Reference source not found.** is blank, the implementation shall not support the use of the option named in the row with the verb named in the column. If a Command Line is specified with a verb and option combination that shall not be supported according to **Error! Reference source not found.**, the implementation shall not execute the command and shall return a Command Status of COMMAND PROCESSING FAILED and a Processing Error of INVALID OPTION. If the implementation supports an option that may or should be supported according to Table 15**Error! Reference source not**

found. for one verb, the implementation may support the option with other verbs that may or should be supported according to Table 15**Error! Reference source not found.**. If a Command Line is specified with a verb and option combination that may or should be supported according to Table 15**Error! Reference source not found.** and is not supported by the implementation, the implementation shall not execute the command and shall return a Command Status of COMMAND PROCESSING FAILED and a Processing Error of OPTION NOT SUPPORTED.

7.2 all

This clause describes the requirements for the `all` option.

7.2.1 Forms

```
-all
-a
```

7.2.2 Example Use

```
show –a log1
```

7.2.3 Behavior

The `all` option instructs the Command Processor to select all values where appropriate for the command. This option is supported only with the `show` command. For a detailed description of its use, see 6.11.3.

When the `all` option is specified, the implementation will select all values where appropriate for the command.

7.3 destination

This clause describes the requirements for the `destination` option.

7.3.1 Forms

```
-destination
```

7.3.2 Example Use

```
dump –destination
ftp://administrator:passw0rd@myserver.com/private/administrator/`memory.dmp
/system1/memory1
```

7.3.3 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 can be expressed as a UFiP if it is a Managed Element in the address space of the MAP. The destination can also be expressed as a URI. The desired protocol to use for transferring the image can be specified as part of the URI. This specification does not mandate support for a specific protocol. Thus the protocols supported are implementation specific. Implementations shall interpret the value specified as a URI defined according to *Uniform Resource Identifiers (URI): Generic Syntax* (RFC2396). The URI specified may contain a scheme that indicates the explicit service and location to be used to capture the dumped data. If the URI specified is relative, then the implementation shall interpret the URI according to the rules specified in 5.1.3.3.

7.4 display

This clause describes the requirements for the `display` option.

7.4.1 Forms

```
-d[isplay] <type>*[,<type>]
```

7.4.2 Example Use

EXAMPLE 1: Shows the commands that are supported for use with `log1` as a target.

```
show –display verbs log1
```

EXAMPLE 2: Shows all of the systems in the address space. No filtering of the results for individual instances is requested.

```
show –level all –display targets=system /
```

EXAMPLE 3: Shows all of the CPUs and SystemDevice association instances in `system1`. No filtering of the results for individual instances is requested.

```
show –level all –display targets=cpu,associations=systemdevice `
/system1
```

EXAMPLE 4: Shows the OperationalStatus property for every power supply and processor in `system1`.

```
show –level all –display ` targets=(cpu,pwrsupply),properties=
operationalstatus ` /system1
```

EXAMPLE 5: Shows the ElementName property for every instance.

```
show –level all –display properties=(elementname) /
```

7.4.3 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. 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 or the verbs (or a subset) supported for an instance. For any two commands that are identical except for arguments to the `display` option, the implementation shall interpret and execute the command such that the set of Targets affected, and the resultant state of those Targets, is identical. The `display` option requires one or more arguments specifying the category of information to include in the Command Results. For each command, the implementation shall not include data in the Command Results returned to the Client unless it is specified by the arguments for the `display` option that are in effect for the command.

The valid types and formats for the arguments of the `display` option are as follows:

`all` Perform no filtering of the results.

`verbs` Display the commands that are valid for this target.

`properties[="("(<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. Can also be used to filter the Command Results so 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 include information about Managed Element instances only.

associations[="("(classname][,<classname>,<classname>,...,<classname>")"]

Filter the Command Results to include information about associations only.

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 (restrict to just properties returned for an instance)
- properties (propname, restrict properties that are returned for an instance)
- verbs

The specification default argument for the display option is targets,properties,verbs.

When the display option is specified with one or more arguments, the implementation shall apply each of the arguments to the operation results according to the rules for each argument. For each argument present in the Command Line, implementations shall apply the arguments in the following order irrespective of the order in which they appear in the Command Line: targets, associations, properties, verbs. When the display option is specified with an argument of all, the implementation shall 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 shall remove the leading and trailing parentheses, if they are present. The filtering provided for by the display option is applied across the entire set of Managed Elements and Associations for which results were generated by the command. For example, when the show command is used with the level option or with the Target Class Addressing, a tree of Managed Elements and Associations can be returned. The filtering would then be applied across each level of the tree, removing any Managed Elements that did not match the target filter, removing any Associations that did not match the association filter, filtering properties returned on any remaining Managed Elements and Associations according to the properties filter, and filtering the list of verbs supported for any remaining Managed Elements. Note that for an individual Association instance, the verbs argument has no effect because Command Results for an instance of an Association do not include information about the verbs supported for an instance. Implementations will accept duplicate values within the comma-delimited list of values for each argument to the display option.

When the display option is specified with an argument of targets and no value is supplied to the argument, the implementation shall 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 shall interpret the supplied value as a comma-delimited, ordered list of UFcTs. The implementation shall apply the ordered list of UFcTs as a filter on the instances for which results are returned. The implementation shall 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 shall add the results for instances in the order their corresponding types were specified in the UFcT list. When selecting instances for which results will be included in the Command Results, the implementation shall apply the "Rules for Selecting Instances by UFcT" defined in DSP0215 for each UFcT supplied as a value to the targets argument and each instance whose results are included in the operation results, where the Selection UFcT is the UFcT supplied as an argument value and the target instance is the instance whose results are included in the operation results.

When the display option is specified with an argument of associations and no value is supplied to the argument, the implementation shall 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 shall interpret the supplied value as a comma-delimited, ordered list of Association Classes. The implementation shall apply the ordered list of Association Classes as a filter on the instances for which results are returned. The implementation shall 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 shall add the results for instances in the order their corresponding types were specified in the list of class names.

The `properties` argument to the `display` option can be used to filter the results returned for a target to be the properties of the target or a specific subset of the properties. Usage is supported with or without additional values. When the `display` option is specified with an argument of `properties`, implementations shall 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 shall interpret the value as a comma-delimited, ordered list of tokens.

The implementation shall 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 shall 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 shall filter the intermediate results and add to the Command Results for an instance information about the verbs that are supported for the instance.

It is 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 shall remove the instance from the Command Results that are returned.

7.5 examine

This clause describes the requirements for the `examine` option.

7.5.1 Forms

```
-examine
-x
```

7.5.2 Example Use

```
stop –force –x /system3
```

7.5.3 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. If the `examine` option is included in the Command Line, the implementation will perform the standard validation of the Command Line but will not execute the command. Thus, any errors that would result in a Command Status of COMMAND PROCESSING FAILED and the accompanying Processing Error if the `examine` option were not specified will still result in a Command Status of COMMAND PROCESSING FAILED and the accompanying Processing Error. Any errors that would result in a COMMAND EXECUTION FAILED will be reported as free-form text in the Command Results and not as a Command Status.

When the `examine` option is included in the Command Line, the Command Processor shall not commit or execute the command. The Command Processor shall verify that the options, Resultant Target, and target properties are valid. The Command Results may describe what action would be taken if the command were to be entered without the `examine` option. Although the Command Processor will not execute the command, applying the `examine` option to the Command Line is itself an activity of the MAP that will result in a Job being created.

7.6 force

This clause describes the requirements for the `force` option.

7.6.1 Forms

```
-force
-f
```

7.6.2 Example Use

```
stop –f /system3/blade2
```

7.6.3 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 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 shall execute the command.

7.7 help

This clause describes the requirements for the `help` option.

7.7.1 Forms

```
-help
-h
```

7.7.2 Example Use

```
stop –help
```

7.7.3 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 its 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 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 shall not take the action specified by the verb. The implementation shall return text describing the proper use of the verb entered as the first term on the Command Line.

7.8 keep

This clause describes the requirements for the `keep` option.

7.8.1 Forms

```
-keep <m[.s]>
-k <m[.s]>
```

m.s is an amount of time specified as 'minutes.seconds'.

7.8.2 Example Use

```
reset –k 10 system1
```

Resets `system1` and retains the Command Status for 10 minutes.

7.8.3 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 the status for a command be preserved, it must be included as part of the initial command request.

Implementations shall 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 shall set the TimeBeforeRemoval property of the CIM_ConcreteJob instance to the value specified by the `keep` option. Implementations may maintain Command Results after the command execution completes even if the `keep` option is used. The `keep` option only controls how long the implementation is required to maintain the Command Status.

If the CLP session ends before the command has completed and the 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.

7.9 level

This clause describes the requirements for the `level` option.

7.9.1 Forms

```
-level <n>
-l <n>
```

n is the number of levels to include in the command scope.

7.9.2 Example Use

EXAMPLE 1: Shows information about the default target and one level of contained Managed Elements.

```
show –l 2
```

EXAMPLE 2: Shows information about Managed Elements up to two levels deep.

```
show –l 3
```

EXAMPLE 3: Stops all contained Managed Elements, and then stops the command target.

```
stop –l all
```

EXAMPLE 4: Shows information about `system3` and all contained Managed Elements.

```
show –l all system3
```

7.9.3 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 5.2.1.3.5.

Any levels specified that extend beyond the containment depth of the command target shall 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 Managed Elements.

n=3 Verb acts on the command target, directly contained Managed Elements, and any Managed Elements contained by those Managed Elements (that is, the current target and "two down").

n=all Verb acts on the command target and all target Managed Elements recursively contained in the command target.

When the `level` option is included with a command, the implementation shall 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.

7.10 output

This clause describes the requirements for the `output` option.

7.10.1 Forms

```
-output <arguments>
-o <arguments>
```

Table 16 lists the arguments for the output option.

Table 16 – Output Option Arguments

Argument	Value Domain (default in bold)	Description
format=<*value*>	**"text"**, "keyword","clpxml"	`format` controls the structure of the output text.
error, **terse**, verbose		This argument 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` and `end` control where to start in the list.

Argument	Value Domain (default in bold)	Description
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, count controls the number of items returned (for example, log items); the default is **'all'** items. The maximum value for count is determined by the class of the target.
number=<x-y>	[integer string]-[integer string]	number requests that a range of results be returned.

7.10.2 Example Use

```
show –output format=clpxml,verbose,order=reverse,number=5 ./log1/record*
show –o verbose /system2/log1
start -o terse system1
```

7.10.3 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 Targets affected by the command.

7.10.3.1 General Requirements

For any two commands that are identical except for arguments to the output option, the implementation shall interpret and execute the command such that the Targets affected, and the resultant state of those Targets, is identical.

EXAMPLE: The following command would cause all of the instances of the UFcT in the container to be deleted, not just four of them.

```
delete –output count=4 UFcT
```

Prior to processing the value for an argument, the implementation shall remove the leading and trailing parentheses if they are present. For each combination of arguments to the output option, implementations shall return identical Command Results irrespective of the relative order in which the arguments appear on the Command Line.

Implementations shall return Command Results consistent with applying arguments to the output option in the following order: number, begin, end, count, order, format, terse, error, verbose, and language.

7.10.3.2 Output Format Selection

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 wants 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 shall 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 shall 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 shall not execute the command and the implementation shall 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 shall not execute the command and the implementation shall return Command Response data compliant with session default output format indicating a Processing Error of INVALID ARGUMENT.

Implementations shall recognize the string `text` as identifying the CLP text output format and shall not recognize any other string as identifying the text output format. Implementations shall recognize the string `keyword` as identifying the CLP keyword/value output format and shall not recognize any other string as identifying the keyword/value output format. Implementations shall recognize the string `clpxml` as identifying the CLP XML output format and shall not recognize any other string as identifying the XML output format.

7.10.3.3 Output Language Selection

If the `output` option with the language argument is included in a command, the implementation shall attempt to interpret the value of the language argument as a language code defined in accordance with ISO 639-2. If the implementation recognizes the value of the language argument as identifying an output language it supports, the implementation shall 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 shall not execute the command, the implementation shall return Command Response data in the session default language indicating a Processing Error of INVALID ARGUMENT.

7.10.3.4 Output Range and Order Selection

Some CLP verbs can return Command Results that include results for more than one Managed Element. The results of a command are marshaled into operation results data. The operation results data contains the results for each Managed Element as an ordered list. For each Managed Element, DSP0216 defines the algorithm for initially ordering the operation results into the natural (default) order. Note that randomized is considered a valid algorithm for ordering the results. Arguments to the `output` option that select a range of elements for which results will be returned or control the order in which results are returned will operate against this list of operation results. The `begin` argument instruments the implementation to select the range of elements for which results will be included from the beginning of the list. The `end` argument instruments the implementation to select the range of elements for which results will be included from the end of the list. The `count` argument allows a user to restrict the number of Managed Elements in the range. The `order` argument allows a user to modify the order in which results are returned for a command. The `number` argument allows a user to select a range of Managed Elements.

The `begin`, `end`, `number`, `order`, and `count` arguments to the `output` option affect the order and range of results displayed. When operation results are returned for Managed Elements of more than one type, the implementation shall apply the range selection once for each Managed Element type to the range of instances of that type. Implementations shall not apply the range selection arguments within the result for an individual Managed Element. Implementations shall apply the output order and range selection arguments to the elements at the deepest level of the containment hierarchy for which operation results have been generated and shall not apply the range or output order selection arguments to instances at any other level of the containment hierarchy for which operation results were generated. 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 shall not execute the command and shall return a Command Status of COMMAND PROCESSING FAILED with a Processing Error of COMMAND SYNTAX ERROR. The `number` and `count` arguments are mutually exclusive. If the implementation receives a command with the `output` option specified with more than one of `number` and `count` specified, the implementation shall not

execute the command and shall return a Command Status of COMMAND PROCESSING FAILED with a Processing Error of COMMAND SYNTAX ERROR.

When the `output` option is specified with an argument of `begin`, the implementation shall use the first element as the beginning element in the range of elements for which results will be returned. When the `output` option is specified with an argument of `end`, the implementation shall use the last element as the ending element in the range of elements for which results will be returned. Implementations shall then apply the `count` argument to determine the number of elements for which operation results are included in the Command Results. When the `output` option is specified with an argument of `count`, the implementation shall not return results for more Managed Elements than the value specified for `count`. Note that the value for the `count` argument is interpreted as an upper limit only. If the operation results contain results for fewer Managed Elements than the value specified for the `count` argument, the `count` argument will not have any effect.

When the `output` option is specified with an argument of `order` and the value selected for `order` is `default`, the implementation shall 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 shall return results in reverse of the natural order for the class. If the `end` argument is not specified, the implementation shall behave as if the `begin` argument was specified. If the `order=reverse` argument is not specified, the implementation shall behave as if `order=default` was specified. Thus the default behavior is to return elements according to their natural order.

The `number` argument allows a user to select a range of Managed Elements. The value of the `number` argument is in the following format:

```
<starting identifier>-<ending identifier>
```

Range selection is inclusive of the start and ending identifier. Any non-negative integer is an acceptable value for the `starting identifier` and `ending identifier`. If the `number` argument is specified with a value for the `starting identifier` or `ending identifier` that is not a non-negative integer, the implementation shall return a Command Status of COMMAND PROCESSING FAILED and a Processing Error of INVALID ARGUMENT. When the `output` option is specified with an argument of `number`, the implementation shall select the range of instances identified by `starting identifier` through the `ending identifier`, inclusive from the list of instances for which results are contained in the operation results. If the value specified for the `starting identifier` is greater than the value specified for the `ending identifier`, the implementation shall return a Command Status of COMMAND PROCESSING FAILED and a Processing Error of INVALID ARGUMENT. If the value for the `ending identifier` is greater than the number of elements in the list of operation results, the implementation shall select the last element in the list of operation results as the end element in the range for which results will be returned. If the value for the `starting identifier` is greater than the number of elements in the list of operation results, the implementation shall not return results for any Managed Elements.

The following examples are provided for informational purposes and do not constitute additional constraints on the implementation. For these examples, assume there are nine instances of a class where the natural ordering of their results is UFcT*1*, UFcT*3*, UFcT*4*, UFcT*5*, UFcT*6*, UFcT*7*, UFcT*8*, UFcT*9*, UFcT*10*. Note that the results do not contain an instance of UFcT*2* in this example.

EXAMPLE 1:

```
- o begin
```

`begin` is the only argument to the `output` option. The implementation will select instance UFcT*1* as the beginning element for the range of results. The `count` argument is not specified, so the default value of "all" is used. The implementation will select all elements, starting at UFcT*1*. The `order` argument is not specified, so the default ordering will be used. Thus, results will be returned in the default ordering of UFcT*1*, UFcT*3*, UFcT*4*, UFcT*5*, UFcT*6*, UFcT*7*, UFcT*8*, UFcT*9*, UFcT*10*.

EXAMPLE 2:

```
- o begin,count=3
```

The implementation will select instance UFcT*1* as the beginning element for the range of results. The `count` is specified with a value of 3, so the implementation will select the first three instances. The `order` argument is not specified, so the default ordering will be used. Thus, results will be returned in the default ordering of UFcT*1*, UFcT*3*, UFcT*4.*

EXAMPLE 3:

```
-o end
```

`end` is the only argument to the `output` option. The implementation will select instance UFcT*10* as the end element for the range of results. The `count` argument is not specified, so the default value of "all" is used. The implementation will select all elements, ending at UFcT*10*. The `order` argument is not specified, so the default ordering will be used. Thus, results will be returned in the default ordering of UFcT*1*, UFcT*3*, UFcT*4*, UFcT*5*, UFcT*6*, UFcT*7*, UFcT*8*, UFcT*9*, UFcT*10.*

EXAMPLE 4:

```
-o end,count=3
```

`end` is specified, so the implementation will select instance UFcT*10* as the end element for the range of results. The `count` argument is specified with a value of 3, so the implementation will select the last three elements ending at UFcT*10*. The `order` argument is not specified, so the default ordering will be used. Thus, results will be returned in the default ordering of UFcT*8*, UFcT*9*, UFcT*10.*

EXAMPLE 5:

```
-o end,order=reverse
```

As in EXAMPLE 3, the implementation will return results for all instances. However, they will be returned in the reverse order of UFcT*10*…UFcT*3*, UFcT*1*.

EXAMPLE 6:

```
-o number=(1-5)
```

The implementation will select the first item in the list of operation results as the first element to include in the range of results. The implementation will select the fifth item in the list of operation results as the last element to include in the range of results. The implementation will include all of the elements in between. The results will be returned as UFcT*1*, UFcT*3,* UFcT*4,* UFcT*5*, UFcT*6.*

7.10.3.5 Output Verbosity

The `terse`, `verbose`, and `error` arguments are used to specify the level of detail an implementation returns when the output format is "text". If the implementation receives a command with the `output` option specified with more than one of the `terse`, `error`, and `verbose` arguments specified, the implementation shall not execute the command and shall return a Command Status of COMMAND PROCESSING FAILED with a Processing Error of COMMAND SYNTAX ERROR.

The `terse`, `verbose`, and `error` arguments have no effect when the output format is a structured output format. When the output format in effect for a command is a structured output and the `terse`, `verbose`, and `error` arguments are in effect for a command, the `terse`, `verbose`, and `error` arguments are ignored.

When the output format for a command is "text" and the `output` option is specified with an argument of `terse`, the implementation shall return Command Status that includes only the Status Tag data element, unless a processing error or execution error occurs, in which case the implementation shall return the Processing Error Tag or Job Error data element as appropriate to the error that occurred.

When the output format for a command is "text" and the `output` option is specified with an argument of `error`, the implementation shall return Command Status if an error occurs in processing the command and shall not return Command Status if an error does not occur in 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 shall return Command Status that includes the Status Tag. If a processing error or execution error occurs, the implementation shall return the Processing Error Tag or Job Error data element as appropriate to the error that occurred. The implementation may return Message data elements.

Table 17 lists the supported arguments for the output option that control the level of detail returned by a command when the output format is text mode.

Table 17 – Output Options for Controlling 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 returned only if an error occurs.

7.11 source

This clauses describes the requirements for the `source` option.

7.11.1 Forms

```
-source
```

7.11.2 Example Use

```
load –source `
ftp://administrator:passw0rd@myserver.com/private/administrator/`firmware.img
/system1/fw1
```

7.11.3 Behavior

The `source` option is used to specify the source of data for a command. The source can be expressed as a UFiP if it is a Managed Element in the address space of the MAP. The source can 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. Implementations shall interpret the value specified as a URI defined according to RFC2396. The URI specified may contain a scheme that indicates the explicit service and location to be used to transfer the source data. If the URI specified is relative, then the implementation shall interpret the URI according to the rules specified in 5.1.3.3.

7.12 version

This clause describes the requirements for the `version` option.

7.12.1 Forms

```
–version
–v
```

7.12.2 Example Use

EXAMPLE 1: Displays the version of the `start` command verb.

```
start –version
```

7.12.3 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 shall not execute the command. The implementation shall return Command Response data that identifies the version of the verb supported by the implementation.

7.13 wait

This clause describes the requirements for the `wait` option.

7.13.1 Forms

```
-wait
-w
```

7.13.2 Example Use

```
stop –w /system3
```

7.13.3 Behavior

When the `wait` option is included in a command, the implementation shall not return control immediately to the Client; instead the implementation shall 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.

8 SM CLP Session

The following clauses describe the requirements for management of CLP sessions.

8.1 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 that 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 shall 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 shall 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 shall not complete the requested command and the implementation shall return a Command Status of COMMAND EXECUTION FAILED, a Job Error Type of Security Error, and a CIM Status of CIM_ERR_ACCESS_DENIED.

Table 18 lists the CLP-defined user groups and the associated authorized CLP commands. For each user group listed in the "Group" column, implementations shall authorize the CLP commands listed in the "SM CLP Commands" column and shall not authorize CLP commands that do not appear in the "SM CLP Commands" column.

Table 18 – 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`

Prior to allowing a Client to issue any SM CLP commands, implementations shall have an authenticated session established between the Client and the CLP Service. Every CLP session shall exist in the context of a CLP User where the context is determined when the session is established and cannot be modified for the session. Establishment of an authenticated session is implementation specific and is outside the scope of this specification.

Implementations shall support adding, removing, displaying, and modifying user accounts through the CLP. Implementations shall 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 18. For more information, see DSP0216. The representation of CLP users and groups is defined in DSP1005. The definition of the initial user account for accessing the CLP is implementation specific and outside the scope of this specification.

8.2 CLP Session

This clause describes the CLP session.

8.2.1 General

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 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 shall not accept any additional commands from the user and shall finish processing the pending command, if there is one. The implementation shall then end the session as if it were processing an `exit` command from the user. If an immediate stop of the CLP

session is initiated, the implementation shall immediately stop the CLP session and shall not return a Command Response to the user. When terminating the CLP session, the implementation may also terminate the transport session.

8.2.2 Session Environment

Interaction with the CLP Service is done within the context of a session. The CLP Service maintains a session context for each active session. The SESSION term (see 5.1.7) references the session in which the term is used. This provides a convenient mechanism for users 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 supported only on a subset of options. For a complete list of options and session attributes that implementations are required to support, see DSP0216. Implementations shall allow users to modify settings for their own sessions irrespective of the groups of which the users are a member. This is an exception to the rules in Table 18.

8.2.3 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 if the user has Administrator authorization.

Implementations shall 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 shall gracefully stop each active CLP session as described in 8.2.1 prior to ending the service. If an immediate stop of the CLP Service is requested, the implementation shall immediately stop each active CLP session as described in 8.2.1 prior to ending the service.

8.2.4 Initial Prompt

When a CLP session is established, implementations shall return the following string followed by the first command prompt:

```
=== SM CLP v<major>.<minor>.<patchlevel> SM ME Addressing
v<major>.<minor>.<patchlevel> <OEM Data> ===
```

The version string that follows "SM CLP" is in the m.n.ud format specified by the *DMTF Release Process, v1.3* (24HDSP4004) and identifies the version of the SM CLP specification supported by the implementation. For implementations compliant with this version of the specification, <major> has a value of 1, <minor> has a value of 0 (zero), and <patchlevel> has a value of 0 (zero). The version string that follows "SM ME Addressing" is in the m.n.ud format specified by 24HDSP4004 and identifies the version of the SM ME Addressing specification supported by the implementation. <OEM Data> is a vendor-defined versioning string. The initial CLP prompt is this version information followed by a command prompt

8.2.5 CLP Interaction Model

The CLP observes the following interaction models documented in this clause, unless otherwise noted:

- CLP Service Discovery (documented in the *CLP Service Discovery Specification* [DSP0218])
- CLP Session Establishment
- CLP Command Interaction
- CLP Session Switching
- CLP Session Termination

The following clauses 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 behavior is outside the scope of this specification.
- Solid horizontal lines indicate behavior required by this specification.

8.2.5.1 Session Establishment

Figure 1 provides an overview of the session establishment sequence.

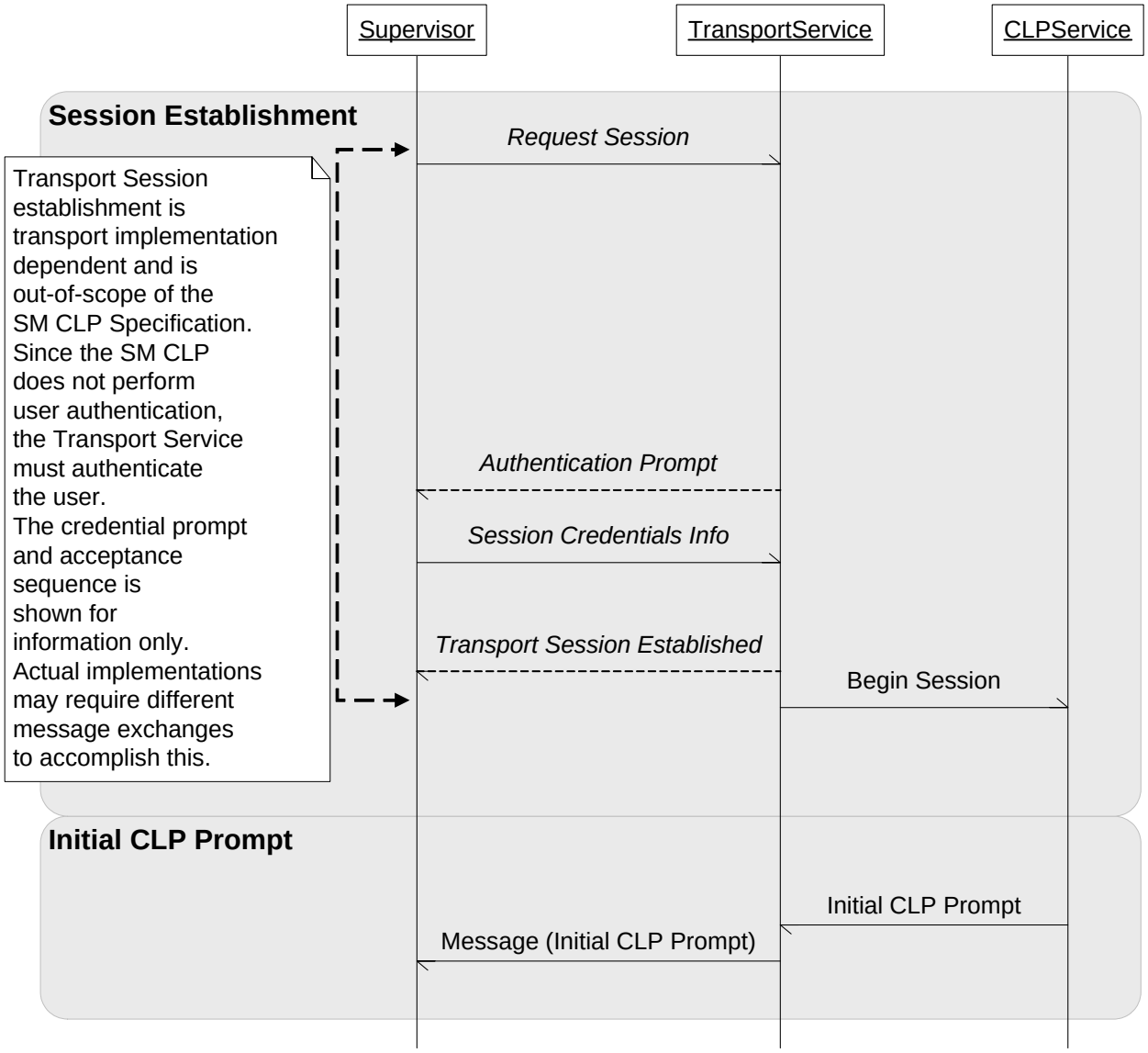
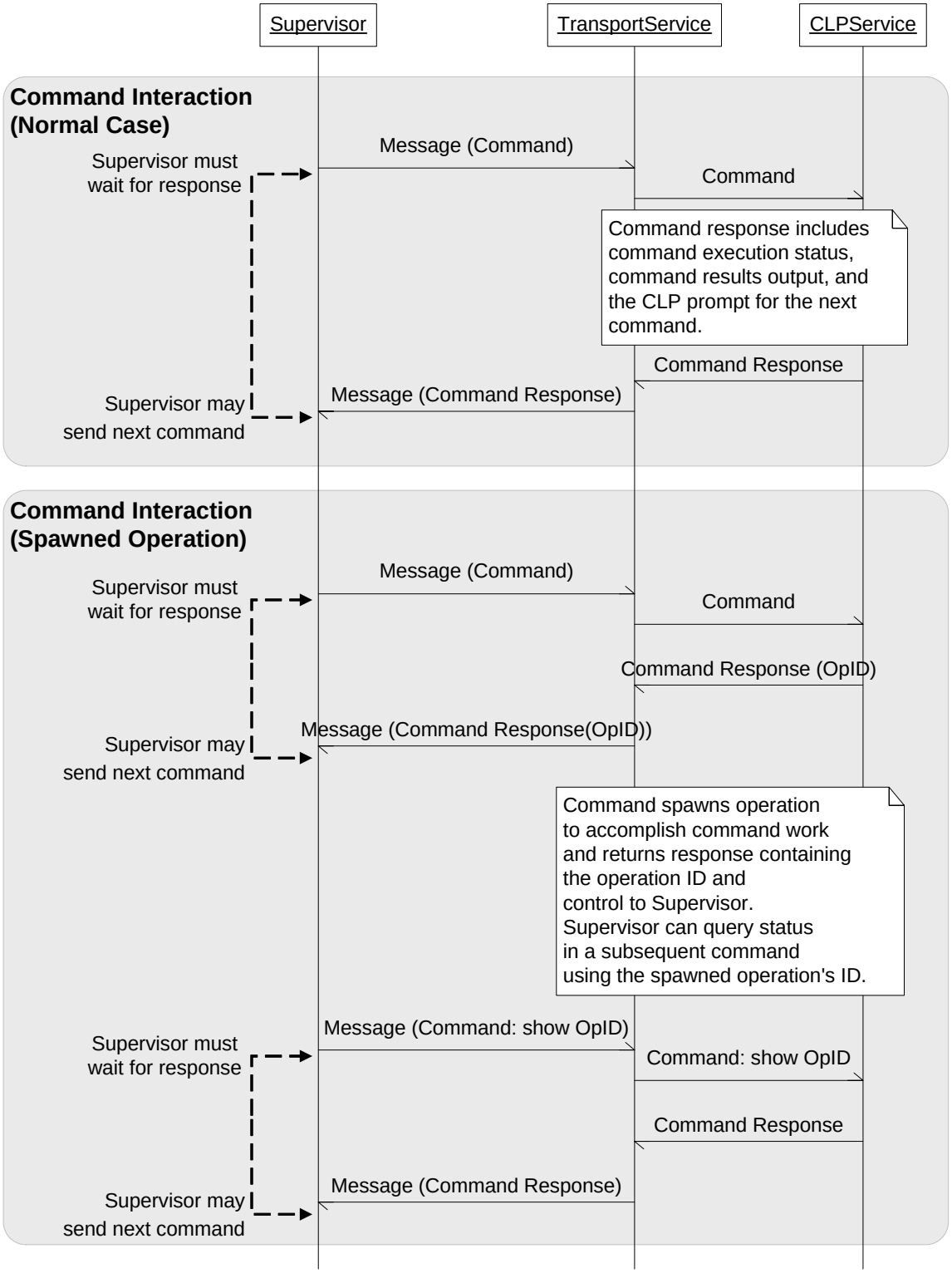


Figure 1 – Session Establishment Sequence

8.2.5.2 Command Interaction

Figure 2 provides an overview of the interaction between a Supervisor and a CLP Service for a CLP command. Two cases are shown. In the first case, a Command Response is returned synchronously. In

5959 the second case, a command executes asynchronously and the Supervisor polls for status.



5960

5961 **Figure 2 – Command Interaction Sequences**

8.2.5.3 Session Switching

Figure 3 provides an overview of the interaction between a Supervisor and the CLP Service when an alternate text session is initiated using the CLP. The alternate text session is established within the CLP session. Therefore, while the alternate text session is established, the Supervisor interacts with the alternate text service and not the CLP Service.

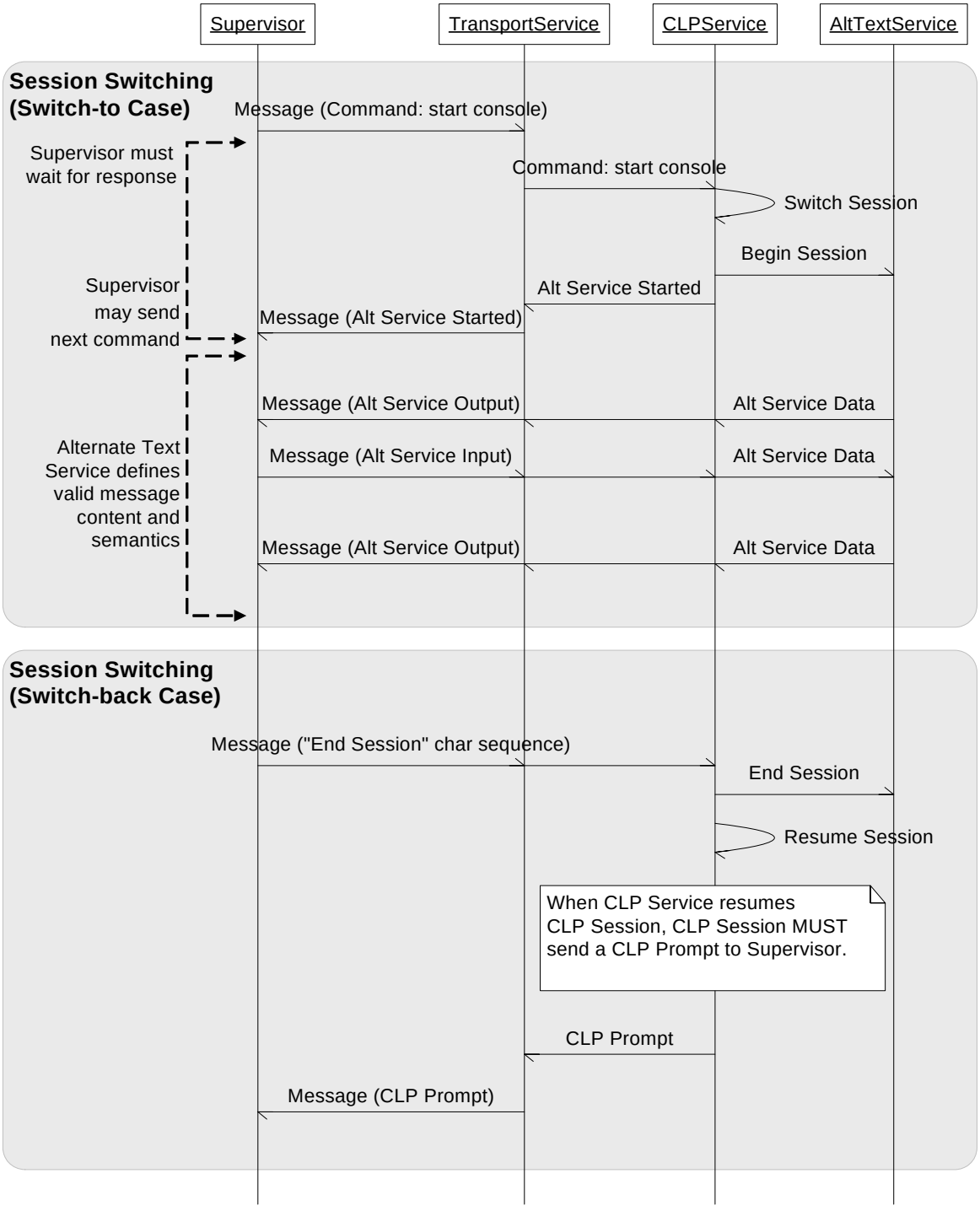


Figure 3 – Session Switching Sequences

8.2.5.4 Session Termination

Figure 4 provides an overview of the interaction between a Supervisor and CLP Service when a session terminates. Two cases are shown. In the first case, the Supervisor requests their own session be terminated. In the second case, a different Supervisor requests that the session of the first Supervisor be terminated.

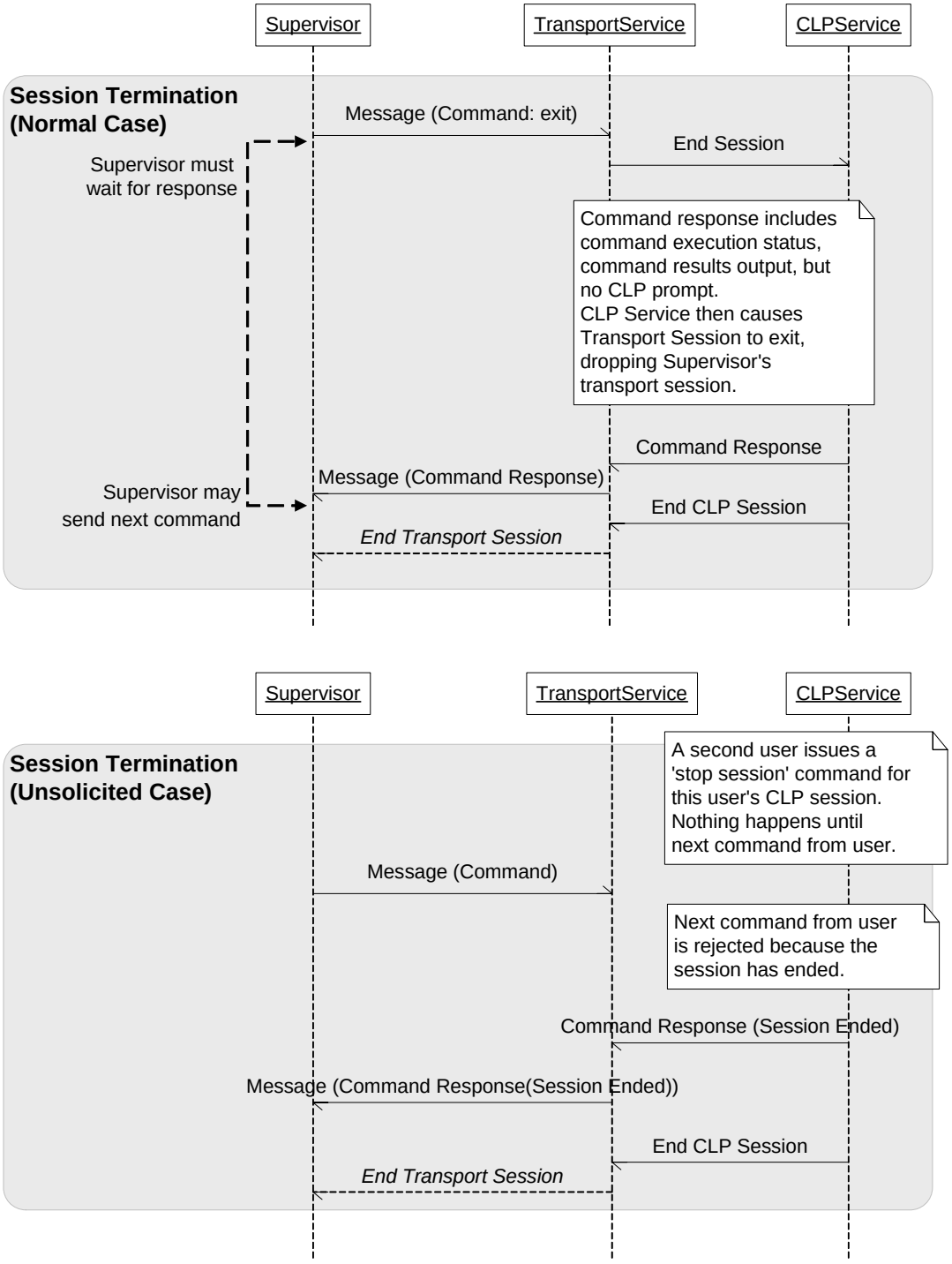


Figure 4 – Session Termination Sequences

8.3 Transport

The CLP is carried over a text message-based protocol. At present, two text message protocol mappings of the CLP are specified:

- Telnet
- Secure Shell (SSH)

8.3.1 General

CLP implementations shall 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 shall 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 directly affects the operation of the implementation or its terminal I/O operations as seen from the transport client application. Implementations of the CLP Service shall 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 uses a transport that 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.

8.3.2 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) can 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 as follows:

- The data representation is 7-bit ASCII transmitted in 8-bit bytes.
- The NVT provides a local echo function.

A CLP implementation over Telnet shall support the base Telnet RFC and Standards and shall support the IETF RFCs listed in Table 19.

Table 19 – Telnet Transport Support Requirements

Number	Title	Author or Ed.	Date Released	Status
STD0008 RFC0854	*Telnet Protocol Specification*	J. Postel, J.K. Reynolds	May 1, 1983	Standard
STD0008 RFC0855	*Telnet Option Specifications*	J. Postel, J.K. Reynolds	May 1, 1983	Standard

Note that, in addition to standard NVT operation as described in 22HRFC0854, implementations shall respond to requests to support UTF-8.

Each line should be terminated by a Carriage Return and Line Feed (UTF-8, from a *Transformation Format of ISO 10646* [RFC3629]).

A CLP implementation may listen on any TCP port number. CLP TCP port numbers may be administratively set or discovered through the Service Location Protocol (SLP). See 29HDSP0218 for more information.

8.3.3 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 the following five core documents. A CLP implementation over SSHv2 shall support the base SSHv2 RFC and Standards and the following IETF RFCs and Internet Drafts:

- *Secure Shell (SSH) Protocol Architecture* (17HRFC4251)—Describes the overall design of SSH.
- *Secure Shell (SSH) Transport Layer Protocol* (18HRFC4253)—Provides a single, full-duplex, byte-oriented connection between client and server, with privacy, integrity, server authentication, and man-in-the-middle protection.
- *Secure Shell (SSH) Authentication Protocol* (19HRFC4252)—Identifies the client to the server.
- *Secure Shell (SSH) Connection Protocol* (20HRFC4254)—Provides richer, application-support services over the transport pipe, such as channel multiplexing, flow control, remote program execution, signal propagation, connection forwarding, and so on.
- *Secure Shell (SSH) Protocol Assigned Numbers* (RFC4250)—Lists various constant assignments made in the other drafts.

The Secure command-shell mode of operation shall be supported by a CLP implementation.

A CLP implementation is not required to listen on any specific predefined TCP port number. A CLP TCP port number may be administratively set or discovered through the SLP. See DSP0218 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.

When an implementation supports SSH, the implementation shall support a mechanism to restrict access to the implementation using tunneled protocols, and the factory-default setting shall disable Telnet as a tunneled protocol.

Annex A
(normative)

SM CLP Command Grammar in ABNF Notation (RFC2234)

```
;;
;; 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:
```

```
;;
Verbs = 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
   / OEM-verb

;;
;; SM CLP command options:
;;
;; General option grammar rules:
;;    option = "-" optName [optArg]
;;    options = option *(tsep option)
;;
Options = all-option
   / destination-option
   / display-option
   / examine-option
   / force-option
   / help-option
   / keep-option
   / level-option
   / output-option
   / source-option
   / version-option
   / wait-option
   / OEM-options

;;
;; SM CLP command properties:
;;
property = propertyName
propertyArrayName = propertyName "[" 1*DIGIT "]"
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-assignValue =/ propertyArrayName "=" propertyValue
property-selectValue = (propertyName / propertyArrayName) "==" 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
destination-option = tsep destination-optionName tsep (<URI> / <UFiP>)
display-option = tsep display-optionName tsep display-optionArgs
    *("," display-  optionArgs)
display-optionPropArgProperty = (propertyName)
display-optionArgs = *( "all" / "verbs" / display-optionAssocArg /
   display-optionTargetsArg / display-optionPropArg )
display-optionAssocArg = "associations" ["=" (className /
   ("("(className *("," className))")")) ]
display-optionTargetsArg = "targets" ["=" (<UFcT> /
   ("("<UFcT> *("," <UFcT>)")")) ]
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" / "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) ")" ))


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:
;;
all-legal-targets = target-Assoc / target-Instance / target-UFsT
target-Instance = tsep targetPath
   / tsep OEM-target
   / tsep "SESSION"
target-UfsT = tsep [addrTermSeparator] [addrTerm *(addrTermSeparator addrTerm)
   addrTermSeparator] UFsT
UFsT = <UFcT> splat ; UFcT*
target-Assoc = target-AssocSingleInstance / target-AssocMultiInstance
target-AssocSingleInstance = tsep targetPath "=>" className "=>" targetPath
target-AssocMultiInstance = tsep targetPath "=>" className
targetPath = [addrTermSeparator] [addrTerm *(addrTermSeparator addrTerm)]
addrTerm = (dot / dotdot / <UFiT>)

;;
;; Top-level command line production
;;
clp-command-line = clp-verb-forms / oem-cmd

;;
;; The CLP command formations
;;
clp-verb-forms   = cd-cmd / create-cmd / delete-cmd / dump-cmd / exit-cmd
clp-verb-forms   =/ help-cmd /    load-cmd / reset-cmd / set-cmd / show-cmd
clp-verb-forms   =/ start-cmd / stop-cmd / version-cmd
;;
;; Per-command formations:
;;
cd-cmd = cd-verb [cd-options] [target-Instance] eol
cd-options   = *(wait-option / common-options)

create-cmd = create-verb [create-options] (target-Instance / target-UFsT)
   properties-assignValues eol
create-cmd =/ create-verb [create-options] source-option [create-options]
   (target-Instance / target-UFsT) eol
create-cmd =/ create-verb (help-option / version-option) eol
create-options = *(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-cmd =/ dump-verb (help-option / version-option) 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] [1*VCHAR *( tsep 1*VCHAR)] eol
help-options = *(wait-option / common-options)

load-cmd = load-verb [load-options] source-option [load-options] [target-Instance]
eol
load-cmd =/ load-verb (help-option / version-option) eol
load-options = *(force-option / wait-option / common-options)

reset-cmd = reset-verb [reset-options] [target-Instance] eol
reset-options = *(force-option / wait-option / common-options)

set-cmd = set-verb [set-options] [target-Instance / target-AssocSingleInstance]
   properties-assignValues eol
set-options = *(force-option / wait-option / common-options)

show-cmd  = show-verb [level-option] [show-options] [level-option] [target-
Instance]    *(propertyName / property-selectValue) eol
show-cmd =/ show-verb [show-options] [target-UFsT / target-Assoc] *(propertyName /
   property-selectValue) eol
show-options = *(all-option / display-option / force-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-verb        = "OEM"<vendor><vendor-specified verb name>
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"]
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"]
source-optionName      = "-source"
version-optionName     = "-v" ["ersion"]
wait-optionName        = "-w" ["ait"]
```

Annex B
(informative)

W3C Universal Resource Identifiers (URI)

URIs are expected to be used as values for some keyword=value pairs, option arguments, and option argument values. 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 ensure that the schema name included in the URI is valid for the given implementation. The CLP itself does not require any specific schema or enforce any specific URIs, but they are expected to adhere to RFC2396, RFC2718, and RFC2717.

Annex C
(informative)

W3C Extensible Markup Language (XML)

The CLP supports generating XML output data (*Extensible Markup Language [XML] 1.0, 3rd edition*), as well as keyword 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 the need for Clients to import data obtained through the CLP into other applications. For more information on the XML output mode, see 5.2.4.3.3.

Annex D
(informative)

POSIX Utility Conventions

The POSIX Utility Conventions (IEEE Std. 1003.1) were considered when defining the CLP syntax. The CLP syntax adheres to as many of the POSIX Utility Guidelines as are feasible, but it 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 term 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, because it is desirable 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 lowercase. 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 that is accustomed to one-letter option names, as established in Guideline 3, 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 are preceded by the ‘-‘ (hyphen) 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 adheres to Guideline 9: the command line form is 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 it 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 that require assignment, and just keywords for operands that do not. Because these are often CIM Schema properties, they are not order dependent. Therefore, the positions of operands do not matter. This is true regardless of the 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.

Annex E
(informative)

Conventions

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

Functions or behaviors are defined to be "implicit" if those functions or behaviors 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 or behaviors are assumed to be in effect unless overridden or specified by the user through a command or command option.

Annex F
(informative)

Notation

Regular Expression (regex) and Augmented Backus-Naur Form (ABNF) are used in this document to describe various aspects of the SM CLP specification. A complete SM CLP grammar in ABNF is in Annex A.

For readability, this specification documents all verb, option, target, and property names in lowercase.

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

The following conventions are used to indicate specification elements:

`courier new`	Used to indicate literal characters in the syntax expression and in examples.
`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 provided for informative purposes and are shown using Courier New font. When the text provides an example of a CLP command and response, the CLP Command Line is emphasized using bold text. The command output is shown in regular text font.

Examples are for information only. When an example contradicts specification text elsewhere in the document, the specification text is the authority. Examples are shown `in this font and format`. Each example consists of a description of the example, 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 5.1.10. When examples do not include the `output` option with a `format` argument, it is assumed that session default format is that of the example.

Annex G
(informative)

Change History

Version 1.02	2007-02-19	Formatted to conform to *ISO/IEC Directives, Part 2: Rules for the structure and drafting of International Standards*

Annex H
(informative)

Acknowledgements

The authors wish to acknowledge the following people.

Editors:

- Aaron Merkin, IBM
- Perry Vincent, Intel

Contributors:

- Bob Blair, Newisys
- Greg Dake, IBM
- Jon Hass, Dell
- Jeff Hilland, HP
- Steffen Hulegaard, OSA Technologies
- Arvind Kumar, Intel
- Jeff Lynch, IBM
- Christina Shaw, HP
- Enoch Suen, Dell
- Michael Tehranian, Sun

Annex I
(informative)

Bibliography

IEEE Std 1003.1, "POSIX Utility Conventions", *The Open Group Base Specifications Issue 6*, 2004 Edition

World Wide Web Consortium (W3C), *Extensible Markup Language (XML) 1.0 (Third Edition)*, February 2004

IETF, RFC4251, *Secure Shell (SSH) Protocol Architecture*, January 2006

IETF, RFC4253, *Secure Shell (SSH) Transport Layer Protocol*, January 2006

IETF, RFC4252, *Secure Shell (SSH) Authentication Protocol*, January 2006

IETF, RFC4254, *Secure Shell (SSH) Connection Protocol*, January 2006

IETF, RFC4250, *Secure Shell (SSH) Protocol Assigned Numbers*, January 2006

IETF, RFC0854, *Telnet Protocol Specification*, May 1983

IETF, RFC0855, *Telnet Option Specifications*, May 1983

DMTF, DSP4004, *DMTF Release Process*, version 1.6.0, January 2007

OMG, Unified Modeling Language (UML) from the Open Management Group (OMG)

DMTF, DSP2001, *Systems Management Architecture for Server Hardware (SMASH) Command Line Protocol (CLP) Architecture White Paper*, version 1.0.1, October 2006

IETF, RFC2718, *Guidelines for new URL Schemes*, November 1999

IETF, RFC2717, *Registration Procedures for URL Scheme Names*, November 1999

DMTF, DSP0218, *Server Management (SM) Command Line Protocol (CLP) Discovery Using the Service Location Protocol (SLP)*, version 1.0, 2005

IETF, RFC2821, *Simple Mail Transfer Protocol*, April 2001

DMTF, DSP0207, *Web-Based Enterprise Management (WBEM) Universal Resource Identifier (URI) Mapping Specification*, version 1.0, January 2006

DMTF, DSP0216, *Command Line Protocol (CLP)-to-Common Information Model (CIM) Mapping Specification*, version 1.0, 2005

DMTF, DSP0217, *Systems Management Architecture for Server Hardware (SMASH) Implementation Requirements*, version 1.0, 2005

IETF, RFC3629, *UTF-8, a transformation format of ISO 10646*, 2003