



1
2
3
4

Document Number: DSP0216

Date: 2009-4-23

Version: 1.0.0

5 **SM CLP-to-CIM Common Mapping Specification**

6 **Document Type: Specification**
7 **Document Status: DMTF Standard**
8 **Document Language: E**
9

10 Copyright notice

11 Copyright © 2006, 2009 Distributed Management Task Force, Inc. (DMTF). All rights reserved.

12 DMTF is a not-for-profit association of industry members dedicated to promoting enterprise and systems
13 management and interoperability. Members and non-members may reproduce DMTF specifications and
14 documents, provided that correct attribution is given. As DMTF specifications may be revised from time to
15 time, the particular version and release date should always be noted.

16 Implementation of certain elements of this standard or proposed standard may be subject to third party
17 patent rights, including provisional patent rights (herein "patent rights"). DMTF makes no representations
18 to users of the standard as to the existence of such rights, and is not responsible to recognize, disclose,
19 or identify any or all such third party patent right, owners or claimants, nor for any incomplete or
20 inaccurate identification or disclosure of such rights, owners or claimants. DMTF shall have no liability to
21 any party, in any manner or circumstance, under any legal theory whatsoever, for failure to recognize,
22 disclose, or identify any such third party patent rights, or for such party's reliance on the standard or
23 incorporation thereof in its product, protocols or testing procedures. DMTF shall have no liability to any
24 party implementing such standard, whether such implementation is foreseeable or not, nor to any patent
25 owner or claimant, and shall have no liability or responsibility for costs or losses incurred if a standard is
26 withdrawn or modified after publication, and shall be indemnified and held harmless by any party
27 implementing the standard from any and all claims of infringement by a patent owner for such
28 implementations.

29 For information about patents held by third-parties which have notified the DMTF that, in their opinion,
30 such patent may relate to or impact implementations of DMTF standards, visit
31 <http://www.dmtf.org/about/policies/disclosures.php>.

32

33

34

CONTENTS

36	1	Scope	7
37	2	Normative References.....	7
38	2.1	Approved References	7
39	2.2	Other References.....	8
40	3	Terms and Definitions	8
41	4	Symbols and Abbreviated Terms	9
42	5	Overview	10
43	5.1	Evaluating and Applying an SM CLP Command.....	10
44	5.2	SM CLP-to-CIM Command Mapping.....	10
45	5.3	Addressing Requirements	11
46	5.4	Determining Requirements for Supporting Functionality.....	11
47	5.4.1	Relative Prioritization of Profile and Mapping Requirements	11
48	5.4.2	Overlapping Mapping Specifications	11
49	6	SM CLP-to-CIM Property Mapping	11
50	6.1	CIM MOF Property Data Type-Based Value Format.....	11
51	6.2	SM CLP Data Type Modifier-Based Value Format.....	12
52	6.3	CLP-to-CIM Property Value String Format Specifications.....	12
53	6.3.1	ABNF Productions for Property Value String Formats	14
54	6.3.2	Value and ValueMap Qualified Properties.....	14
55	7	Common Mapping Components	14
56	7.1	Common Mapping Functions.....	14
57	7.1.1	smAddError.....	14
58	7.1.2	smCommandCompleted.....	15
59	7.1.3	smCommandExecutionFailed.....	15
60	7.1.4	smConvertToDateTime.....	16
61	7.1.5	smGetObjectPath.....	17
62	7.1.6	smGetSession.....	17
63	7.1.7	smMakeCommandStatus	17
64	7.1.8	smMessage.....	18
65	7.1.9	smNewInstance	18
66	7.1.10	smSortInstancePaths.....	19
67	7.2	SM CLP-to-CIM Command Mapping Functions for Common Object Classes.....	19
68	7.2.1	smCreateInstance.....	19
69	7.2.2	smDeleteInstance.....	20
70	7.2.3	smDisplayInstance.....	20
71	7.2.4	smProcessOpError	20
72	7.2.5	smRequestStateChange.....	21
73	7.2.6	smReset.....	25
74	7.2.7	smResetRSC	26
75	7.2.8	smSetInstance	26
76	7.2.9	smShowAssociationInstances—One Reference	27
77	7.2.10	smShowInstance.....	29
78	7.2.11	smShowInstancePseudoProperties.....	29
79	7.2.12	smShowInstances.....	29
80	7.2.13	smShowInstancesByInstancePaths.....	30
81	7.2.14	smShowInstancesPseudoProperties	31
82	7.2.15	smStartRSC	31
83	7.2.16	smStopRSC	31
84	7.3	SM CLP-to-CIM Command Mapping Functions for Intrinsic Operations.....	31
85	7.3.1	smOpAssociators.....	31
86	7.3.2	smOpAssociatorNames	32
87	7.3.3	smOpReferences.....	32

88 7.3.4 smOpReferenceNames 33

89 7.3.5 smOpCreateInstance 34

90 7.3.6 smOpDeleteInstance 34

91 7.3.7 smOpEnumerateInstances 34

92 7.3.8 smOpEnumerateInstanceNames..... 34

93 7.3.9 smOpGetInstance 35

94 7.3.10 smOpInvokeMethod..... 35

95 7.3.11 smOpModifyInstance 35

96 7.4 SM CLP-to-CIM Common Messages 36

97 8 SM CLP-to-CIM Common Verb Mappings 37

98 8.1.1 cd 37

99 8.1.2 exit 38

100 8.1.3 help 38

101 8.1.4 version 39

102 ANNEX A (informative) Conventions 40

103 A.1 Notation..... 40

104 A.2 Pseudo-code..... 40

105 ANNEX B (informative) Per-Profile Mappings..... 43

106 B.1 Preconditions and Assumptions for Per-Profile SM CLP-to-CIM Command

107 Mapping Behaviors 43

108 ANNEX C (informative) Change Log..... 45

109 Bibliography 46

110

111 **Tables**

112 Table 1 – CLP-to-CIM Property Value String Formats 12

113 Table 2 – Common SM CLP Messages..... 36

114 Table A.1 – ERROR Data Type Fields 41

115 Table A.2 – CommandStatus Data Type Fields..... 41

116 Table A.3 – Message Data Type Fields..... 42

117

118

Foreword

119 The *SM CLP-to-CIM Common Mapping Specification* (DSP0216) was prepared by the Server
120 Management Working Group of the DMTF.

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

123 **Acknowledgements**

124 The authors wish to acknowledge the following people.

125 Contributors:

- 126 • Aaron Merkin – IBM
- 127 • Khachatur Papanyan – Dell
- 128 • Perry Vincent – Intel

129 Participants from the DMTF Server Management Working Group:

- 130 • Jon Hass – Dell
- 131 • Enoch Suen – Dell
- 132 • Jeff Hilland – HP
- 133 • Christina Shaw – HP
- 134 • John Leung – Intel

135

136

Introduction

137 The *SM CLP-to-CIM Common Mapping Specification* describes the common requirements for mapping
138 SM CLP commands, command options, command option argument values, and command target
139 properties to elements of the Common Information Model (CIM). This specification defines the basis for
140 implementations' conformance to the Command Line Protocol specifications. Mapping requirements
141 specific to a Management Profile are defined in a mapping specification for each Management Profile.
142 The *SM CLP-to-CIM Common Mapping Specification*, combined with per-profile command mapping
143 specifications, provides the detail that developers of CLP Services, CIM Servers, and Class Providers
144 need to build compliant implementations of the Server Management Command Line Protocol (SM CLP).

145

SM CLP-to-CIM Common Mapping Specification

146 1 Scope

147 The *SM CLP-to-CIM Common Mapping Specification* describes the common requirements for mapping
148 commands, command options, command option argument values, and command target properties to
149 elements of the Common Information Model (CIM). This specification defines the basis for
150 implementations' conformance to the Command Line Protocol specifications.

151 This document assumes that the reader is familiar with the information provided in the following
152 resources:

- 153 • [CIM Infrastructure Specification](#)
- 154 • [CIM Schema](#)
- 155 • [Systems Management Architecture for Server Hardware \(SMASH\) Command Line Protocol
156 \(CLP\) Architecture White Paper](#)
- 157 • [Server Management Managed Element Addressing Specification](#)
- 158 • [Server Management Command Line Protocol Specification](#)

159 2 Normative References

160 The following referenced documents are indispensable for the application of this document. For dated
161 references, only the edition cited applies. For undated references, the latest edition of the referenced
162 document (including any amendments) applies.

163 2.1 Approved References

164 [CIM Schema](#), 2.14

165 DMTF DSP0200, *CIM Operations over HTTP 1.2.0*,
166 http://www.dmtf.org/standards/published_documents/DSP200.pdf

167 DMTF DSP0004, *CIM Infrastructure Specification 2.3.0*,
168 http://www.dmtf.org/standards/published_documents/DSP0004V2.3_final.pdf

169 DMTF DSP0214, *Server Management Command Line Protocol Specification 1.0.0*,
170 http://www.dmtf.org/standards/published_documents/DSP0214.pdf

171 DMTF DSP0215, *Server Management Managed Element Addressing Specification 1.0.0*,
172 http://www.dmtf.org/standards/published_documents/DSP0215_1.0.0.pdf

173 DMTF DSP0201, *Specification for the Representation of CIM in XML 2.2.0*,
174 http://www.dmtf.org/standards/published_documents/DSP201.pdf

175 SNIA, *Storage Management Initiative Specification (SMI-S) 1.1.0*,
176 http://www.snia.org/tech_activities/standards/curr_standards/smi

177 2.2 Other References

- 178 ISO/IEC Directives, Part 2, *Rules for the structure and drafting of International Standards*,
 179 <http://isotc.iso.org/livelink/livelink.exe?func=ll&objId=4230456&objAction=browse&sort=subtype>
- 180 *Unified Modeling Language (UML) from the Open Management Group (OMG)*, <http://www.uml.org>
- 181 IETF RFC 2396, *Uniform Resource Identifiers (URI): Generic Syntax*, August 1998,
 182 <http://www.ietf.org/rfc/rfc2396.txt>
- 183 IETF RFC 2718, *Guidelines for new URL Schemes*, November 1999, <http://www.ietf.org/rfc/rfc2718.txt>
- 184 IETF RFC 2717, *Registration Procedures for URL Scheme Names*, November 1999,
 185 <http://www.ietf.org/rfc/rfc2717.txt>
- 186 World Wide Web Consortium, *Extensible Markup Language (XML) 1.0 (Third Edition)*, February 2004,
 187 <http://www.w3.org/tr/rec-xml>

188 3 Terms and Definitions

189 For the purposes of this document, the following terms and definitions apply. For the purposes of this
 190 document, the terms and definitions given in [DSP0214](#) and [DSP0201](#) also apply.

191 3.1

192 **can**

193 used for statements of possibility and capability, whether material, physical, or causal

194 3.2

195 **cannot**

196 used for statements of possibility and capability, whether material, physical, or causal

197 3.3

198 **conditional**

199 indicates requirements to be followed strictly in order to conform to the document when the specified
 200 conditions are met

201 3.4

202 **mandatory**

203 indicates requirements to be followed strictly in order to conform to the document and from which no
 204 deviation is permitted

205 3.5

206 **may**

207 indicates a course of action permissible within the limits of the document

208 3.6

209 **need not**

210 indicates a course of action permissible within the limits of the document

211 3.7

212 **optional**

213 indicates a course of action permissible within the limits of the document

- 214 **3.8**
215 **referencing profile**
216 indicates a profile that owns the definition of this class and can include a reference to this profile in its
217 “Related Profiles” table
- 218 **3.9**
219 **shall**
220 indicates requirements to be followed strictly in order to conform to the document and from which no
221 deviation is permitted
- 222 **3.10**
223 **shall not**
224 indicates requirements to be followed in order to conform to the document and from which no deviation is
225 permitted
- 226 **3.11**
227 **should**
228 indicates that among several possibilities, one is recommended as particularly suitable, without
229 mentioning or excluding others, or that a certain course of action is preferred but not necessarily required
- 230 **3.12**
231 **should not**
232 indicates that a certain possibility or course of action is deprecated but not prohibited
- 233 **3.13**
234 **unspecified**
235 indicates that this profile does not define any constraints for the referenced CIM element or operation

236 **4 Symbols and Abbreviated Terms**

237 The following symbols and abbreviations are used in this document.

- 238 **4.1**
239 **ABNF**
240 Augmented Backus-Naur Form
- 241 **4.2**
242 **CIM**
243 Common Information Model
- 244 **4.3**
245 **CIM Server**
246 Common Information Model Server
- 247 **4.4**
248 **CLP**
249 Command Line Protocol
- 250 **4.5**
251 **MAP**
252 Manageability Access Point

253 **4.6**
254 **ME**
255 Managed Element

256 **4.7**
257 **MOF**
258 Managed Object File

259 **4.8**
260 **UFcT**
261 User-Friendly class Tag

262 **4.9**
263 **UFiT**
264 User-Friendly instance Tag

265 **4.10**
266 **UFsT**
267 User-Friendly selection Tag

268 **5 Overview**

269 The *SM CLP-to-CIM Common Mapping Specification* describes the mapping of Server Management
270 Command Line Protocol (SM CLP) commands, command options, command option argument values,
271 and command target properties to elements of the Common Information Model (CIM). This specification
272 defines the basis for an implementation's conformance to the SM CLP. The command mapping
273 specification for each profile defines additional requirements to be met when the profile is supported by
274 the instrumentation.

275 **5.1 Evaluating and Applying an SM CLP Command**

276 The steps for executing an SM CLP command are as follows:

- 277 1) Command line is validated against command line grammar.
- 278 2) Command line is validated against requirements in [DSP0214](#).
- 279 3) A job is created to track command execution.
- 280 4) The Resultant Target is resolved to an Object Path.
- 281 5) The SM CLP commands are mapped to the CIM instrumentation.
- 282 6) The job is completed.
- 283 7) The CLP output is produced.

284 The scope of this specification is limited to steps 5 and 6.

285 **5.2 SM CLP-to-CIM Command Mapping**

286 An SM CLP-to-CIM command mapping is a functional definition of how to interpret an SM CLP command
287 and apply it against underlying CIM instrumentation. This mapping includes the validation of requirements
288 specified in the following sections as well as the requirements specified in the per-profile command
289 mapping specification in which the targets are defined. Each mapping involves the conversion from
290 SM CLP syntax and semantics to CIM elements and operations.

291 **5.3 Addressing Requirements**

292 The SM CLP command target address term identifies the CIM instance to which the command is applied.
293 The SM CLP-to-CIM common mapping is not dependent on particular target address term syntax. Each
294 mapping assumes the Resultant Target of a command has been resolved to an Object Path identifying a
295 single CIM instance.

296 **5.4 Determining Requirements for Supporting Functionality**

297 The following sections describe how requirements are prioritized for profiles and overlapping mapping
298 specifications.

299 **5.4.1 Relative Prioritization of Profile and Mapping Requirements**

300 Management profiles specify requirements for support of CIM elements and operations. The requirement
301 to support an element can be mandatory, optional, or conditional. Command mapping specifications state
302 the requirement for support of each CLP command form for a CIM element. The implementation of these
303 command forms may depend on CIM elements that are not mandatory in the underlying instrumentation.
304 Requirements for support of a CLP command form are conditional; they depend on support for the CIM
305 elements that are required to implement the mapping of the CLP command form. In other words, a
306 requirement to support a particular CLP command form does not create a requirement to support the
307 underlying CIM elements. Rather, if the underlying CIM elements are supported, support for the CLP
308 command form would be required.

309 **5.4.2 Overlapping Mapping Specifications**

310 More than one profile and mapping specification pair may define requirements for supporting a CLP
311 command for a CIM class. For a given CIM instance, the requirements for supporting a CLP command
312 shall be the union of requirements specified by the profiles and mapping specifications supported by an
313 implementation.

314 **6 SM CLP-to-CIM Property Mapping**

315 This section specifies how implementations are to use string tokens to reference properties and property
316 values of instances of the target instance's CIM class.

317 The string name of CIM class properties defined in the MOF for the corresponding class shall be
318 recognized as property names for a SM CLP command target. Implementations shall allow the use of
319 upper or lower case characters for property names.

320 Implementations shall use the property name in structured output as a SM CLP output keyword or a
321 CLPXML tag as specified in the [DSP0214](#).

322 **6.1 CIM MOF Property Data Type-Based Value Format**

323 The [DSP0004](#) contains specifications for character string formats that are used when initializing property
324 values in a CIM MOF.

325 When a property value is specified for a property name that matches the corresponding property in the
326 CIM MOF for the target class, the character string format for the property value shall match the format
327 specified in column 3 in Table 1 when the property has the data type specified in column 1 in Table 1.

328 **6.2 SM CLP Data Type Modifier-Based Value Format**

329 This specification defines a set of string tokens that select a specific value string format for the property
 330 that it modifies. The data type modifier type string is appended to a property name, separated only by a
 331 single sharp or pound sign character (#).

332 When a property value is specified for a property name that matches the corresponding property in the
 333 CIM MOF for the class and the property is modified using the form “#<modifier>”, the implementation shall
 334 accept the character string format for the value that corresponds to the property’s data type and selected
 335 modifier as listed in Table 1.

336 **6.3 CLP-to-CIM Property Value String Format Specifications**

337 Table 1 specifies the character string format to be used to express and interpret values for target
 338 properties based on the data type of the property as defined by the CIM class MOF, the corresponding
 339 specifications in the [DSP0004](#) Version 2.3, Section 2.2, and SM CLP data type modifiers.

340 **Table 1 – CLP-to-CIM Property Value String Formats**

CIM MOF Intrinsic Data Types SM-CLP Data Type Modifiers	CIM Infrastructure Specification 2.3 Interpretation	CIM Infrastructure Specification 2.3 Grammar Production Token or SM-CLP Modifier Specification	Examples
uint8	Unsigned 8-bit integer	DSP0004 , Appendix A, token = integerValue	01001101b 77 0x4d O75
sint8	Signed 8-bit integer	DSP0004 , Appendix A, token = integerValue	-01001101b +77 -0x4d +O75
uint16	Unsigned 16-bit integer	See section 6.3.2.	01001101b 77 0x4d O75
sint16	Signed 16-bit integer	DSP0004 , Appendix A, token = integerValue	-01001101b +77 -0x4d +O75
uint32	Unsigned 32-bit integer	DSP0004 , Appendix A, token = integerValue	01001101b 77 0x4d O75
sint32	Signed 32-bit integer	DSP0004 , Appendix A, token = integerValue	-01001101b +77 -0x4d

CIM MOF Intrinsic Data Types SM-CLP Data Type Modifiers	CIM Infrastructure Specification 2.3 Interpretation	CIM Infrastructure Specification 2.3 Grammar Production Token or SM-CLP Modifier Specification	Examples
			+O75
uint64	Unsigned 64-bit integer	DSP0004 , Appendix A, token = integerValue	01001101b 77 0x4d O75
sint64	Signed 64-bit integer	DSP0004 , Appendix A, token = integerValue	-01001101b +77 -0x4d +O75
string	UCS-2 string	DSP0004 , Appendix A, token = stringValue	foo "foo bar"
boolean	Boolean	DSP0004 , Appendix A, token = booleanValue	true TRUE False FAISe
real32	IEEE 4-byte floating-point	DSP0004 , Appendix A, token = realValue	-10.234e2 -10.234e-2 1.0234e0
real64	IEEE 8-byte floating-point	DSP0004 , Appendix A, token = realValue	-10.234e2 -10.234e-2 1.0234e0
datetime	A string containing a date-time as specified in DSP0004	Section 6.3.1, datetime-absolute	19980525133015.0000000-300
#time	User-friendly string specifying absolute date time	Section 6.3.1, datetime-friendly	12:34:00 1998-05-25.12:34:00 1998-05-25.12:34:00-300
#interval	User-friendly string specifying interval	Section 6.3.1, datetime-interval	12:34:00 0002-11-25.12:34:00
<classname> ref	Strongly-typed CIM classname reference	DSP0214 , Section 4.14, token = target-Instance	CIM_System CIM_SettingData
char16	16-bit UCS-2 character	DSP0004 , Appendix A, token = charValue	a b ' * <

341 **6.3.1 ABNF Productions for Property Value String Formats**

342 The following are ABNF productions for property value string formats:

343 `datetime-absolute = 14DIGIT "." 6DIGIT ("+" / "-") 3DIGIT`

344 `datetime-friendly = datetime-interval [("+" / "-") 3DIGIT]`

345 `datetime-interval = [4DIGIT "-" 2DIGIT "-" 2DIGIT ":"] 2DIGIT ":" 2DIGIT ":" 2DIGIT`

346 **6.3.2 Value and ValueMap Qualified Properties**

347 When an integer property does not have the Value and ValueMap qualifiers, the format of the property
 348 shall be as specified by the `stringValue` token in [DSP0004](#), Appendix A. When an integer property has
 349 the Value and ValueMap qualifiers, the string value contained in the Value qualifier shall be interpreted as
 350 identifying the corresponding integer value in the ValueMap qualifier. The ValueMap value may be
 351 accepted when it is provided in the format specified by the `stringValue` token in [DSP0004](#),
 352 Appendix A.

353 Examples:

354 `RequestedState="Enabled"`

355 `RequestedState="2"`

356 **7 Common Mapping Components**

357 The *SM CLP-to-CIM Common Mapping Specification* defines a set of common patterns or algorithms in
 358 SM CLP implementations as reusable functions.

359 Section 7.1 describes common functions for accomplishing frequently performed sequences when
 360 implementing basic SM CLP processing and output generation, such as returning error data.

361 Common functions for object classes that are common across per-profile command mappings are
 362 expressed in pseudo-code syntax in section 7.2.

363 Common functions used to accomplish standard CIM operations are defined in section 7.3.

364 Common SM CLP messages are documented in section 7.4.

365 **7.1 Common Mapping Functions**

366 This section defines functions that are used by command mapping behavior pseudo-code when
 367 documenting functionality that is reusable for SM CLP functionality, such as producing command output
 368 elements.

369 **7.1.1 smAddError**

370 This function is used to associate a `CIM_Error` instance with the instance of `CIM_ConcreteJob` that
 371 represents operation execution. No associations are defined that can be used to associate the `CIM_Error`
 372 instance. Nor is there an extrinsic method on `CIM_ConcreteJob` to set the reference. This function is
 373 defined to provide a clear marker in the pseudo code for the `CIM_Error` instance that a mapping intends
 374 to define as the top-level `CIM_Error` instance for the operation.

```
375     smAddError($job, $error)    {
376         <unspecified magic>
377     }
```

378 \$job is the CIM_ConcreteJob instance that represents the overall operation.

379 \$error is the CIM_Error instance that is the top level error for the operation.

380 7.1.2 smCommandCompleted

381 This function forms the SM CLP Command status elements when a command completes successfully.

```
382     Sub smCommandCompleted($job)      {
383         <CommandStatus>.status = 0
384         <CommandStatus>.status_tag = "COMMAND COMPETED SUCCESSFULLY"
385         <CommandStatus>.job_id = $job.InstanceID
386     }
```

387 7.1.3 smCommandExecutionFailed

388 The function is used to map CIM_Error Instances to SM CLP Command Status Elements.

389 For more information on how processing errors are handled, see section 3.1.6 ("Command Processing")
390 in the [DSP0214](#).

```
391 //Map an array of CIM_Error instances to SM CLP Command Status data elements
392 sub smCommandExecutionFailed($job, $errors[])      {
393     $Error = $errors[0];
394     <CommandStatus>.status = 3
395     <CommandStatus>.status_tag = COMMAND EXECUTION FAILED
396     //DSP1005 constrains the property value to contain the Job ID
397     <CommandStatus>.job_id = $job.ElementName
398
399     <CommandStatus>.errtype $Error.ErrorType;
400     if (1 != $Error.ErrorType)  {
401         <CommandStatus>.errtype_desc = Values[$Error.ErrorType];
402     }
403     else  {
404         <CommandStatus>.errtype_desc = $Error.OtherErrorType;
405     }
406     <CommandStatus>.cimstat = $Error.CIMStatusCode;
407     if (1 != $Error.CIMStatusCode)  {
408         <CommandStatus>.cimstat_desc = Values[$Error.CIMStatusCode];
409     }
410     else  {
411         <CommandStatus>.cimstat_desc = $Error.CIMStatusCodeDescription;
412     }
413
414     <CommandStatus>.severity = $Error.PerceivedSeverity;
415     <CommandStatus>.severity_desc = Values[$Error.PerceivedSeverity];
416     <CommandStatus>.probcause = $Error.ProbableCause;
417     if (1 != $Error. ProbableCause)  {
418         <CommandStatus>.probcause_desc = Values[$Error. ProbableCause];
419     }
420     else  {
421         <CommandStatus>.probcause_desc = $Error. ProbableCauseDescription;
422     }
423     <CommandStatus>.recmdaction = $Error.RecommendedActions;
424     <CommandStatus>.errsource = $Error.ErrorSource;
425     <CommandStatus>.errsourceform = $Error.ErrorSourceFormat;
426     if(1 != $Error.ErrorSourceFormat) {
427         <CommandStatus>.errsourceform_desc = Values[$Error.ErrorSourceFormat];
428     }
429     else  {
430         <CommandStatus>.errsourceform_desc = $Error.OtherErrorSourceFormat;
431     }
432 }
```

```

432     &smMessage(<CommandStatus>, $Error);
433     #i = 1;
434     for ); #i< $errors.Length; #i++) {
435         smMessage $errors[#i];
436     }
437
438 }//end smErrors

```

439 7.1.4 smConvertToDateTime

440 This function is used to convert Time, Time#Time, and Time#Interval values to DateTime values to
 441 provide consistent errors for poorly formed values of Time, Time#Time, Time#Interval.

442 A value of "Time" for #expectedFormat indicates that #propertyValueString is expected to be formatted in
 443 a manner that is compliant with the production datetime-absolute specified in section 6.3.1.

444 A value of "Time#Time" for #expectedFormat indicates that #propertyValueString is expected to be
 445 formatted in a manner that is compliant with the production datetime-friendly specified in section 6.3.1.

446 A value of "Time#Interval" indicates that #propertyValueString is expected to be formatted in a manner
 447 that is compliant with the production datetime-interval specified in section 6.3.1.

```

448 sub #requestedtime smConvertToDatetime(#propertyValueString, #expectedFormat)
449 {
450     if ("Time#Time" == #expectedFormat) {
451         //validate against section 6.3.1 datetime-friendly,
452         if (#valid) {
453             //return datetime value
454         }
455     }
456     else {
457         //validate against Section 6.3.1 datetime-interval
458         if (#valid) {
459             //convert to datetime interval value and return
460         }
461     }
462
463     //value is returned if valid, so if we're here the value must be invalid and we
464     need to return the error
465
466     $OperationError = smNewInstance("CIM_Error");
467     //CIM_ERR_INVALID_PARAMETER
468     $OperationError.CIMStatusCode = 4;
469     //Other
470     $OperationError.ErrorType = 1;
471     //Low
472     $OperationError.PerceivedSeverity = 2;
473     $OperationError.OwningEntity = DMTF:SMCLP;
474     $OperationError.MessageID = 0x00000011;
475     $OperationError.Message = "A property value is incorrectly formatted.";
476     &smAddError($job, $OperationError);
477     &smMakeCommandStatus($job);
478     smEnd;
479 }

```


480 7.1.5 smGetObjectPath

481 This function is used to convert a Server Management Managed Element address into an object path of a
482 CIM instance. (The mechanism by which an SM ME address is converted to an object path is outside the
483 scope of this specification.)

```
484     sub $instance-> smGetObjectPath(#address)    {
485         if (<successfully mapped>) {
486             return $instance->
487         }
488         else {
489             $OperationError = smNewInstance("CIM_Error");
490             //CIM_ERR_NOT_FOUND
491             $OperationError.CIMStatusCode = 6;
492             //Other
493             $OperationError.ErrorType = 1;
494             //Low
495             $OperationError.PerceivedSeverity = 2;
496             &smAddError($job, $OperationError);
497             &smMakeCommandStatus($job);
498             smEnd;
499         }

```

500 #address is a string that contains an absolute path defined according to the [DSP0215](#).

501 7.1.6 smGetSession

502 This function is used to fetch the instance of CIM_CLPProtocolEndpoint that represents the session that
503 originated the CLP command for which the invoking mapping was applied. The mechanism by which an
504 instance of CIM_CLPProtocolEndpoint is correlated to a particular CLP session is undefined.

```
505     sub $instance smGetSession()    {
506         <unspecified magic>
507     }

```

508 An instance of CIM_CLPProtocolEndpoint is returned.

509 7.1.7 smMakeCommandStatus

510 This function creates the Command Status portion of a Command Response.

```
511     sub smMakeCommandStatus($job) {
512         //assume the job has stopped ( normally or otherwise )
513         //if the OperationalStatus is not okay, then there should be an instance of
514         CIM_Error available through
515         //GetError()
516         if (2 != $job.OperationalStatus)    {
517             %InArguments[] = { }
518             %OutArguments[] = {newArgument("Job",
519             $instanceConcreteJob.getObjectPath())}
520             #Error = smOpInvokeMethod($job,
521                 "GetError"
522                 %InArguments,
523                 %OutArguments,
524                 #returncode);
525
526             //Method invocation failed, internal processing error
527             if ( 0 != #Error.code || 0 != #returncode )    {
528                 <CommandStatus>.status = 3

```

```

529         <CommandStatus>.status_tag = COMMAND EXECUTION FAILED
530         <CommandStatus>.job_id = $job.InstanceID
531         <CommandStatus>.errtype = 4;
532         <CommandStatus>.errtype_desc = "Software Error";
533         <CommandStatus>.cimstat = 1;
534         <CommandStatus>.cimstat_desc = "CIM_ERR_FAILED";
535         <CommandStatus>.severity = 0;
536         <CommandStatus>.severity_desc = "Unknown";
537         <CommandStatus>.messages[<CommandStatus>.messages.length].message =
538         "An internal software error has occurred.";
539         <CommandStatus>.messages[<CommandStatus>.messages.length].owningentity
540         y = "DMTF:SMCLP";
541         <CommandStatus>.messages[<CommandStatus>.messages.length].message_id
542         = 0x00000009;
543         smEnd;
544     }
545     else {
546         //make command status
547         $joberror = %OutArguments["Error"];
548         &smCommandExecutionFailed($job, {$joberror});
549         smEnd;
550     } //end if have CIM_Error from GetError()
551 }
552 else {
553     //command completed successfully
554     &smCommandCompleted($job);
555     smEnd;
556 }
557 } //end smMakeCommandStatus()

```

558 7.1.8 smMessage

559 This function maps CIM_Error message data to SM CLP message elements.

```

560 //Produce SM CLP Message data element for instance of CIM_Error
561 sub smMessage(<CommandStatus>, $Error) {
562     <CommandStatus>.messages[<CommandStatus>.messages.length].message =
563     $Error.Message;
564     <CommandStatus>.messages[<CommandStatus>.messages.length].owningentity =
565     $Error.OwningEntity;
566     <CommandStatus>.messages[<CommandStatus>.messages.length].message_id =
567     $Error.MessageID;
568     #i = 0;
569     for );#i < $Error.MessageArguments.Length; #i+++ {
570         <CommandStatus>.messages[<CommandStatus>.messages.length].message_arg[#i] =
571         $Error.MessageArguments[#i];
572     }
573 }

```

574 <CommandStatus> is the Command Status instance to which messages will be added.

575 \$Error is the instance of CIM_Error that contains the message information.

576 7.1.9 smNewInstance

577 This function is used to create a new CIMInstance that is local to the implementation of a mapping.
578 Values for all properties of the instance are initially unassigned.

579 When using the smOpCreateInstance function to create an instance in the CIMOM, it is first necessary to
580 allocate the template instance in the client space. The smNewInstance function performs this step. The
581 template instance does not exist in the underlying CIMOM. To create an instance in the CIMOM that has

582 the specified property values, the implementation uses the smOpCreateInstance function, described in
 583 section 7.3.5, specifying the template instance created using the smNewInstance function as the
 584 parameter.

```
585     sub $instance smNewInstance(string #className)      {
586         <unspecified magic>
587     }
```

588 #className identifies the CIM class of which an instance will be instantiated.

589 7.1.10 smSortInstancePaths

590 This function sorts a CIMObjectPath array based on the property in the order specified and returns a
 591 sorted CIMObjectPath array.

```
592 Sub $sortedInstancePaths[] smSortInstancePaths ( $instancePaths->[], string
593 #propertyNameSortBy = NULL, [boolean] #descendingOrder = true ) {
594     <algorithm that sorts instances based on the property in the order
595     specified>
596 }
```

597 #descendingOrder identifies the order of sorted CIMInstances based on the value of a certain
 598 property. The property that is sorted by can be of different data types.

599 #descendingOrder=true represents the specified order for the following data types:

```
600     integer – decreasing numerical value
601     boolean – TRUE value first and FALSE value last
602     real – decreasing numerical value
603     string – decreasing alphanumerical value
604     datetime – from earliest date time (the most recent) to the latest date time
```

605 7.2 SM CLP-to-CIM Command Mapping Functions for Common Object Classes

606 The functions described in the following sections implement functionality that is common across multiple
 607 object classes for mappings of SM CLP verbs.

608 7.2.1 smCreateInstance

609 This function is a wrapper around the intrinsic CreateInstance method that includes mapping to SM CLP
 610 Command Status data elements.

```
611 //PRECONDITIONS
612 // 1. $target parameter contains the object to create
613 // 2. $job contains the instance of ConcreteJob modeling the CLP operation
614 sub void smCreateInstance( $target) {
615     #Error = smOpCreateInstance( $target);
616     if (0 != #Error.code)
617     {
618         &smProcessOpError (#Error);
619         //includes smEnd;
620     }
621     else {
622         //completed successfully
623         &smCommandCompleted($job);
624         smEnd;
625     }
626 }
```

627 7.2.2 smDeleteInstance

628 This function is a wrapper around the intrinsic DeleteInstance method that includes mapping to SM CLP
629 Command Status data elements.

```
630 //PRECONDITIONS
631 // 1. $target-> parameter contains the objectpath of instance to delete
632 // 2. $job contains the instance of ConcreteJob modeling the CLP operation
633 sub void smDeleteInstance( $target) {
634     #Error = smOpDeleteInstance( $target);
635     if (0 != #Error.code)
636     {
637         &smProcessOpError (#Error);
638         //includes smEnd;
639     }
640     else {
641         //completed successfully
642         &smCommandCompleted($job);
643         smEnd;
644     }
645 }
```

646 7.2.3 smDisplayInstance

647 This function displays the properties of the instance, conforming to the output format specifications
648 described in the [DSP0214](#).

649 7.2.3.1 Signature without Pseudo-Properties

650 If the #propertyNameToDisplay[] parameter is NULL, all properties of the instance are displayed. If
651 the #propertyNameToDisplay[] parameter is not NULL, then only those specified properties are
652 displayed.

```
653 Sub void smDisplayInstance ( $instance, #propertyNameToDisplay[] ) {
654     <unspecified magic>
655     //if #propertyNameToDisplay[] is not NULL, limit output to only those properties
656 }
```

657 7.2.3.2 Signature with Pseudo-Properties

658 If the #propertyNameToDisplay[] parameter is NULL, all properties of the instance are displayed. If
659 the #propertyNameToDisplay[] parameter is not NULL, then only those specified properties are
660 displayed. If #pseudoPropertiesToDisplay is non-NULL, these properties will be appended to the output.

```
661 Sub void smDisplayInstance ( $instance, #propertyNameToDisplay[],
662 #pseudoPropertiesToDisplay[] ) {
663     <unspecified magic>
664     //if #propertyNameToDisplay[] is not NULL, limit properties intrinsic to the
665     instance to only those properties listed
666 }
```

667 7.2.4 smProcessOpError

668 This function implements error processing for an intrinsic operation. The function will construct
669 appropriate command status for the execution.

```
670 //Preconditions
671 // 1. #Error represents the Error object returned from CIM operation function
672 // 2. global $job contains the CIM_ConcreteJob instance for the operation
673 sub void smProcessOpError (<Error> #Error ) {
674     if (0 != #Error.code) {
```

```

675 //method invocation failed
676     if ( (null != #Error.$error) && (null != #Error.$error[0]) )    {
677         //if the method invocation contains an embedded error
678         //use it for the Error for the overall job
679         &smAddError($job, #Error.$error[0]);
680         &smMakeCommandStatus($job);
681         smEnd;
682     }
683     else {
684         //operation failed, but no detailed error instance, need to make one up
685         //make an Error instance and associate with job for Operation
686         $OperationError = smNewInstance("CIM_Error");
687         //CIM_ERR_FAILED
688         $OperationError.CIMStatusCode = #Error.code;
689         //Software Error
690         $OperationError.ErrorType = 4;
691         //Unknown
692         $OperationError.PerceivedSeverity = 0;
693         $OperationError.OwningEntity = DMTF:SMCLP;
694         $OperationError.MessageID = 0x00000009;
695         $OperationError.Message = "An internal software error has occurred.";
696         &smAddError($job, $OperationError);
697         &smMakeCommandStatus($job);
698         smEnd;
699     }
700 } //if CIM op failed
701 }

```

702 7.2.5 smRequestStateChange

703 This function implements a generic invocation of the RequestStateChange() method on sub-classes of
704 CIM_EnabledLogicalElement. The function handles the implementation through the production of
705 Command Status.

706 Note that the Timeout parameter is not used with this version of the method.

```

707 //PRECONDITIONS
708 // 1. $target-> parameter contains the object name of the target instance
709 // 2. #requestedState parameter contains the requested state
710 // 3. global $job contains the CIM_ConcreteJob instance for the operation
711 sub void smRequestStateChange($target->, string #requestedState)    {
712     $instanceConcreteJob = smNewInstance ("CIM_ConcreteJob");
713
714     %InArguments[] = {newArgument("RequestedState", #requestedState),
715                     newArgument("TimeoutPeriod", NULL) }
716     %OutArguments[] = {newArgument("Job", $instanceConcreteJob.getObjectPath())}
717     #Error = smOpInvokeMethod ($target->,
718                               "RequestStateChange",
719                               %InArguments[],
720                               %OutArguments[],
721                               #returnStatus);
722     if (0 != #Error.code)    {
723         //method invocation failed
724         if ( (null != #Error.$error) && (null != #Error.$error[0]) )    {
725             //if the method invocation contains an embedded error
726             //use it for the Error for the overall job
727             &smAddError($job, #Error.$error[0]);
728             &smMakeCommandStatus($job);
729             smEnd;
730         }
731         else if (#Error.code == 17)    {
732             //trap for CIM_METHOD_NOT_FOUND

```

```

733 //and make nice Unsupported msg.
734 //unsupported
735 $OperationError = smNewInstance("CIM_Error");
736 //CIM_ERR_NOT_SUPPORTED
737 $OperationError.CIMStatusCode = 7;
738 //Other
739 $OperationError.ErrorType = 1;
740 //Low
741 $OperationError.PerceivedSeverity = 2;
742 $OperationError.OwningEntity = DMTF:SMCLP;
743 $OperationError.MessageID = 0x00000001;
744 $OperationError.Message = "Operation is not supported.";
745 &smAddError($job, $OperationError);
746 &smMakeCommandStatus($job);
747 smEnd;
748 }
749 else {
750 //operation failed, but no detailed error instance, need to make one up
751 //make an Error instance and associate with job for Operation
752 $OperationError = smNewInstance("CIM_Error");
753 //CIM_ERR_FAILED
754 $OperationError.CIMStatusCode = 1;
755 //Software Error
756 $OperationError.ErrorType = 4;
757 //Unknown
758 $OperationError.PerceivedSeverity = 0;
759 $OperationError.OwningEntity = DMTF:SMCLP;
760 $OperationError.MessageID = 0x00000009;
761 $OperationError.Message = "An internal software error has occurred.";
762 &smAddError($job, $OperationError);
763 &smMakeCommandStatus($job);
764 smEnd;
765 }
766 else {
767 //operation failed, but no detailed error instance, need to make one up
768 //make an Error instance and associate with job for Operation
769 $OperationError = smNewInstance("CIM_Error");
770 //CIM_ERR_FAILED
771 $OperationError.CIMStatusCode = 1;
772 //Software Error
773 $OperationError.ErrorType = 4;
774 //Unknown
775 $OperationError.PerceivedSeverity = 0;
776 $OperationError.OwningEntity = DMTF:SMCLP;
777 $OperationError.MessageID = 0x00000009;
778 $OperationError.Message = "An internal software error has occurred.";
779 &smAddError($job, $OperationError);
780 &smMakeCommandStatus($job);
781 smEnd;
782 }
783 }//if CIM op failed
784 else if (0 == #returnStatus) {
785 //completed successfully
786 &smCommandCompleted($job);
787 smEnd;
788 }
789 else if (0x4096 == #returnStatus) {
790 //job spawned, need to watch for it to finish
791 //while the jobstate is "Running"
792 while (4 == $instanceConcreteJob.JobState){<busy wait>}
793 if (2 != $job.OperationalStatus) {
794 %InArguments[] = { }
795 %OutArguments[] = {newArgument("Job", $instanceConcreteJob.getObjectPath())}
796 #Error = smOpInvokeMethod($job,

```

```

797         "GetError"
798         %InArguments,
799         %OutArguments,
800         #returncode);

801     //Method invocation failed, internal processing error
802     if ( (0 != #Error.code) || (0 != #returncode) ) {
803     //make an Error instance and associate with job for Operation
804         $OperationError = smNewInstance("CIM_Error");
805         //CIM_ERR_FAILED
806         $OperationError.CIMStatusCode = 1;
807         //Software Error
808         $OperationError.ErrorType = 4;
809         //Unknown
810         $OperationError.PerceivedSeverity = 0;
811         $OperationError.OwningEntity = DMTF:SMCLP;
812         $OperationError.MessageID = 0x00000009;
813         $OperationError.Message = "An internal software error has occurred.";
814         &smAddError($job, $OperationError);
815         &smMakeCommandStatus($job);
816         smEnd;
817     }
818     else {
819         //make command status
820         $joberror = %OutArguments["Error"];
821         &smCommandExecutionFailed($job, {$joberror});
822     } //end if have CIM_Error from GetError()
823 } //embedded job not OK
824 else {
825     //the job ran to completion (we assume)
826     &smCommandComplete($job);
827     smEnd;
828 }
829 } //if job spawned
830 else if (1 == #returnStatus) {
831     //unsupported
832     $OperationError = smNewInstance("CIM_Error");
833     //CIM_ERR_NOT_SUPPORTED
834     $OperationError.CIMStatusCode = 7;
835     //Other
836     $OperationError.ErrorType = 1;
837     //Low
838     $OperationError.PerceivedSeverity = 2;
839     $OperationError.OwningEntity = DMTF:SMCLP;
840     $OperationError.MessageID = 0x00000001;
841     $OperationError.Message = "Operation is not supported.";
842     &smAddError($job, $OperationError);
843     &smMakeCommandStatus($job);
844     smEnd;
845 }
846 else if (5 == #returnStatus) {
847     //unsupported
848     $OperationError = smNewInstance("CIM_Error");
849     //CIM_ERR_INVALID_PARAMETER
850     $OperationError.CIMStatusCode = 4;
851     //Other
852     $OperationError.ErrorType = 1;
853     //Low
854     $OperationError.PerceivedSeverity = 2;
855     $OperationError.OwningEntity = DMTF:SMCLP;
856     $OperationError.MessageID = 0x00000004;
857     $OperationError.Message = "One or more parameters specified are invalid.";
858     &smAddError($job, $OperationError);
859     &smMakeCommandStatus($job);

```

```

860     smEnd;
861 }
862 else if (6 == #returnStatus || 4099 == #returnStatus) {
863     //busy
864     $OperationError = smNewInstance("CIM_Error");
865     //CIM_ERR_FAILED
866     $OperationError.CIMStatusCode = 1;
867     //Other
868     $OperationError.ErrorType = 1;
869     //Low
870     $OperationError.PerceivedSeverity = 2;
871     $OperationError.OwningEntity = DMTF:SMCLP;
872     $OperationError.MessageID = 0x0000000A;
873     $OperationError.Message = "The target is busy and its state cannot be
874     changed.";
875     &smAddError($job, $OperationError);
876     &smMakeCommandStatus($job);
877     smEnd;
878 }
879 else if (4097 == $returnStatus) {
880     //invalid state transition
881     $OperationError = smNewInstance("CIM_Error");
882     //CIM_ERR_FAILED
883     $OperationError.CIMStatusCode = 1;
884     //Other
885     $OperationError.ErrorType = 1;
886     //Low
887     $OperationError.PerceivedSeverity = 2;
888     $OperationError.OwningEntity = DMTF:SMCLP;
889     $OperationError.MessageID = 0x0000000B;
890     $OperationError.Message = "The target cannot transition to the requested state
891     from its current state.";
892     &smAddError($job, $OperationError);
893     &smMakeCommandStatus($job);
894 }
895 else if (2 == #returnStatus || 4 == #returnStatus ||
896     3 == $returnStatus) {
897     //generic failure
898     $OperationError = smNewInstance("CIM_Error");
899     //CIM_ERR_FAILED
900     $OperationError.CIMStatusCode = 1;
901     //Other
902     $OperationError.ErrorType = 1;
903     //Low
904     $OperationError.PerceivedSeverity = 2;
905     $OperationError.OwningEntity = DMTF:SMCLP;
906     $OperationError.MessageID = 0x00000002;
907     $OperationError.Message = "Failed. No further information is available.";
908     &smAddError($job, $OperationError);
909     &smMakeCommandStatus($job);
910 }
911 else {
912     //unspecified return code, generic failure
913     $OperationError = smNewInstance("CIM_Error");
914     //CIM_ERR_FAILED
915     $OperationError.CIMStatusCode = 1;
916     //Other
917     $OperationError.ErrorType = 1;
918     //Low
919     $OperationError.PerceivedSeverity = 2;
920     $OperationError.OwningEntity = DMTF:SMCLP;
921     $OperationError.MessageID = 0x00000002;
922     $OperationError.Message = "Failed. No further information is available.";
923     &smAddError($job, $OperationError);

```



```

924         &smMakeCommandStatus($job);
925         smEnd;
926     }
927 }//end smRequestStateChange()

```

928 7.2.6 smReset

929 This function implements a generic invocation of the Reset() method on sub-classes of
930 CIM_LogicalDevice.

```

931 // 1. $target-> parameter contains the object name of the target instance
932 // 2. global $job contains the CIM_ConcreteJob instance for the operation
933 sub void smReset($target->) {
934     $instanceConcreteJob = smNewInstance ("CIM_ConcreteJob");
935     %InArguments[] = { };
936     %OutArguments[] = { };
937     #Error = smOpInvokeMethod ($target->,
938                               "Reset",
939                               %InArguments[],
940                               %OutArguments[],
941                               #returnStatus);
942     if (0 != #Error.code) {
943         //method invocation failed
944         if ( (null != #Error.$error) && (null != #Error.$error[0]) ) {
945             //if the method invocation contains an embedded error
946             //use it for the Error for the overall job
947             &smAddError($job, #Error.$error[0]);
948             &smMakeCommandStatus($job);
949             smEnd;
950         }
951         else {
952             //operation failed, but no detailed error instance, need to make one up
953             //make an Error instance and associate with job for Operation
954             $OperationError = smNewInstance("CIM_Error");
955             //CIM_ERR_FAILED
956             $OperationError.CIMStatusCode = 1;
957             //Software Error
958             $OperationError.ErrorType = 4;
959             //Unknown
960             $OperationError.PerceivedSeverity = 0;
961             $OperationError.OwningEntity = DMTF:SMCLP;
962             $OperationError.MessageID = 0x00000009;
963             $OperationError.Message = "An internal software error has occurred.";
964             &smAddError($job, $OperationError);
965             &smMakeCommandStatus($job);
966             smEnd;
967         }
968     }
969     else if (0 == #returnStatus) {
970         //completed successfully
971         &smCommandCompleted($job);
972         smEnd;
973     }
974     else if (1 == #returnStatus) {
975         //unsupported
976         $OperationError = smNewInstance("CIM_Error");
977         //CIM_ERR_NOT_SUPPORTED
978         $OperationError.CIMStatusCode = 7;
979         //Other
980         $OperationError.ErrorType = 1;
981         //Low
982         $OperationError.PerceivedSeverity = 2;
983         $OperationError.OwningEntity = DMTF:SMCLP;

```

```

984     $OperationError.MessageID = 0x00000001;
985     $OperationError.Message = "Operation is not supported.";
986     &smAddError($job, $OperationError);
987     &smMakeCommandStatus($job);
988     smEnd;
989 }
990 else {
991     //unspecified return code, generic failure
992     $OperationError = smNewInstance("CIM_Error");
993     //CIM_ERR_FAILED
994     $OperationError.CIMStatusCode = 1;
995     //Other
996     $OperationError.ErrorType = 1;
997     //Low
998     $OperationError.PerceivedSeverity = 2;
999     $OperationError.OwningEntity = DMTF:SMCLP;
1000     $OperationError.MessageID = 0x00000002;
1001     $OperationError.Message = "Failed. No further information is available.";
1002     &smAddError($job, $OperationError);
1003     &smMakeCommandStatus($job);
1004     smEnd;
1005 }
1006 } //end smReset()

```

1007 7.2.7 smResetRSC

1008 This function implements an invocation of the RequestStateChange() method with a RequestedState
1009 parameter of "Reset". This function is the generic mapping of the reset verb to an instance of a sub-class
1010 of CIM_EnabledLogicalElement. This function uses the RequestStateChange() extrinsic method as
1011 opposed to the smReset function (see section 7.2.6), which uses the Reset() extrinsic method.

```

1012 //PRECONDITIONS
1013 // 1. $target-> parameter contains the object name of the target instance
1014
1015 sub void smResetRSC($target->) {
1016     &smRequestStateChange($target->, "Reset");
1017     smEnd;
1018 }

```

1019 7.2.8 smSetInstance

1020 This function sets the property values for the specified instance.

```

1021 Sub void smSetInstance ( $instance, string #propertyName[], string propertyValues[] )
1022 {
1023     if ( 0 != #propertyName.length ) {
1024         for #i in #propertyName[] {
1025             $instance.<#propertyName[#i]> = #propertyValues[#i]
1026         }
1027         #Error = &smOpModifyInstance ( $instance, #propertyName[] );
1028         if ( 0 != #Error.code ) {
1029             &smProcessOpError ( #Error );
1030             //includes smEnd;
1031         }
1032         #Error = &smOpGetInstance ( $instance.getObjectPath(), #propertyName[],
1033             $outInstance );
1034         if ( 0 != #Error.code )
1035         {
1036             &smProcessOpError ( #Error );
1037             //includes smEnd;
1038         }
1039         &smDisplayInstance ( $outInstance, #propertyName[] );
1040     }

```

1041 7.2.9 smShowAssociationInstances—One Reference

1042 This function enumerates and displays association instances; these instances might be sorted by a
 1043 specific property value in descending or ascending order. Due to various uses of this function with
 1044 inconsistent parameter lists, there are multiple versions of the function defined and the signatures do not
 1045 follow a pattern.

1046 7.2.9.1 Method Signature One—One Reference

1047 This function enumerates and displays all instances of an association class that reference an instance.
 1048 The association instances are sorted by the value of a property, if one is specified.

1049 Note that because this version of the function does not take a list of properties, the default behavior is to
 1050 return all mandatory, non-key properties.

```

1051 Sub void smShowAssociationInstances (string #assocClassName, $containerInstancePath->,
1052 string #propertyNameSortBy, boolean #descendingOrder = true ) {
1053
1054     #Error = &smOpReferences ( $containerInstancePath->, #assocClassName, #className,
1055     NULL, NULL, NULL, $outAssocInstancePaths->[]);
1056     if (0 != #Error.code)
1057     {
1058         &smProcessOpError (#Error);
1059         //includes smEnd;
1060     }
1061     &smShowInstancesByInstancePaths ( $outAssocInstancePaths->[], #propertyNameSortBy,
1062     #descendingOrder );
1063 }
  
```

1064 7.2.9.2 Method Signature Two—One Reference

1065 This function enumerates and displays all instances of an association class that reference an instance. If
 1066 an array of property names is specified, only those properties will be displayed for each instance of the
 1067 association.

```

1068 Sub void smShowAssociationInstances (string #assocClassName, $containerInstancePath->,
1069 #propertyNamesToShow[] = null ) {
1070
1071     if (null == #propertyNamesToShow) {
1072         #propertyNamesToShow = { //array of mandatory non-key properties};
1073     }
1074
1075     #Error = &smOpReferences ( $containerInstancePath->, #assocClassName, #className,
1076     NULL, NULL, NULL, $outAssocInstancePaths->[]);
1077     if (0 != #Error.code)
1078     {
1079         &smProcessOpError (#Error);
1080         //includes smEnd;
1081     }
1082     &smShowInstancesByInstancePaths ( $outAssocInstancePaths->[], #propertyNameSortBy,
1083     #descendingOrder, #propertyNamesToShow[] );
  
```

1084 7.2.9.3 smShowAssociationInstances Method Signature One—Two References

1085 This function enumerates and displays association instances; these instances might be sorted by a
 1086 specific property value in descending or ascending order. Note that because this version of the function
 1087 does not take a list of properties, the default behavior is to return all mandatory, non-key properties.

```

1088 sub void smShowAssociationInstances (string #assocClassName, $instancePathA->,
1089 $instancePathB->,string #propertyNameSortBy, boolean #descendingOrder = true ) {
1090
1091     #Error = &smOpReferences ( $instancePathA->, #assocClassName, #className, NULL,
1092     NULL, NULL, $outAssocInstancePathsA->[]);
1093     if (0 != #Error.code)
1094     {
1095         &smProcessOpError (#Error);
1096         //includes smEnd;
1097     }
1098     #Error = &smOpReferences ( $instancePathB->, #assocClassName, #className, NULL,
1099     NULL, NULL, $outAssocInstancePathsB->[]);
1100     if (0 != #Error.code)
1101     {
1102         &smProcessOpError (#Error);
1103         //includes smEnd;
1104     }
1105     #outIndex = 0;
1106     for $instancePathA-> in $outAssocInstancePathsA->[] {
1107         if ( contains($instancePathA->,$outInstancePathsB->[] ) {
1108             $outAssocInstancePathsBoth->[$outIndex++] = $instancePathA->;
1109         }
1110     }
1111     &smShowInstancesByInstancePaths ( $outAssocInstancePathsBoth->[],
1112     #propertyNameSortBy, #descendingOrder );
1113 }

```

1114 7.2.9.4 smShowAssociationInstances Method Signature Two—Two References

1115 This function enumerates and displays association instances. If an array of property names is specified,
 1116 only those properties will be displayed for each instance of the association.

```

1117 sub void smShowAssociationInstances (string #assocClassName, $instancePathA->,
1118 $instancePathB->, #propertyNamesToShow[] = null) {
1119
1120     if (null == #propertyNamesToShow) {
1121         #propertyNamesToShow = { //array of mandatory non-key properties};
1122     }
1123
1124     #Error = &smOpReferences ( $instancePathA->, #assocClassName, #className, NULL,
1125     NULL, NULL, $outAssocInstancePathsA->[]);
1126     if (0 != #Error.code)
1127     {
1128         &smProcessOpError (#Error);
1129         //includes smEnd;
1130     }
1131
1132     #Error = &smOpReferences ( $instancePathB->, #assocClassName, #className, NULL,
1133     NULL, NULL, $outAssocInstancePathsB->[]);
1134     if (0 != #Error.code)
1135     {
1136         &smProcessOpError (#Error);
1137         //includes smEnd;
1138     }

```

```

1136     #outIndex = 0;
1137     for $instancePathA-> in $outAssocInstancePathsA->[] {
1138         if ( contains($instancePathA->,$outInstancePathsB->[]) {
1139             $outAssocInstancePathsBoth->[$outIndex++] = $instancePathA->;
1140         }
1141     }
1142     &smShowInstancesByInstancePaths ( $outAssocInstancePathsBoth->[],
1143     #propertyNameSortBy, #descendingOrder, #propertyNamesToShow[] );
1144 }

```

1145 7.2.10 smShowInstance

1146 For a specified instance, this function displays all of the property values or a subset of the property values
 1147 specified using the #propertyNamesToShow[] array parameter.

```

1148 sub void smShowInstance ( $instance, #propertyNamesToShow[] ) {
1149     &smDisplayInstance ( $instance, #propertyNamesToShow[] );
1150 }

```

1151 7.2.11 smShowInstancePseudoProperties

1152 For a specified instance, this function displays all of the property values or a subset of the property values
 1153 specified using the #propertyNamesToShow[] array parameter. This includes support for annotating
 1154 referenced properties that have been added to the instance.

```

1155 sub void smShowInstancePseudoProperties( $Object, #propertynamelist[],
1156 #pseudopropertylist[] ) {
1157     &smDisplayInstance ( $instance, #propertyNamesToShow[],#pseudopropertylist[] );
1158 }

```

1159 7.2.12 smShowInstances

1160 This function enumerates and displays instances; these instances might be sorted by a specific property
 1161 value in descending or ascending order. Optionally, the #propertyNamesToShow[] array parameter can
 1162 be used to restrict the returned property values to a specific set.

1163 7.2.12.1 Method Signature One

1164 This function displays all instances of the class identified by the className parameter that are associated
 1165 through instances of the association class identified by the addressAssocClassName parameter with the
 1166 instance identified by the containerInstancePath parameter. Instances are returned irrespective of the
 1167 roles the instances play in the association.

```

1168     Sub void smShowInstances (string #className,
1169     String #addressAssocClassName,
1170     string $containerInstancePath->,
1171     string #propertyNameSortBy,
1172     boolean #descendingOrder = true,
1173     #propertyNamesToShow[] = null ) {
1174
1175     &smShowInstances (#className, #addressAssocClassName, NULL, NULL,
1176     $containerInstancePath->, #propertyNameSortBy, #descendingOrder,
1177     #propertyNamesToShow[]);
1178 }

```

1179 7.2.12.2 Method Signature Two

1180 This function displays all instances of the class identified by the className parameter that are associated
 1181 through instances of the association class identified by the addressAssocClassName parameter with the

1182 instance identified by the containerInstancePath parameter. This version of the function provides the
1183 ability to filter instances based on their role in the association.

```

1184 sub void smShowInstances (string #className, string
1185 #addressAssocClassName,
1186 string #role=NULL, string #resultRole=NULL,
1187 $containerInstancePath->, string
1188 #propertyNameSortBy, boolean #descendingOrder = true, #propertyNamesToShow[] ) {
1189 {
1190     #Error = &smOpAssociators ( $containerInstancePath->, #addressAssocClassName,
1191     #className, #role, #resultRole, NULL, $outInstancePaths->[]);
1192     if (0 != #Error.code)
1193     {
1194         &smProcessOpError (#Error);
1195         //includes smEnd;
1196     }
1197     for (#i=0; I < $outInstancePaths->[].length; i++) {
1198         &lShowRole($outInstancePaths->[i++];
1199     }
1200 }

```

1201 7.2.13 smShowInstancesByInstancePaths

1202 This function enumerates and displays instances; these instances might be sorted by a specific property
1203 value in descending or ascending order. Optionally, the #propertyNamesToShow[] array parameter can
1204 be used to restrict the returned property values to a specific set.

```

1205 sub void smShowInstancesByInstancePaths ( $instancePaths->[], string
1206 #propertyNameSortBy, boolean #descendingOrder = true, #propertyNamesToShow[] = null )
1207 {
1208     if (0 != $instancePaths->[].length() )
1209     {
1210         if ( NULL != propertyNameSortBy )
1211         {
1212             $sortedInstancePaths->[] = &smSortInstancePaths ( $instancePaths->[],
1213 #propertyNameSortBy, #descendingOrder );
1214             for #i in $sortedInstancePaths->[]
1215             {
1216                 #Error = &smGetInstance ( sortedInstancePaths->[#i], $instance );
1217                 if (0 != #Error.code) {
1218                     &smProcessOpError (#Error);
1219                     //includes smEnd;
1220                 }
1221                 &smDisplayInstance ( $instance, #propertyNamesToShow[] );
1222             }
1223         }
1224         else {
1225             for #i in $instancePaths->[]
1226             {
1227                 #Error = &smGetInstance ( $instancePaths->[#i], $instance );
1228                 if (0 != #Error.code)
1229                 {
1230                     &smProcessOpError (#Error);
1231                     //includes smEnd;
1232                 }
1233                 &smDisplayInstance ( $instance, #propertyNamesToShow[] );
1234             }
1235         }
1236     }

```

1237 **7.2.14 smShowInstancesPseudoProperties**

1238 This function shows instances and any referenced properties that have been attached to the instances.

```

1239 sub void smShowInstancesPseudoProperties( $Objects[], #propertynamelist[],
1240 #pseudopropertylist[]) {
1241
1242     for #i in $Objects[]
1243     {
1244         &smDisplayInstance ( $instance,
1245         #propertyNamesToShow[],#pseudopropertylist[]);
1246     }
1247 }

```

1248 **7.2.15 smStartRSC**

1249 This function implements an invocation of the RequestStateChange() method with a RequestedState
1250 parameter of "Enabled". This function is the generic mapping of the start verb to an instance of a sub-
1251 class of CIM_EnabledLogicalElement.

```

1252 //PRECONDITIONS
1253 // 1. $target-> parameter contains the object name of the target instance
1254 sub void smStartRSC($target->) {
1255     &smRequestStateChange($target->, "Enabled");
1256     smEnd;
1257 }

```

1258 **7.2.16 smStopRSC**

1259 This function implements an invocation of the RequestStateChange() method with a RequestedState
1260 parameter of "Disabled". This function is the generic mapping of the stop verb to an instance of a sub-
1261 class of CIM_EnabledLogicalElement.

```

1262 //PRECONDITIONS
1263 // 1. $target-> parameter contains the object name of the target instance
1264
1265 sub void smStartRSC($target->) {
1266     &smRequestStateChange($target->, "Disabled");
1267     smEnd;
1268 }

```

1269 **7.3 SM CLP-to-CIM Command Mapping Functions for Intrinsic Operations**

1270 This section defines wrapper functions for the intrinsic operations defined in [DSP0200](#).

1271 **7.3.1 smOpAssociators**

1272 The SM CLP-to-CIM command mapping specifications use this function to find CIM instances associated
1273 with a target CIM instance. This function corresponds to the Associators intrinsic operation that is defined
1274 in section 2.3.2.14 of [DSP0200](#). The function signature is as follows:

```

1275 Sub <ERROR>smOpAssociators(
1276     [IN] $instancePath->,
1277     [IN] string #assocClass,
1278     [IN,OPTIONAL,NULL] string #resultClass = NULL,
1279     [IN,OPTIONAL,NULL] string #role = NULL,
1280     [IN,OPTIONAL,NULL] string #resultRole = NULL,
1281     [IN,OPTIONAL,NULL] string #propertyList[] = NULL,
1282     [OUT] $associatedInstances>[])

```

1283 <Error> is defined in section A.2.3.

1284 \$instancePath-> is a reference to the CIM Instance that is the root of the search space.

1285 \$assocClass identifies an association class. Results are filtered such that instances returned are
 1286 associated with \$ObjectName through instances of this association.

1287 #resultClass is an optional filter on the class of instances that will be returned.

1288 #role is an optional identifier for the Role that the target instance plays in the association.

1289 #resultRole is an optional identifier for the Role that the result instances play in the association.

1290 propertyList[] is an array listing the names of the properties that shall be included.
 1291 propertyList[] shall be either NULL, which indicates that all properties of the instance are included,
 1292 or an array of property name strings, which indicates that only the properties specified in the array are
 1293 included. propertyList[] shall not be an empty array.

1294 \$associatedInstances[] is an array that, upon successful completion of the function, contains
 1295 CIMObjectPath values for CIM instances that meet the filter criteria.

1296 7.3.2 smOpAssociatorNames

1297 The SM CLP-to-CIM command mapping specifications use this function to find CIM instances associated
 1298 with a target CIM instance. This function corresponds to the Associators intrinsic operation that is defined
 1299 in section 2.3.2.14 of [DSP0200](#). The function signature is as follows:

```
1300 Sub <ERROR>smOpAssociatorNames(
1301     [IN] $instancePath->,
1302     [IN] string #assocClass,
1303     [IN,OPTIONAL,NULL] string #resultClass = NULL,
1304     [IN,OPTIONAL,NULL] string #role = NULL,
1305     [IN,OPTIONAL,NULL] string #resultRole = NULL,
1306     [IN,OPTIONAL,NULL] string #propertyList[] = NULL,
1307     [OUT] $associatedInstancePaths->[])
```

1308 <Error> is defined in section A.2.3.

1309 \$instancePath-> is a reference to the CIM Instance that is the root of the search space.

1310 \$assocClass identifies an association class. Results are filtered such that instances returned are
 1311 associated with \$ObjectName through instances of this association.

1312 #resultClass is an optional filter on the class of instances that will be returned.

1313 #role is an optional identifier for the Role that the target instance plays in the association.

1314 #resultRole is an optional identifier for the Role that the result instances play in the association.

1315 propertyList[] is an array listing the names of the properties which shall be included.
 1316 propertyList[] shall be either NULL, which indicates that all properties of the instance are included,
 1317 or an array of property name strings, which indicates that only the properties specified in the array are
 1318 included. propertyList[] shall not be an empty array.

1319 \$associatedInstancePaths->[] is an array that, upon successful completion of the function,
 1320 contains CIMObjectPath values for CIM instances that meet the filter criteria.

1321 7.3.3 smOpReferences

1322 This operation is used to enumerate the association objects that refer to a particular target CIM Object
 1323 (Class or Instance).

1324 The SM CLP-to-CIM command mapping specifications use this function to find CIM association instances
 1325 that refer to a target CIM instance. This function corresponds to the References intrinsic operation that is
 1326 defined in section 2.3.2.16 of [DSP0200](#). The function signature is as follows:


```

1327 Sub <ERROR>smOpReferences (
1328     [IN] $instancePath->,
1329     [IN,OPTIONAL,NULL] string #resultClass = NULL,
1330     [IN,OPTIONAL,NULL] string #role = NULL,
1331     [IN,OPTIONAL,NULL] string #resultRole = NULL,
1332     [IN,OPTIONAL,NULL] string #propertyList[] = NULL,
1333     [OUT] $referencedInstances[])

```

1334 <Error> is defined in section A.2.3.

1335 \$instancePath-> is a CIMObjectPath that identifies the CIM Instance that is the root of the search
1336 space.

1337 #resultClass is an optional filter on the class of instances that will be returned.

1338 #role is an optional identifier for the Role that the target instance plays in the association.

1339 #resultRole is an optional identifier for the Role that the result instances play in the association.

1340 propertyList[] is an array listing the names of the properties that shall be included.

1341 propertyList[] shall be either NULL, which indicates that all properties of the instance are included,
1342 or an array of property name strings, which indicates that only the properties specified in the array are
1343 included. propertyList[] shall not be an empty array.

1344 \$referencedInstances[] is an array that, upon successful completion of the function, contains
1345 CIMObjectPath values for CIM instances that meet the filter criteria.

1346 7.3.4 smOpReferenceNames

1347 This operation is used to enumerate the association objects that refer to a particular target CIM Object
1348 (Class or Instance).

1349 The SM CLP-to-CIM command mapping specifications use this function to find CIM association instances
1350 that refer to a target CIM instance. This function corresponds to the References intrinsic operation that is
1351 defined in section 2.3.2.16 of [DSP0200](#). The function signature is as follows:

```

1352 Sub <ERROR>smOpReferenceNames (
1353     [IN] $instancePath->,
1354     [IN,OPTIONAL,NULL] string #resultClass = NULL,
1355     [IN,OPTIONAL,NULL] string #role = NULL,
1356     [IN,OPTIONAL,NULL] string #resultRole = NULL,
1357     [IN,OPTIONAL,NULL] string #propertyList[] = NULL,
1358     [OUT] $referencedInstancePaths->[])

```

1359 <Error> is defined in section A.2.3.

1360 \$instancePath-> is a CIMObjectPath that identifies the CIM Instance that is the root of the search
1361 space.

1362 #resultClass is an optional filter on the class of instances that will be returned.

1363 #role is an optional identifier for the Role that the target instance plays in the association.

1364 #resultRole is an optional identifier for the Role that the result instances play in the association.

1365 propertyList[] is an array listing the names of the properties that shall be included.

1366 propertyList[] shall be either NULL, which indicates that all properties of the instance are included,
1367 or an array of property name strings, which indicates that only the properties specified in the array are
1368 included. propertyList[] shall not be an empty array.

1369 \$referencedInstancePaths->[] is an array that, upon successful completion of the function,
1370 contains CIMObjectPath values for CIM instances that meet the filter criteria.

1371 7.3.5 smOpCreateInstance

1372 The SM CLP-to-CIM command mapping specifications use this function to create a CIM instance. This
 1373 function corresponds to the CreateInstance intrinsic operation that is defined in section 2.3.2.6 of
 1374 [DSP0200](#). It is assumed that the caller has used the smNewInstance function (see section 7.1.9) to make
 1375 a local copy of the instance and is invoking this function to perform the update in the CIMOM. Prior to
 1376 invoking the function, key properties are required to have a value assigned only if the assignment of a
 1377 specific value is necessary for the clarity of the mapping that is creating the instance.

```
1378 Sub <ERROR>smOpCreateInstance (
1379     [IN] $instance)
```

1380 <Error> is defined in section A.2.3.

1381 \$instance is a reference to the CIM instance to be created.

1382 7.3.6 smOpDeleteInstance

1383 The SM CLP-to-CIM command mapping specifications use this function to delete a CIM instance. The
 1384 function also deletes all the association classes that reference the CIM instance. This function
 1385 corresponds to the DeleteInstance intrinsic operation that is defined in section 2.3.2.4 of [DSP0200](#).

```
1386 Sub <ERROR>smOpDeleteInstance (
1387     [IN] $instancePath->)
```

1388 <Error> is defined in section A.2.3.

1389 \$instancePath-> is a CIMObjectPath that identifies the CIM instance to be deleted.

1390 7.3.7 smOpEnumerateInstances

1391 The SM CLP-to-CIM command mapping specifications get CIM instances by using the
 1392 smOpEnumerateInstances() common function. This function corresponds to the EnumerateInstances
 1393 intrinsic operation that is defined in section 2.3.2.11 of [DSP0200](#). The function signature is as follows:

```
1394 Sub <ERROR>smOpEnumerateInstances ( [IN] string #className,
1395                                     [OUT] string propertyList[] = NULL,
1396                                     [OUT] $instances[])
```

1397 <Error> is defined in section A.2.3.

1398 #className identifies the target class name that is the basis for the enumeration.

1399 propertyList[] is an array listing the names of the properties that shall be included.

1400 propertyList[] shall be either NULL, which indicates that all properties of the instance are included,
 1401 or an array of property name strings, which indicates that only the properties specified in the array are
 1402 included. propertyList[] shall not be an empty array.

1403 \$instances is a reference to an array of returned CIMInstance values.

1404 7.3.8 smOpEnumerateInstanceNames

1405 The SM CLP-to-CIM command mapping specifications get CIM instances by using the
 1406 smOpEnumerateInstanceNames() common function. This function corresponds to the EnumerateInstanceNames
 1407 intrinsic operation that is defined in section 2.3.2.11 of [DSP0200](#). The function signature is as follows:

```
1408 Sub <ERROR>smOpEnumerateInstanceNames ( [IN] string #className,
1409                                           [OUT] $instancePaths->[])
```

1410 <Error> is defined in section A.2.3.

1411 #className is a string that identifies the target class name that is the basis for the enumeration.

1412 \$instancePaths->[] is a reference to an array of CIMObjectPath values that identifies the
1413 enumerated instances.

1414 7.3.9 smOpGetInstance

1415 The SM CLP-to-CIM command mapping specifications get CIM instances by using the
1416 smOpGetInstance() common function. This function corresponds to the GetInstance intrinsic operation
1417 that is defined in section 2.3.2.2 of [DSP0200](#). The function signature is as follows:

```
1418 Sub <ERROR>smOpGetInstance ( [IN] $instancePath->,
1419                             [IN] string #propertyList[],
1420                             [OUT] $instance )
```

1421 <Error> is defined in section A.2.3.

1422 \$instancePath-> is the CIMObjectPath to the instance to retrieve.

1423 propertyList[] is an array listing the names of the properties which shall be included.

1424 propertyList[] shall be either NULL, which indicates that all properties of the instance are included,
1425 or an array of property name strings, which indicates that only the properties specified in the array are
1426 included. propertyList[] shall not be an empty array.

1427 \$instance is a reference to the returned CIMInstance.

1428 7.3.10 smOpInvokeMethod

1429 This function is used to invoke an extrinsic method. This function corresponds to an extrinsic method call
1430 using the METHODCALL and METHODRESPONSE XML elements as specified in section 2.3.1 of
1431 [DSP0200](#). The function signature is as follows:

```
1432 Sub <ERROR>smOpInvokeMethod ( [IN] $instancePath->,
1433                               [IN] string methodName,
1434                               [IN] %inArguments[],
1435                               [OUT] %outArguments[],
1436                               [OUT] integer #returnValue)
```

1437 <Error> is defined in section A.2.3.

1438 \$instancePath-> identifies the CIM instance on which the method will be invoked.

1439 #methodName is the name of the method to invoke.

1440 %inArguments is the array of input arguments to the method. These are the parameters where the IN
1441 qualifier has a value of TRUE. The array is indexed by parameter name.

1442 %outArguments is the array of output arguments from the method. These are the parameters where the
1443 OUT qualifier has a value of TRUE. The array is indexed by parameter name.

1444 #returnValue is the return code from the extrinsic method.

1445 7.3.11 smOpModifyInstance

1446 The SM CLP-to-CIM command mapping specifications modify CIM instances by using the
1447 smOpModifyInstance() common function. This function corresponds to the ModifyInstance intrinsic
1448 operation that is defined in section 2.3.2.8 of [DSP0200](#). The function signature is as follows:

```
1449 Sub <ERROR>smOpModifyInstance($modifiedInstance,
1450                               string #propertyList[])
```

1451 <ERROR> is defined in section A.2.3.

1452 \$modifiedInstance is a modified CIMInstance.

1453 #propertyList[] is an array listing the names of the modified properties that should be updated.
 1454 propertyList[] shall not be an empty array or NULL.

1455 **7.4 SM CLP-to-CIM Common Messages**

1456 Table 2 lists the standard messages defined for the *SM CL-to-CIM Common Mapping Specification*. The
 1457 Owning Entity data element must have a value of DMTF:SMCLP for all messages defined in this
 1458 specification.

1459 **Table 2 – Common SM CLP Messages**

Message ID	Message	Message Arguments	Notes
0x00000000	Request completed successfully.	None	None
0x00000001	Operation is not supported.	None	None
0x00000002	Failed. No further information is available.	None	None
0x00000003	Operation cannot complete within specified timeout period.	None	The operation was not initiated because it cannot complete within timeout period.
0x00000004	One or more parameters specified are invalid.	None	
0x00000005	Requested access is not supported.	None	SharedDeviceManagementService.Sharedevice() return code 6.
0x00000006	Use of timeout parameter is not supported.	None	None
0x00000007	The target device is busy and cannot be re-assigned.	None	None
0x00000008	Timeout expired prior to completion of operation.	None	Example: RequestStateChange() return code 3
0x00000009	An internal software error has occurred.	None	
0x0000000A	The target is busy and its state cannot be changed.	None	
0x0000000B	The target cannot transition to the requested state from its current state.	None	
0x0000000C	The target already exists.	None	
0x0000000D	The switch is in use, and the port mappings cannot be changed.	None	Example: AssignPorts() return code 3
0x0000000E	The specified ports are not mapped and cannot be unmapped.	None	Example: AssignPorts() return code 4
0x0000000F	The selected configuration is already active. Use the force option to re-apply it.	None	This error is appropriate when a client attempts to apply a configuration that is already active and to the mapping requires the force option to reapply the configuration.
0x00000010	The target association cannot be created between the specified targets.	None	This error is appropriate when a client attempts to create an association between two targets.

Message ID	Message	Message Arguments	Notes
0x00000011	A property value is incorrectly formatted.	None	This error is appropriate when the format used for a property value is incorrect.

1460 8 SM CLP-to-CIM Common Verb Mappings

1461 This section defines the mappings for verbs that are supported across all mappings.

1462 8.1.1 cd

1463 This section describes how to implement the cd verb irrespective of the current default target or the
 1464 resultant target of a CLP command. Implementations must support the use of the cd verb for any
 1465 command target.

1466 8.1.1.1 Using cd without a Command Target Term

1467 This command form corresponds to the cd verb specified without a command target term, in which the
 1468 only behavior is to echo back the current default target.

1469 **Command Form:** cd

1470 **CIM Requirements:** <CIM_CLPProtocolEndpoint.CurrentDefaultTarget>

1471 Behavior Requirements:

1472 Pseudo code:
 1473 \$session = &smGetSession();
 1474 //echo \$session.CurrentDefaultTarget
 1475 &smCommandCompleted(\$job);
 1476
 1477

1478 8.1.1.2 Using cd with a Command Target Term

1479 This command form corresponds to the cd verb specified with a command line term that is resolved to an
1480 absolute address.

1481 **Command Form:** `cd <CIM_ManagedElement single object>`

1482 **CIM Requirements:**

1483 **Behavior Requirements:**

1484 #address contains the UFiP that results from evaluating the command target term.

1485

1486 //this will generate an error if the address can't be resolved to an object path

1487 \$instance-> = &smGetObjectPath(#address);

1488 \$session = &smGetSession();

1489 \$session.CurrentDefaultTarget = \$instance->;

1490

1491 #Error = &smOpModifyInstance (\$session, {"CurrentDefaultTarget"});

1492 if (0 != #Error.code) {

1493 &smProcessOpError (#Error);

1494 //includes smEnd;

1495 }

1496 //echo \$session.CurrentDefaultTarget

1497 &smCommandCompleted(\$job);

1498 8.1.2 exit

1499 This section describes how to implement the exit verb. The behavior of the exit verb is unaffected by the
1500 Current Default Target.

1501 8.1.2.1 exit

1502 This command form corresponds to all uses of the exit verb.

1503 **Command Form:** `exit`

1504 **CIM Requirements:**

1505 **Behavior Requirements:**

1506 \$Session = &smGetSession();

1507 &smDeleteInstance(\$Session.getObjectPath());

1508 &smEnd;

1509 8.1.3 help

1510 This section describes how to implement the help verb. The behavior of the help verb is not affected by
1511 the current default target.

1512 8.1.3.1 help

1513 This command form applies to all uses of the help verb.

1514 **Command Form:** `help`

1515 **CIM Requirements:**

1516 **Behavior Requirements:**

1517

1518 //display help, behavior is implementation specific

```
1519     &smCommandCompleted($job);
```

1520 **8.1.4 version**

1521 This section describes how to implement the version verb. The behavior of the version verb is not
1522 affected by the Current Default Target.

1523 **8.1.4.1 version**

1524 This command form corresponds to all uses of the version verb.

1525 **Command Form:** `version`

1526 **CIM Requirements:**

1527 **Behavior Requirements:**

```
1528
1529 $session = &smGetSession();
1530
1531 //1 Find the Service for the session
1532 #Error = &smOpAssociatorNames(
1533     $session->,
1534     "CIM_ProvidesEndpoint",
1535     "CIM_ProtocolService",
1536     NULL,
1537     NULL,
1538     NULL,
1539     $SvcInstancePaths[])
1540
1541 if (0 != #Error.code)
1542 {
1543     &smProcessOpError (#Error);
1544     //includes smEnd;
1545 }
1546
1547 //2 Find the capabilities for the Service
1548 #Error = &smOpAssociators(
1549     $$SvcInstancePaths->[0],
1550     "CIM_ElementCapabilities",
1551     "CIM_CLPCapabilities",
1552     NULL,
1553     NULL,
1554     NULL,
1555     $CapInstances[])
1556
1557 if (0 != #Error.code)
1558 {
1559     &smProcessOpError (#Error);
1560     //includes smEnd;
1561 }
1562
1563 $cap = $CapInstances[0];
1564 //echo $cap.CLPVersions[]
1565 //echo $cap.SMMEAddressVersions[];
1566 &smCommandCompleted($job);
```

1563

ANNEX A (informative)

Conventions

1564
1565
1566
1567
1568

1569 A.1 Notation

1570 The following notations are used:

1571 *<string>* Italicized text enclosed in angle brackets indicates that a substitution
1572 is to be made in the text where this string appears.

1573 CIM_Classname, *CIM_Classname*

1574 The full CIM class name is used to reference a CIM class (including
1575 the prefix "CIM_"). If the class is an abstract class, the entire class
1576 name string appears in italics.

1577 <CIM_Classname> Angle brackets on either side of a CIM class name represent a
1578 target address term that resolves into one or more instances of the
1579 particular CIM class that is referenced. When this target address
1580 form is used, the command form is valid for single or multiple
1581 instances of the referenced class.

1582 A.2 Pseudo-code

1583 The command mappings in this specification are documented using pseudo-code to define the
1584 normalized behavior of the command execution. The pseudo-code sections are not intended to be
1585 compiled but are intended to be interpreted according to the particular CIM server interface used in the
1586 SM CLP implementation.

1587 A.2.1 Pseudo-code Syntax

1588 The pseudo-code conventions utilized in this document extend the Recipe Conventions that are defined
1589 in [Storage Management Initiative Specification](#), section 7.6.

1590 A.2.2 Pseudo-code Conventions

1591 The pseudo-code used to define the command mapping behaviors uses the following conventions:

- 1592 • `smEnd` indicates the end of the command mapping sequence for a single command mapping
1593 algorithm.
- 1594 • When a function signature includes the assignment of a value to a parameter, the value
1595 assigned is the default value for the parameter, if the parameter is not specified when the
1596 method is called.

1597 A.2.3 <ERROR> Data Type

1598 The <ERROR> data type is patterned after the ERROR element returned within a METHODRESPONSE
 1599 or IMETHODRESPONSE, as described in [DSP0201](#), section 3.2.6.12. It contains the fields listed in Table
 1600 A.1.

1601 **Table A.1 – ERROR Data Type Fields**

Field	Notes
code	Code Attribute of <ERROR> element. A value of 0 indicates that the operation was successful. All other values are per the table in section 2.3.1.3 of DSP0200 .
description	Description attribute of <ERROR> element
\$error[]	Possibly null array of CIM_Error instances returned by operation

1602 A.2.4 CommandStatus Data Type

1603 The CommandStatus data type represents a Command Status construct as defined in the [DSP0214](#).
 1604 Table A.2 lists the fields in the data type and provides references to the relevant text within the *Server
 1605 Management Command Line Protocol Specification*.

1606 **Table A.2 – CommandStatus Data Type Fields**

Field	Notes
status	See DSP0214 , Table 4.
status_tag	See DSP0214 , Table 4.
job_id	See DSP0214 , Table 8.
Errtype	See DSP0214 , Table 8.
Errtype_desc	See DSP0214 , Table 8.
Cimstat	See DSP0214 , Table 8.
Cimstat_desc	See DSP0214 , Table 8.
Severity	See DSP0214 , Table 8.
Severity_desc	See DSP0214 , Table 8.
Probcause	See DSP0214 , Table 8.
Probcause_desc	See DSP0214 , Table 8.
Recmdaction	See DSP0214 , Table 8.
Errsource	See DSP0214 , Table 8.
Errsourceform	See DSP0214 , Table 8.
Errsourceform_desc	See DSP0214 , Table 8.
Messages	One or more <Message> instances. See section A.2.5 for a definition of the structure.

1607 At most, one instance of the CommandStatus data type is in scope for an operation. This instance is
 1608 universally addressed as <CommandStatus>. Fields in the data type are addressed using standard dot
 1609 notation.

1610 For example, the `status` field is addressed as follows:

1611 `<CommandStatus>.status`

1612 See section 7.1.3 for example usage.

1613 **A.2.5 <Message> Data Type**

1614 The <Message> data type groups the set of CLP Command Status elements that comprise a single
 1615 message. This data type is not intended to be used independently of the <CommandStatus> data type.
 1616 Table A.3 lists the fields in the data type and provides references to the relevant text within the *Server*
 1617 *Management Command Line Protocol Specification*.

1618 **Table A.3 – Message Data Type Fields**

Field	Notes
message	See DSP0214 , Table 6.
message_id	See DSP0214 , Table 6.
message_arg	An array of message arguments. Each index corresponds to an occurrence of the <code>message_arg</code> keyword in the CLP Command Status. See DSP0214 , Table 6.
Owningentity	See DSP0214 , Table 6.

1619

ANNEX B (informative)

Per-Profile Mappings

1625 The SM CLP-to-CIM command mappings for classes identified in a CIM Profile are documented in a
1626 separate per-profile SM CLP-to-CIM command mapping specification.

1627 **B.1 Preconditions and Assumptions for Per-Profile SM CLP-to-CIM Command** 1628 **Mapping Behaviors**

1629 Each of the command mappings in the per-profile command mapping specifications document the
1630 execution of a single SM CLP command issued to an implementation. The “Behavior Requirements”
1631 subsections contain the pseudo-code representation of the algorithm to be implemented to execute each
1632 command in a conformant manner.

1633 **B.1.1 Preconditions**

1634 – Job instance

1635 The implementation will create an instance of a CIM_Job to represent and track the execution of the
1636 command. The instance is stored in the variable #jobId. Any CIM_Error instances created are to be
1637 associated with this CIM_Job instance.

1638 **B.1.2 Assumptions**

1639 – Synchronous execution

1640 The algorithms documented in the “Behavior Requirements” subsections are assumed to be
1641 executed synchronously by the implementation. Any interruptions or parallel processing must be
1642 managed by the implementation. The resulting implementation must return command results as if the
1643 command execution were performed uninterrupted.

1644 – Output stream

1645 Because the algorithms are assumed to be executed synchronously, the algorithms contain
1646 statements that indicate where output elements are to be generated sequentially into the output. If
1647 any buffering, caching, storing, or reorganization of the output data is performed, it is the
1648 responsibility of the implementation to produce the output in a conformant form.

1649 A SM CLP-to-CIM command mapping specification specifies the level of support for each command that
1650 implementations must provide for CIM classes identified by a CIM profile. SM CLP command mappings
1651 specify the detailed description of command behavior when applied to instances of a specific CIM class,
1652 including expected state change and property value change behaviors. For each SM CLP command, the
1653 per-profile command mapping defines its “Support Requirement” by selecting one of the following levels
1654 of support: “shall”, “should”, “may”, or “shall not”.

1655 For each SM CLP command that has a “Support Requirement” of “shall”, “should”, or “may”, the per-
1656 profile command mapping provides a “Description and Usage” statement and one or more Command
1657 Forms with corresponding CIM and Behavior Requirements. The “Description and Usage” statement
1658 describes, in general for the CIM class, how the user uses the command to manipulate a target instance
1659 of the class. Below the “Description and Usage” statement, the specification defines one or more
1660 command mappings, using the requirements specified in this *SM CLP-to-CIM Common Mapping*
1661 *Specification*.

1662 Following each verb section header, each “Command Form” subsection denotes a particular form of the
1663 command that is specified by the *SM CLP-to-CIM Common Mapping Specification*. Each subsection
1664 describes a particular use of the verb, any required options and option arguments, target object
1665 instances, and property names and values specific to that use of the command verb.

1666 For each command form, two types of specification are given:

- 1667 • The “CIM Requirements” section specifies the particular CIM classes, properties, and methods
1668 that are mapped to the command in some manner. In most cases, this mapping is direct. In some
1669 cases, the mapping is indirect and may involve associations to related instances of other classes
1670 or multiple references into the same class.
- 1671 • The “Behavior Requirements” section specifies any detailed mapping requirements that must be
1672 followed by the implementation. The “Behavior Requirements” are documented using the pseudo-
1673 code syntax defined in section A.2.

1674 Commands may act on either a specific instance or many instances. In the “Command Form” section, the
1675 notation “<CIM_classname single instance>” is used to denote that a single instance target is
1676 used in the command form. If the command is supported for multiple instances of the class, a second
1677 command mapping item is included, specifying the behavior when the target address term resolves to
1678 multiple instances of the class. The notation “<CIM_classname multiple instances>” indicates
1679 that the command target resolves into multiple instances in this command form.

1680

1681
1682
1683
1684
1685

ANNEX C (informative)

Change Log

1686

Version	Date	Author	Description
1.0.0	4/23/2009		DMTF Standard Release

1687

Bibliography

1688

1689 DMTF DSP0217, *SMASH Implementation Requirements*, 1.0.0,
1690 http://www.dmtf.org/standards/published_documents/DSP0217_1.0.0.pdf

1691 DMTF DSP2001, *Systems Management Architecture for Server Hardware (SMASH) Command Line
1692 Protocol (CLP) Architecture White Paper*, 1.0.0,
1693 http://www.dmtf.org/standards/published_documents/DSP2001_1.0.1.pdf

1694