



1
2

3

4

5

Document Number: DSP1057

Date: 2010-04-22

Version: 1.0.0

6 **Virtual System Profile**

7 **Document Type: Specification**

8 **Document Status: DMTF Standard**

9 **Document Language: E**

10 Copyright Notice

11 Copyright © 2007, 2010 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
15 to 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 inaccu-
20 rate identification or disclosure of such rights, owners or claimants. DMTF shall have no liability to any
21 party, in any manner or circumstance, under any legal theory whatsoever, for failure to recognize, dis-
22 close, or identify any such third party patent rights, or for such party's reliance on the standard or incorpo-
23 ration thereof in its product, protocols or testing procedures. DMTF shall have no liability to any party im-
24 plementing such standard, whether such implementation is foreseeable or not, nor to any patent owner or
25 claimant, and shall have no liability or responsibility for costs or losses incurred if a standard is withdrawn
26 or modified after publication, and shall be indemnified and held harmless by any party implementing the
27 standard from any and all claims of infringement by a patent owner for such implementations.

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

CONTENTS

32	1	Scope	9
33	2	Normative references	9
34	3	Terms and definitions	10
35	4	Symbols and abbreviated terms	11
36	5	Synopsis	12
37	6	Description	13
38	6.1	Profile relationships	13
39	6.2	Virtual system class schema	15
40	6.3	Virtual system concepts: Definition, instance, representation, and configuration	16
41	6.4	Virtual system states and transitions	17
42	6.4.1	Virtual system states	17
43	6.4.2	Virtual system state transitions	18
44	6.4.3	Summary of virtual system states and virtual system state transitions	19
45	7	Implementation	21
46	7.1	Virtual system	21
47	7.1.1	CIM_ComputerSystem.EnabledState property	21
48	7.1.2	CIM_ComputerSystem.RequestedState property	22
49	7.2	Virtual resource	24
50	7.3	Virtual system configuration	24
51	7.3.1	Structure	24
52	7.3.2	The "state" virtual system configuration	24
53	7.3.3	The "defined" virtual system configuration	25
54	7.3.4	Implementation approaches for "state" and "defined" virtual system configuration	25
55	7.3.5	Other types of virtual system configurations	26
56	7.3.6	CIM_VirtualSystemSettingData.Caption property	26
57	7.3.7	CIM_VirtualSystemSettingData.Description property	27
58	7.3.8	CIM_VirtualSystemSettingData.ElementName property	27
59	7.3.9	CIM_VirtualSystemSettingData.VirtualSystemIdentifier property	27
60	7.3.10	CIM_VirtualSystemSettingData.VirtualSystemType property	27
61	7.3.11	CIM_ElementSettingData.IsDefault property	27
62	7.3.12	CIM_ElementSettingData.IsNext property	28
63	7.4	Profile registration	29
64	7.4.1	This profile	29
65	7.4.2	Scoped profiles	29
66	7.5	Capabilities	29
67	7.6	Client state management	29
68	7.7	Power state management	30
69	7.7.1	CIM_AssociatedPowerManagementService.PowerState property	30
70	8	Methods	30
71	8.1	Extrinsic methods	31
72	8.1.1	CIM_ComputerSystem.RequestStateChange() method	31
73	8.1.2	CIM_PowerManagementService.RequestPowerStateChange() method	31
74	8.2	Profile conventions for operations	32
75	8.2.1	CIM_ComputerSystem	32
76	8.2.2	CIM_ConcreteJob	32
77	8.2.3	CIM_ElementSettingData	33
78	8.2.4	CIM_EnabledLogicalElementCapabilities	33
79	8.2.5	CIM_ReferencedProfile	33
80	8.2.6	CIM_RegisteredProfile	33
81	8.2.7	CIM_VirtualSystemSettingData	33
82	8.2.8	CIM_VirtualSystemSettingDataComponent	33
83	9	Use-cases	33
84	9.1	Virtual system detection and inspection	33
85	9.1.1	Discover conformant virtual systems using SLP	33
86	9.1.2	Determine a virtual system's state and other properties	34

87	9.1.3	Determine the "defined" virtual system configuration	35
88	9.1.4	Determine the virtual system structure	36
89	9.1.5	Determine resource type support	37
90	9.1.6	Determine the next boot configuration	41
91	9.2	Virtual system operation	41
92	9.2.1	Change virtual system state	41
93	9.2.2	Activate virtual system	42
94	10	CIM elements	42
95	10.1	CIM_AffectedJobElement	43
96	10.2	CIM_ComputerSystem	44
97	10.3	CIM_ConcreteJob	44
98	10.4	CIM_ElementConformsToProfile	44
99	10.5	CIM_ElementSettingData	45
100	10.6	CIM_EnabledLogicalElementCapabilities	46
101	10.7	CIM_PowerManagementService	46
102	10.8	CIM_ReferencedProfile	46
103	10.9	CIM_RegisteredProfile	46
104	10.10	CIM_SettingsDefineState	47
105	10.11	CIM_VirtualSystemSettingData	47
106	10.12	CIM_VirtualSystemSettingDataComponent	48
107	Annex A (Informative)	Virtual system modeling — background information	49
108	Annex B (Informative)	Implementation details	52
109	Annex C (Informative)	Change Log	56

110

111 Figures

112	Figure 1 – Profiles related to system virtualization	14
113	Figure 2 – Virtual System Profile: Class diagram	15
114	Figure 3 – Virtual system states	20
115	Figure 4 – Virtual system representation and virtual system configuration	24
116	Figure 5 – Sample virtual system configuration	35
117	Figure 6 – Sample virtual system in "active" state	36
118	Figure 7 – Instance diagram: Profile conformance of scoped resources	37
119	Figure 8 – State-dependent presence of model elements	50
120	Figure 9 – Sample virtual system in "defined" state (Dual-configuration approach)	52
121	Figure 10 – Sample virtual system in a state other than "defined" (Dual-configuration approach)	53
122	Figure 11 – Sample virtual system in the "defined" state (Single-configuration approach)	54
123	Figure 12 – Sample virtual system in a state other than "defined" (Single-configuration approach)	55

124

125 Tables

126	Table 1 – Related profiles	12
127	Table 2 – Observation of virtual system states	21
128	Table 3 – Observation of virtual system state transitions	23
129	Table 4 – CIM_ComputerSystem.RequestStateChange() method: Parameters	31
130	Table 5 – CIM_PowerManagementService.RequestPowerStateChange() method: Parameters	31
131	Table 6 – CIM elements: Virtual System Profile	43
132	Table 7 – Association: CIM_AffectedJobElement	43
133	Table 8 – Class: CIM_ComputerSystem	44
134	Table 9 – Class: CIM_ConcreteJob	44
135	Table 10 – Association: CIM_ElementConformsToProfile	45

136 Table 11 – Association: CIM_ElementSettingData 45

137 Table 12 – Class: CIM_EnabledLogicalElementCapabilities 46

138 Table 13 – Association: CIM_ReferencedProfile 46

139 Table 14 – Class: CIM_RegisteredProfile 47

140 Table 15 – Association: CIM_SettingsDefineState 47

141 Table 16 – Class: CIM_VirtualSystemSettingData 47

142 Table 17 – Association: CIM_VirtualSystemSettingDataComponent 48

143

145

Foreword

146 This profile — the *Virtual System Profile* (DSP1057) — was prepared by the System Virtualization, Parti-
147 tioning and Clustering Working Group of the DMTF.

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

150 Acknowledgments

151 The authors wish to acknowledge the following people.

- 152 • Editor:
 - 153 – Michael Johanssen – IBM
- 154 • Participants from the DMTF System Virtualization, Partitioning and Clustering Working Group:
 - 155 – Gareth Bestor – IBM
 - 156 – Chris Brown – HP
 - 157 – Mike Dutch - Symantec
 - 158 – Jim Fehlig – Novell
 - 159 – Kevin Fox – Sun Microsystems, Inc.
 - 160 – Ron Goering – IBM
 - 161 – Steve Hand - Symantec
 - 162 – Daniel Hiltgen – EMC / VMware
 - 163 – Michael Johanssen – IBM
 - 164 – Larry Lamers – EMC / VMware
 - 165 – Andreas Maier - IBM
 - 166 – Aaron Merkin – IBM
 - 167 – John Parchem – Microsoft
 - 168 – Nihar Shah – Microsoft
 - 169 – David Simpson – IBM
 - 170 – Carl Waldspurger – EMC / VMware

171

Introduction

172 The information in this specification should be sufficient for a provider or consumer of this data to identify
173 unambiguously the classes, properties, methods, and values that shall be instantiated and manipulated to
174 represent and manage the components described in this document. The target audience for this specifi-
175 cation is implementers who are writing CIM-based providers or consumers of management interfaces that
176 represent the components described in this document.

177

Virtual System Profile

178 1 Scope

179 This profile — the *Virtual System Profile* — is an autonomous profile that defines the minimum object
180 model needed to provide for the inspection of a virtual system and its components. In addition, it defines
181 optional basic control operations for activating, deactivating, pausing, or suspending a virtual system.

182 2 Normative references

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

186 DMTF DSP0004, *CIM Infrastructure Specification 2.5*

187 http://www.dmtf.org/standards/published_documents/DSP0004_2.5.pdf

188 DMTF DSP0200, *CIM Operations over HTTP 1.3*

189 http://www.dmtf.org/standards/published_documents/DSP0200_1.3.pdf

190 DMTF DSP0205, *WBEM Discovery Using the Service Location Protocol (SLP) 1.0*

191 http://www.dmtf.org/standards/published_documents/DSP0205_1.0.pdf

192 DMTF DSP1001, *Management Profile Specification Usage Guide 1.0*

193 http://www.dmtf.org/standards/published_documents/DSP1001_1.0.pdf

194 DMTF DSP1012, *Boot Control Profile 1.0*

195 http://www.dmtf.org/standards/published_documents/DSP1012_1.0.pdf

196 DMTF DSP1022, *CPU Profile 1.0*

197 http://www.dmtf.org/standards/published_documents/DSP1022_1.0.pdf

198 DMTF DSP1026, *System Memory Profile 1.0*

199 http://www.dmtf.org/standards/published_documents/DSP1026_1.0.pdf

200 DMTF DSP1027, *Power State Management Profile 1.0*

201 http://www.dmtf.org/standards/published_documents/DSP1027_1.0.pdf

202 DMTF DSP1033, *Profile Registration Profile 1.0*

203 http://www.dmtf.org/standards/published_documents/DSP1033_1.0.pdf

204 DMTF DSP1041, *Resource Allocation Profile 1.1*

205 http://www.dmtf.org/standards/published_documents/DSP1041_1.1.pdf

206 DMTF DSP1042, *System Virtualization Profile 1.0*

207 http://www.dmtf.org/standards/published_documents/DSP1042_1.0.pdf

208 DMTF DSP1043, *Allocation Capabilities Profile 1.0*

209 http://www.dmtf.org/standards/published_documents/DSP1043_1.0.pdf

210 DMTF DSP1044, *Processor Device Resource Virtualization Profile 1.0*

211 http://www.dmtf.org/standards/published_documents/DSP1044_1.0.pdf

- 212 DMTF DSP1045, *Memory Resource Virtualization Profile 1.0*
213 http://www.dmtf.org/standards/published_documents/DSP1045_1.0.pdf
- 214 DMTF DSP1047, *Storage Resource Virtualization Profile 1.0*
215 http://www.dmtf.org/standards/published_documents/DSP1047_1.0.pdf
- 216 DMTF DSP1052, *Computer System Profile 1.0*
217 http://www.dmtf.org/standards/published_documents/DSP1052_1.0.pdf
- 218 DMTF DSP1059, *Generic Device Resource Virtualization Profile 1.0*
219 http://www.dmtf.org/standards/published_documents/DSP1059_1.0.pdf
- 220 ISO/IEC Directives, Part2:2004, *Rules for the structure and drafting of International Standards*,
221 <http://isotc.iso.org/livelink/livelink.exe?func=ll&objId=4230456&objAction=browse&sort=subtype>

222 **3 Terms and definitions**

223 For the purposes of this document, the following terms and definitions apply. For the purposes of this
224 document, the terms and definitions given in [DSP1033](#), [DSP1001](#), and [DSP1052](#) also apply.

225 **3.1**

226 **can**

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

228 **3.2**

229 **cannot**

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

231 **3.3**

232 **conditional**

233 indicates requirements strictly to be followed in order to conform to the document and from which no de-
234 viation is permitted when the specified conditions are met

235 **3.4**

236 **mandatory**

237 indicates requirements strictly to be followed in order to conform to the document and from which no de-
238 viation is permitted

239 **3.5**

240 **may**

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

242 **3.6**

243 **need not**

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

245 **3.7**

246 **optional**

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

248 **3.8**

249 **referencing profile**

250 indicates a profile that owns the definition of this class and can include a reference to this profile in its
251 "Related Profiles" table

- 252 **3.9**
 253 **shall**
 254 indicates requirements strictly to be followed in order to conform to the document and from which no de-
 255 viation is permitted
- 256 **3.10**
 257 **shall not**
 258 indicates requirements strictly to be followed in order to conform to the document and from which no de-
 259 viation is permitted
- 260 **3.11**
 261 **should**
 262 indicates that among several possibilities, one is recommended as particularly suitable, without mention-
 263 ing or excluding others, or that a certain course of action is preferred but not necessarily required
- 264 **3.12**
 265 **should not**
 266 indicates that a certain possibility or course of action is deprecated but not prohibited
- 267 **3.13**
 268 **unspecified**
 269 indicates that this profile does not define any constraints for the referenced CIM element
- 270 **3.14**
 271 **implementation**
 272 a set of CIM providers that realize the classes specified by this profile
- 273 **3.15**
 274 **client**
 275 an application that exploits facilities specified by this profile
- 276 **3.16**
 277 **virtualization platform**
 278 virtualizing infrastructure provided by a host system enabling the deployment of virtual systems
- 279 **3.17**
 280 **WBEM service**
 281 an abstract class for WBEM services such as the ObjectManager, providers, CIM repositories, or any
 282 other WBEM Server component. It is a type of service that provides associated capabilities and details
 283 about the component.

284 **4 Symbols and abbreviated terms**

- 285 **4.1**
 286 **CIM**
 287 Common Information Model
- 288 **4.2**
 289 **RASD**
 290 CIM_ResourceAllocationSettingData
- 291 **4.3**
 292 **SLP**
 293 service location protocol

294 **4.4**
 295 **VS**
 296 virtual system
 297 **4.5**
 298 **VSSD**
 299 CIM_VirtualSystemSettingData

300 **5 Synopsis**

301 **Profile Name:** Virtual System

302 **Version:** 1.0.0

303 **Organization:** DMTF

304 **CIM Schema Version:** 2.22

305 **Central Class:** CIM_ComputerSystem

306 **Scoping Class:** CIM_ComputerSystem

307 This profile is an autonomous profile that defines the minimum object model needed to provide for the
 308 inspection of a virtual system and its components. In addition, it defines optional basic control operations
 309 for activating, deactivating, pausing, or suspending a virtual system.

310 The instance of the CIM_ComputerSystem class representing a virtual system shall be the central in-
 311 stance and the scoping instance of this profile.

312 Table 1 lists related profiles that this profile depends on, or that may be used in context of this profile.
 313 [DSP1052](#) lists additional related profiles; these relationships are not further specified in this profile.

314

Table 1 – Related profiles

Profile Name	Organization	Version	Relationship	Description
Profile Registration	DMTF	1.0	Mandatory	The profile that specifies registered profiles.
Computer System	DMTF	1.0	Specialization	The abstract autonomous profile that specifies the minimum object model needed to define a basic computer system.
<i>Power State Management</i>	DMTF	1.0	Optional	The component profile that specifies an object model needed to describe and manage the power state of server systems.
<i>Boot Control</i>	DMTF	1.0	Optional	The component profile that specifies an object model that represent boot configurations, including boot devices and computer system settings used during booting.

315 6 Description

316 This profile (DSP1057, Virtual System Profile) specializes [DSP1052](#) (*Computer System Profile*) that de-
 317 fines the minimum top-level object model needed to define a basic computing platform. The primary de-
 318 sign objective applied by this profile is that a virtual system and its components appear to a client in the
 319 same way as a non-virtual system. Typical management tasks such as enumerating, analyzing, control-
 320 ling, or configuring a system should be enabled without requiring the client to understand specific aspects
 321 of virtual systems.

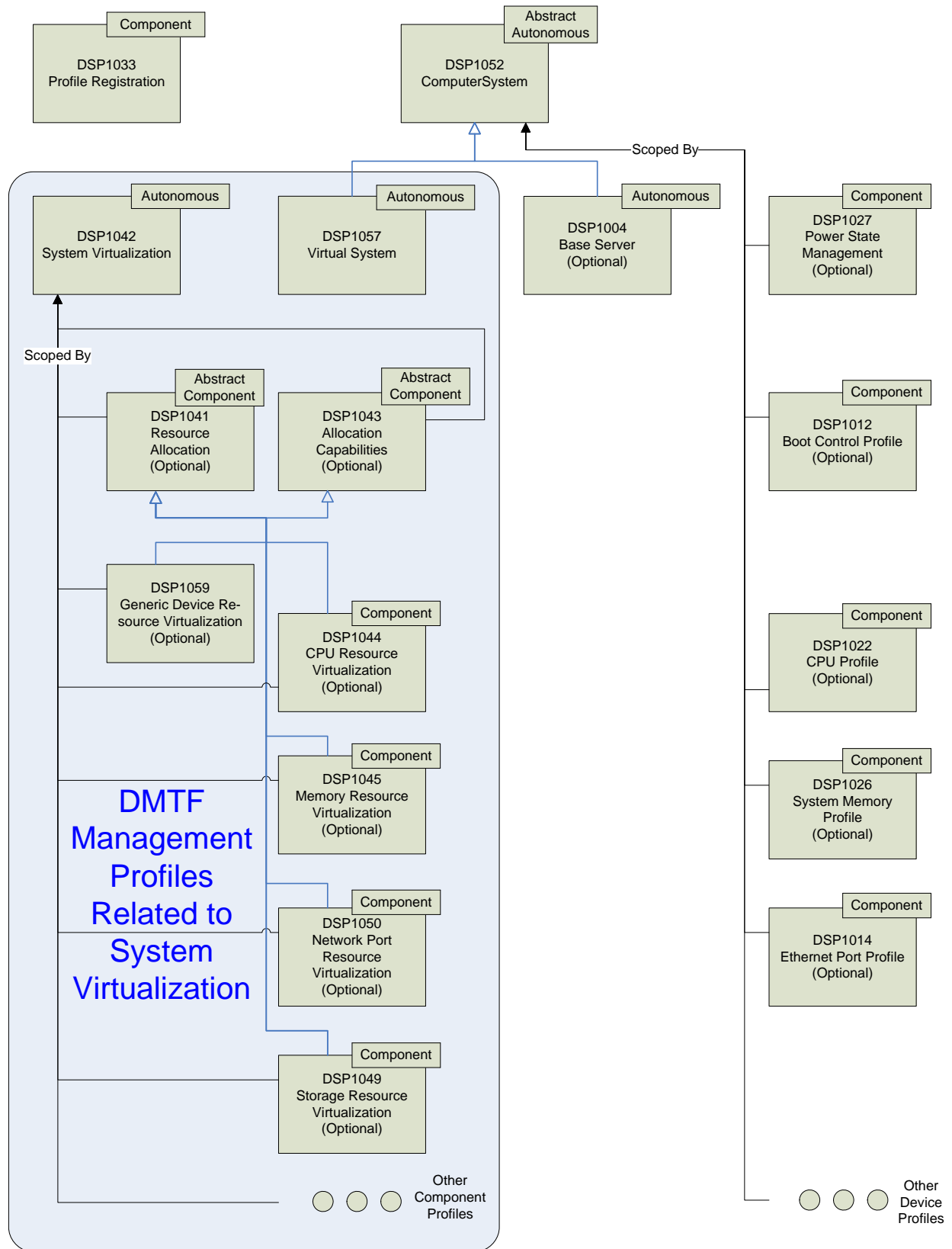
322 6.1 Profile relationships

323 This profile (DSP1057) is complementary to [DSP1042](#) (*System Virtualization Profile*):

- 324 • This profile focuses on virtualization aspects that relate to virtual systems and their virtual re-
 325 sources, such as modeling the *structure* of virtual systems and their resources. The profile in-
 326 troduces the concept of virtual system configurations allowing the inspection of virtual system
 327 configuration and state information.
- 328 • [DSP1042](#) focuses on virtualization aspects that relate to host systems and their resources, such
 329 as modeling the *relationships* between host resources and virtual resources. Further it ad-
 330 dresses virtualization-specific tasks such as the creation or modification of virtual systems and
 331 their configurations.

332 Figure 1 shows a structure of profiles. For example, an implementation that instruments a virtualization
 333 platform may implement some of the following profiles:

- 334 • This profile (DSP1057)
 335 This profile enables the inspection of and basic operations on virtual systems.
- 336 • [DSP1042](#)
 337 [DSP1042](#) enables the inspection of host systems, their capabilities, and their services for crea-
 338 tion and manipulation of virtual systems.
- 339 • Resource-type-specific profiles
 340 Resource-type-specific profiles enable the inspection and operation of resources for one par-
 341 ticular resource type. They apply to both virtual and host resources; they do not cover virtualiza-
 342 tion-specific aspects of resources. A client may exploit resource-type-specific management pro-
 343 files for the inspection and manipulation of virtual and host resources in a similar manner.
- 344 • Resource allocation profiles
 345 Resource allocation profiles enable the inspection of existing resource allocations and of host
 346 and other resources available for allocation. Resource allocation profiles are based on
 347 [DSP1041](#) and [DSP1043](#), and they are scoped by [DSP1042](#). A client may exploit resource allo-
 348 cation profiles to inspect all of the following:
 - 349 – the allocation of resources
 - 350 – the allocation dependencies that virtual resources have on host resources and resource
 351 pools
 - 352 – the capabilities describing possible values for resource allocations
 - 353 – the capabilities describing the mutability of resource allocations
- 354 • [DSP1059](#) (*Generic Device Resource Virtualization Profile*) is a resource-type-independent re-
 355 source allocation profile that specifies the management of the allocation of basic virtual re-
 356 sources. For some resource types, specific resource allocation profiles are defined that address
 357 resource-type-specific allocation aspects and capabilities.



358

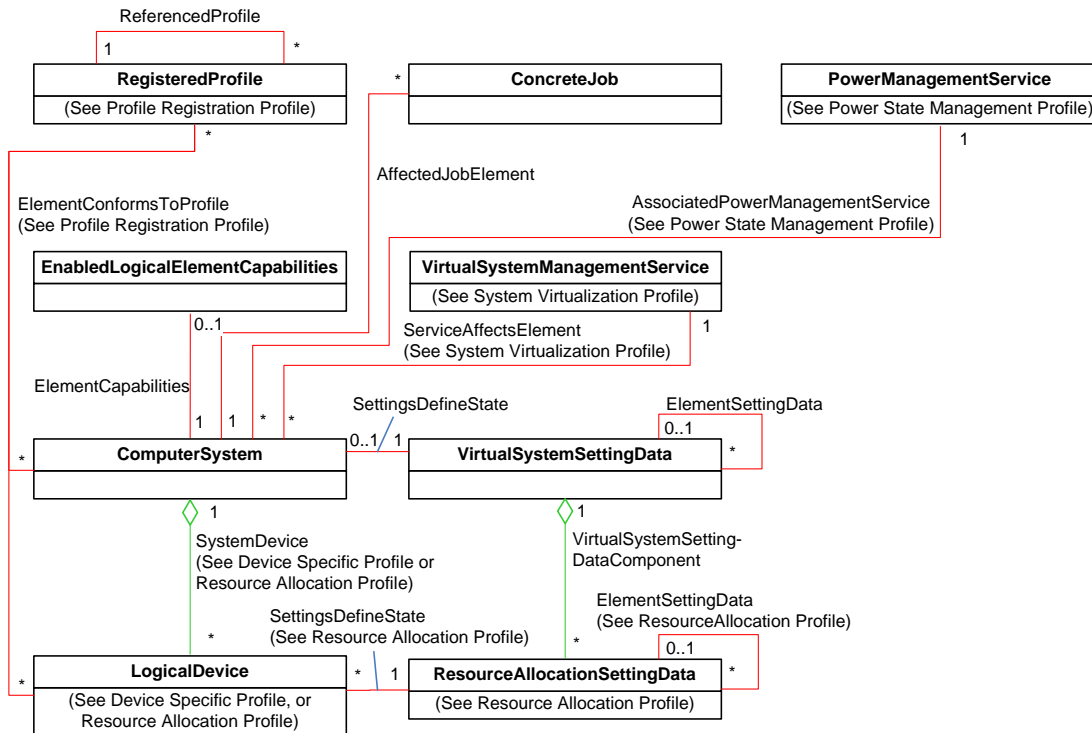
359

Figure 1 – Profiles related to system virtualization

360 **6.2 Virtual system class schema**

361 Figure 2 shows the class schema of this profile. It outlines the elements that are owned or specialized by
 362 this profile, as well as the dependency relationships between elements of this profile and other profiles.
 363 For simplicity in diagrams the prefix CIM_ has been removed from class and association names.

364 [DSP1052](#) references additional classes in its class diagram that outline relationships with certain
 365 resources, services, and protocol endpoints. This profile (DSP1057) provides no specialization of these
 366 dependencies. For that reason they are not shown in the class diagram. For details, refer to [DSP1052](#)
 367 and to the component profiles referenced there.



368

369

Figure 2 – Virtual System Profile: Class diagram

370 This profile specifies the use of the following classes and associations:

- 371 • the CIM_ComputerSystem class to represent virtual systems
- 372 • the CIM_RegisteredProfile class and the CIM_ElementConformsToProfile association to model
 373 conformance with this profile
- 374 • the CIM_ReferencedProfile association to model dependencies between this profile and
 375 resource-type-specific resource allocation profiles
- 376 • the CIM_EnabledLogicalElementCapabilities class and the CIM_ElementCapabilities
 377 association to model capabilities of a virtual system such as characteristics of certain properties
 378 or the set of potential state transitions
- 379 • the CIM_VirtualSystemSettingData class to model virtualization-specific aspects of a virtual
 380 system
- 381 • the CIM_VirtualSystemSettingDataComponent association to model the aggregation of instan-
 382 ces of the CIM_ResourceAllocationSettingData class to one instance of the CIM_VirtualSystem-
 383 SettingData class, forming a virtual system configuration

- 384 • the CIM_SettingsDefineState association to model the relationship between an instance of the
385 CIM_ComputerSystem class representing a virtual system and an instance of the CIM_Virtual-
386 SystemSettingData class representing virtualization specific aspects of that virtual system
- 387 • the CIM_ElementSettingData association to model the relationship between an element and
388 configuration data applicable to the element
- 389 • the CIM_ConcreteJob class and the CIM_AffectedJobElement association to model a mecha-
390 nism that allows tracking of asynchronous tasks resulting from operations such as the optional
391 RequestStateChange() method applied to instances of the CIM_ComputerSystem class

392 In general, any mention of a class in this document means the class itself or its subclasses. For example,
393 a statement such as "an instance of the CIM_LogicalDevice class" implies an instance of the CIM_Logi-
394 calDevice class or a subclass of the CIM_LogicalDevice class.

395 For information about modeling concepts applied in this profile, see Annex A.

396 **6.3 Virtual system concepts: Definition, instance, representation, and configu- 397 ration**

398 The term *virtual system definition* refers to a virtualization platform's internal description of a virtual sys-
399 tem and its virtual resources. A typical realization of a virtual system definition is an entry within a configu-
400 ration file with a set of formal configuration statements. The virtual system definition may be regarded as
401 the recipe that a virtualization platform uses in the process of creating a virtual system instance. Except
402 for persistent resource allocations, a virtual system definition does not cause the reservation or consump-
403 tion of resources.

404 The term *virtual system instance* refers to a virtualization platform's internal representation of the virtual
405 system and its components. A typical realization of a virtual system instance is a set of interrelated data
406 structures in memory. During instantiation all elements of a virtual system instance are allocated such that
407 the virtual system is enabled to perform tasks.

408 The term *virtual system representation* refers to the set of CIM class instances that represent the current
409 state of a virtual system instance. A virtual system representation consists of one top-level instance of the
410 CIM_ComputerSystem class and a set of aggregated instances of the CIM_LogicalDevice class. The
411 state of the system and logical devices is thus represented by the set of property values in these in-
412 stances. Virtualization specific state is not yet represented; for that purpose the next paragraph intro-
413 duces a virtualization specific state extension to the virtual system representation. The presence of in-
414 stances of the CIM_LogicalDevice class within the virtual system representation is controlled by speciali-
415 zations of [DSP1041](#). The specializations describe how instances of the CIM_LogicalDevice class are
416 added or removed from the virtual system representation as virtual resources are allocated or de-
417 allocated.

418 The term *virtual system configuration* refers to an aggregation of instances of the CIM_SettingData class:
419 One top-level instance of the CIM_VirtualSystemSettingData class and a set of aggregated instances of
420 the CIM_ResourceAllocationSettingData class. This profile specifies the use of virtual system configura-
421 tions for two principal purposes:

- 422 • Virtual system configurations are used for the representation of configuration information, in par-
423 ticular for the representation of virtual system definitions.
- 424 • Virtual system configurations are used for the representation of virtualization specific "State" that
425 extends the virtual system representation. A single "state" virtual system configuration is associ-
426 ated to a virtual system. Elements of the "state" virtual system configuration extend corresponding
427 elements of the virtual system representation with virtualization-specific properties. A variety of vir-
428 tual system configurations may be associated with the "state" configuration via the CIM_Element-
429 SettingData association. An example is the representation of the virtual system definition by a
430 separate "Defined" virtual system configuration.

431 Virtualization platforms may support modifications on virtual system definitions or virtual system instances
432 through various means, for example through direct configuration file editing, through a command-line in-
433 terface, through a program interface, or through a CIM-based interface as modeled in [DSP1042](#). Regard-
434 less of the mechanism used to effect a modification on a virtual system definition or a virtual system in-
435 stance that modification becomes visible to clients through the CIM model view defined in this profile
436 (DSP1057), as expressed by the respective virtual system configuration or the virtual system representa-
437 tion.

438 **6.4 Virtual system states and transitions**

439 This subclause informally describes virtual system states and virtual system state transitions. Clause 7
440 normatively specifies how states and state transitions are observed, and a mechanism for the initiation of
441 state transitions.

442 **6.4.1 Virtual system states**

443 This subclause describes various virtual system states and their semantics. Normative requirements for
444 the observation of virtual system states are specified in 7.1.1.

445 **6.4.1.1 Semantics of the "defined" state**

446 In the "defined" state a virtual system is defined at the virtualization platform, but the virtual system and its
447 virtual resources need not be instantiated by the virtualization platform. A virtual system in the "defined"
448 state is not enabled to perform tasks. In this state the virtual system does not consume any resources of
449 the virtualization platform, with the exception of persistent resource allocations that remain allocated re-
450 gardless of the virtual system state. An example is virtual disk allocations.

451 **6.4.1.2 Semantics of the "active" state**

452 In the "active" state a virtual system is instantiated at the virtualization platform. Generally the virtual re-
453 sources are enabled to perform tasks. For example, virtual processors of the virtual system are enabled
454 to execute instructions. Other virtual resources are enabled to perform respective resource-type-specific
455 tasks. Nevertheless some virtual resources may not be enabled to perform tasks for various reasons like
456 for example missing resource allocation. A virtual system is considered to be in the "active" state as soon
457 a transition is initiated from another state, and as long as a transition from the "active" state to another
458 state is not yet complete. Examples are the activation and deactivation of virtual systems.

459 **6.4.1.3 Semantics of the "paused" state**

460 In the "paused" state the virtual system and its virtual resources remain instantiated and resources remain
461 allocated as in the "active" state, but the virtual system and its virtual resources are not enabled to per-
462 form tasks.

463 **6.4.1.4 Semantics of the "suspended" state**

464 In the "suspended" state the state of the virtual system and its virtual resources are stored on non-volatile
465 storage. The system and its resources are not enabled to perform tasks. It is implementation-dependent
466 whether virtual resources continue to be represented by instances of the CIM_LogicalDevice class even if
467 some or all resources allocated to the virtual resources were de-allocated.

468 **6.4.1.5 Vendor-defined states**

469 Additional vendor-defined states for virtual systems are possible. This profile specifies mechanisms allow-
470 ing the observation of vendor-defined states, but does not specify vendor-specific state semantics.

471 6.4.1.6 Semantics of the "unknown" state

472 "unknown" is a pseudo-virtual system state indicating that the present virtual system state cannot be de-
473 termined. For example, the implementation may not be able to contact the virtualization platform hosting
474 the virtual system because of networking problems.

475 6.4.2 Virtual system state transitions

476 This subclause describes various virtual system state transitions and their semantics. Normative require-
477 ments for the observation of virtual system state transitions are specified in 7.1.2.

478 A virtual system state transition is the process of changing the state of a virtual system from an initial
479 state to a target state. It is implementation-dependent, at which point a state transition becomes visible
480 through the CIM model.

481 6.4.2.1 The "define" state transition

482 This is a virtualization-specific operation addressing the definition of new virtual system within a virtualiza-
483 tion platform. It is described in the *System Virtualization Profile* and is named here for completeness only.

484 6.4.2.2 Semantics of the "activate" state transition

485 While performing the "activate" state transition from the "defined" state, missing resources are allocated
486 according to the virtual system definition, the virtual system and its virtual resources are instantiated and
487 enabled to perform tasks.

488 While performing an "activate" state transition from the "suspended" state back to the "active" state any
489 resources that were de-allocated during the transition to and while the system was in the "suspended"
490 state are re-allocated, all virtual resources are restored to their previous state and the virtual system is re-
491 enabled to perform tasks, continuing from the point before the system was suspended.

492 In both cases it is possible that some virtual resources were not instantiated for various reasons. For ex-
493 ample, a resource backing the virtual resource might not be available. In this case it is implementation
494 dependent whether the whole activation fails or whether the activation continues with a reduced set of
495 resources.

496 While performing an "activate" state transition from the "paused" state back to the "active" state the virtual
497 system and its resources are re-enabled to perform tasks continuing from the point before the system was
498 paused.

499 6.4.2.3 Semantics of the "deactivate" state transition

500 While performing the "deactivate" state transition the virtual system and its virtual resources are disabled
501 to perform tasks, non-persistent virtual resources are released, their backing resources are de-allocated,
502 and the virtual system instance is removed from the virtualization platform. If a "deactivate" state transi-
503 tion originates from the "suspended" state, previously saved state information of virtual system and re-
504 sources is removed. The virtual system remains defined at the virtualization platform.

505 NOTE The "deactivate" transition is assumed to be disruptive with respect to the virtual system and its components
506 performing tasks.

507 6.4.2.4 Semantics of the "pause" state transition

508 While performing the "pause" state transition the virtual system and its virtual resources are disabled to
509 perform tasks. The virtual system and its virtual resources remain instantiated with their backing re-
510 sources allocated.

511 6.4.2.5 Semantics of the "suspend" state transition

512 While performing the "suspend" state transition the virtual system and its virtual resources are disabled to
513 perform tasks and the state of the virtual system and its resources are saved to non-volatile storage. Re-
514 sources may be de-allocated.

515 6.4.2.6 Semantics of the "shut down" state transition

516 While performing the "shut down" state transition from the "active" state, the software that is executed by
517 the virtual system is notified to shut down. It is assumed that the software then terminates all its tasks and
518 terminates itself. Subsequent steps of the "shut down" state transition should be the same as for the "de-
519 activate" state transition.

520 6.4.2.7 Semantics of the "reboot" state transition

521 While performing the "reboot" state transition, the software that is executed by the virtual system is noti-
522 fied to re-cycle or re-boot. Virtual resources remain instantiated with their backing resources allocated.

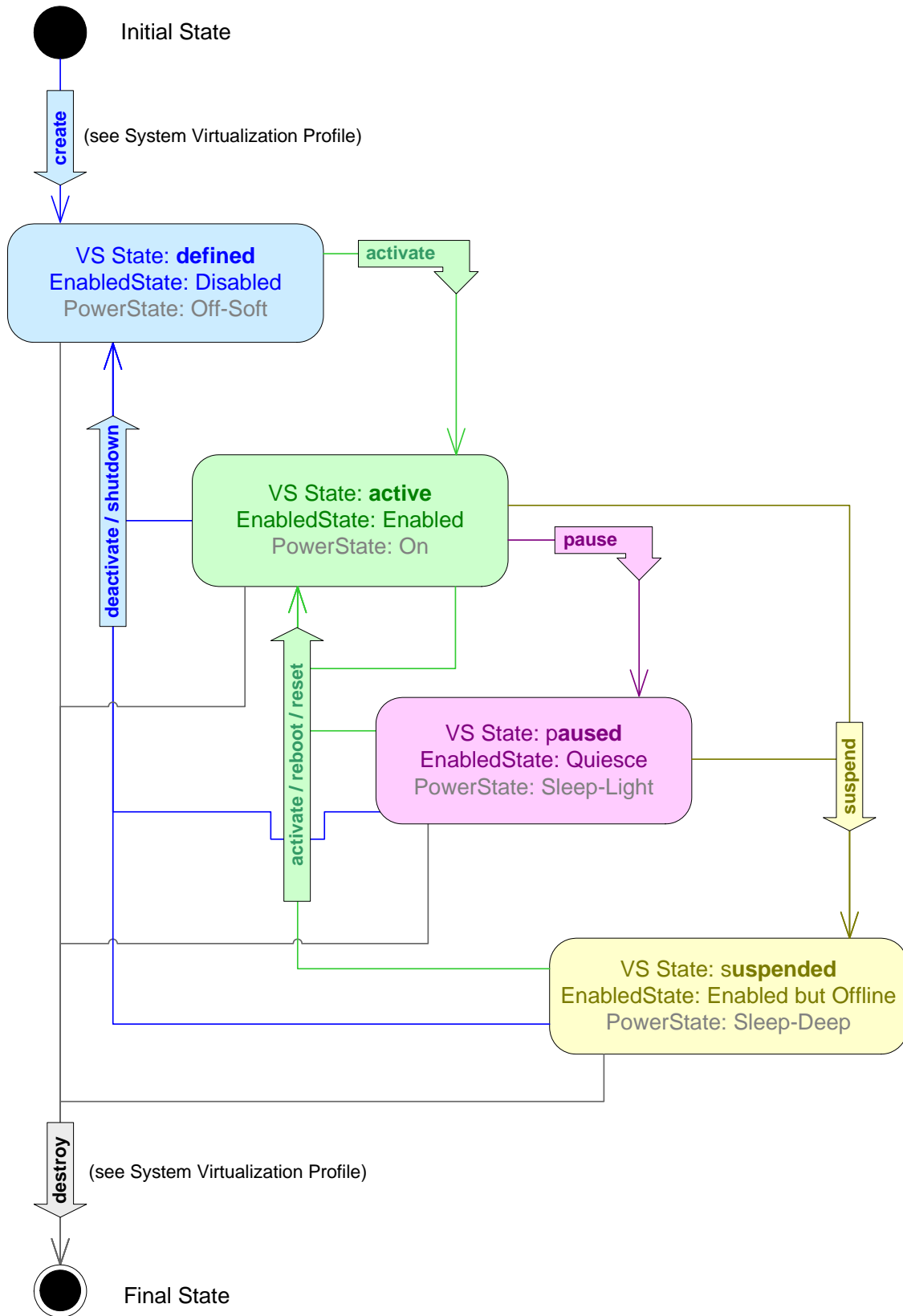
523 6.4.2.8 Semantics of the "reset" state transition

524 Logically the "reset" state transition consists of a "deactivate" state transition followed by an "activate"
525 state transition, except that resource are not de-allocated during deactivation and thus need not be re-
526 allocated during activation..

527 NOTE The "reset" transition is assumed to be disruptive with respect to the virtual system and its components per-
528 forming tasks, and state information of the virtual system and its resources may be lost, including state in-
529 formation saved during a previous "Suspend" state transition.

530 6.4.3 Summary of virtual system states and virtual system state transitions

531 Figure 3 summarizes virtual system states that are assumed by this profile and possible state transitions
532 between those states. Further, Figure 3 shows the mapping of virtual system states to properties of the
533 CIM_ComputerSystem class and the CIM_AssociatedPowerManagementService association.



534

535

Figure 3 – Virtual system states

536 **7 Implementation**

537 This clause details the requirements related to classes and their properties for implementations of this
 538 profile. The CIM Schema descriptions for any referenced element and its sub-elements apply.

539 The list of all methods covered by this profile is in clause 8. The list of all properties covered by this profile
 540 is in clause 10.

541 In references to CIM Schema properties that enumerate values, the numeric value is normative and the
 542 descriptive text following it in parenthesis is informational. For example, in the statement "If an instance of
 543 the CIM_VirtualSystemManagementCapabilities class contains the value 3 (DestroySystemSupported) in
 544 an element of the SynchronousMethodsSupported[] array property", the "value 3" is normative text and
 545 "(DestroySystemSupported)" is descriptive text.

546 **7.1 Virtual system**

547 The CIM_ComputerSystem class shall be used to represent virtual systems. One instance of the
 548 CIM_ComputerSystem class shall exist for each virtual system that is conformant to this profile, regard-
 549 less of its state.

550 This subclause and its secondary subclauses apply to instances of the CIM_ComputerSystem class that
 551 represent virtual systems.

552 **7.1.1 CIM_ComputerSystem.EnabledState property**

553 The EnabledState property shall be implemented and used as the primary means to support the observa-
 554 tion of virtual system state (see 6.4.1). Note that as a particular virtual system state is observed through
 555 the value of the EnabledState property a state transition to a different state may already be in progress;
 556 this issue is resolved by modeling the observation of state transitions through the value of the Re-
 557 questedState property (see 7.1.2).

558 The "defined" and "active" states as defined in 6.4.1 shall be implemented; support of additional states is
 559 optional.

560 Table 2 provides the normative mapping of virtual system states to values of the EnabledState property.
 561 The value of the EnabledState property shall be set depending on the state of the virtual system. For ex-
 562 ample, if a virtual system is in the "active" state then the EnabledState property should have a value of 2
 563 (Enabled), but may have a value of 8 (Deferred) or 4 (Shutting Down) if respective conditions apply, as
 564 defined by the description of the CIM_EnabledLogicalElement class in the CIM Schema.

565 **Table 2 – Observation of virtual system states**

Observation of virtual system state	Requirement	CIM_ComputerSystem EnabledState Property Value	CIM_AssociatedPower-ManagementService.PowerState Property Value (Optional)
"defined" (See 6.4.1.1)	Mandatory	3 (Disabled)	8 (Off – Soft) 6 (Off – Hard)
"active" (See 6.4.1.2)	Mandatory	2 (Enabled) 4 (Shutting Down) 8 (Deferred) 10 (Starting)	2 (On)
"paused" (Optional) (See 6.4.1.3)	Optional	9 (Quiesce)	3 (Sleep – Light)

Observation of virtual system state	Requirement	CIM_ComputerSystem EnabledState Property Value	CIM_AssociatedPowerManagementService.PowerState Property Value (Optional)
"suspended" (Optional) (See 6.4.1.4)	Optional	6 (Enabled but Offline)	4 (Sleep – Deep) 7 (Hibernate (Off – Soft))
Vendor Defined (Optional) (See 6.4.1.5)	Optional	1 (Other)	1 (Other) or (0x7FFF-0xFFFF)
"unknown" (Optional) (See 6.4.1.6)	Optional	0 (Unknown)	n/a
Unspecified (Values shall not be used by conformant implemen- tations.)	Not supported	5 (Not Applicable) 7 (In Test)	n/a
NOTE Preferred values of the EnabledState property are shown in bold face; other possible values are shown in regular style.			

566 The use of the values in the "CIM_AssociatedPowerManagementService.PowerState Property Value
567 (Optional)" column listed in Table 2 is described in 7.7.1 .

568 NOTE This profile clearly distinguishes between the observation of virtual system state (as defined in this sub-
569 clause) and client state management (as defined in 7.6). In particular with respect to the observation of vir-
570 tual system state no mechanism is specified for determining a supported subset of virtual system states; in-
571 stead any virtual system state as defined by Table 2 is possible. Opposed to that the set of state transitions
572 that may be effected through client state management is modeled in 7.6 through the CIM_EnabledLogical-
573 ElementCapabilities class.

574 7.1.2 CIM_ComputerSystem.RequestedState property

575 The RequestedState property shall be implemented. The RequestedState property shall be used to indi-
576 cate whether the observation of virtual system state transitions is implemented, and if the observation of
577 virtual system state transitions is implemented the property shall indicate ongoing virtual system state
578 transitions.

579 The following provisions apply:

- 580 • If the observation of virtual system state transitions is not implemented, the RequestedState
581 property shall be set to a value of 12 (Not Applicable).
- 582 • If the observation of one or more virtual system state transitions is implemented, the value of
583 the RequestedState property shall be used to facilitate the observation of virtual system state
584 transitions. The following provisions apply:
 - 585 – The RequestedState property shall not have a value of 12 (Not Applicable).
 - 586 – The RequestedState property shall have a value designating the most recently requested
587 state transition according to Table 3. For example, if a virtual system is performing an "Ac-
588 tivate" state transition, then the RequestedState property shall have a value of 2 (Enabled).
 - 589 – If a state transition completes successfully, the value of the EnabledState property shall re-
590 flect the "To" virtual system state as defined by Table 3, using values as defined by Table
591 2. For example, if a virtual system has successfully performed an "activate" state transition,
592 then it shall be in the "active" virtual system state and show a value of 2 (Enabled) for the
593 EnabledState property. The RequestedState property shall maintain the value designating
594 the most recently requested state transition according to Table 3.

595 – If a state transition fails, the value of the EnabledState property shall represent the current
 596 state of the virtual system as defined by Table 2. The RequestedState property shall have
 597 a value of 5 (No Change).

598 – If the implementation is unable to access information about the most recent or pending
 599 state transition the RequestedState property shall have a value of 5 (No Change).

600 NOTE State transitions may be observed even if client state management as described in 7.6 is not implemented.
 601 For example, a state transition might be initiated by means inherent to the virtualization platform, or it might
 602 be triggered during activation of the virtualization platform itself.

603 Table 3 provides the normative mapping of virtual system state transitions to values of the Requested-
 604 State property and the RequestedState parameter.

605 **Table 3 – Observation of virtual system state transitions**

Observation of Virtual System Transition	Requirement	"From" Virtual System State	"To" Virtual System State	RequestedState Property and Parameter Value	RequestPowerStateChange(): Property Value
Observation of state transitions not supported	n/a	n/a	n/a	12 (Not Applicable)	n/a
"define" (Optional) (See 6.4.2.1)	Optional	No CIM_ComputerSystem instance	"Defined"	Not applicable. For definition of virtual systems see the <i>System Virtualization Profile</i> .	
"activate" (Optional) (See 6.4.2.2)	Optional	"Defined" "Paused" "Suspended"	"Active"	2 (Enabled)	2 (On)
"deactivate" (Optional) (See 6.4.2.3)	Optional	"Active" "Paused" "Suspended"	"Defined"	3 (Disabled)	8 (Off – Soft)
"pause" (Optional) (See 6.4.2.4)	Optional	"Active"	"Paused"	9 (Quiesce)	3 (Sleep–Light)
"suspend" (Optional) (See 6.4.2.5)	Optional	"Active" "Paused"	"Suspended"	6 (Offline)	4 (Sleep –Deep)
"shut down" (Optional) (See 6.4.2.6)	Optional	"Active" "Paused" "Suspended"	"Defined"	4 (Shut Down)	8 (Off – Soft)
"reboot" (Optional) (See 6.4.2.7)	Optional	"Active" "Paused" "Suspended"	"Active"	10 (Reboot)	5 (Power Cycle (Off – Soft))
"reset" (Optional) (See 6.4.2.8)	Optional	"Active" "Paused" "Suspended"	"Active"	11 (Reset)	9 (Power Cycle (Off – Hard))
Information about recent or pending state transitions not available	Optional	n/a	n/a	5 (No Change)	n/a

NOTE Preferred values of the RequestedState property are shown in bold face; other possible values are shown in regular style.

606 NOTE This profile clearly distinguishes between the observation of virtual system state transitions (as defined in
 607 this subclause) and client state management (as defined in 7.6). In particular, with respect to the observation
 608 of virtual system state transitions, no mechanism is specified for determining a supported subset of virtual
 609 system state transitions; instead any virtual system state transition as defined by Table 3 is possible. Op-
 610 posed to that, the set of state transitions that may be effected through client state management is modeled
 611 in 7.6 through the CIM_EnabledLogicalElementCapabilities class.

612 7.2 Virtual resource

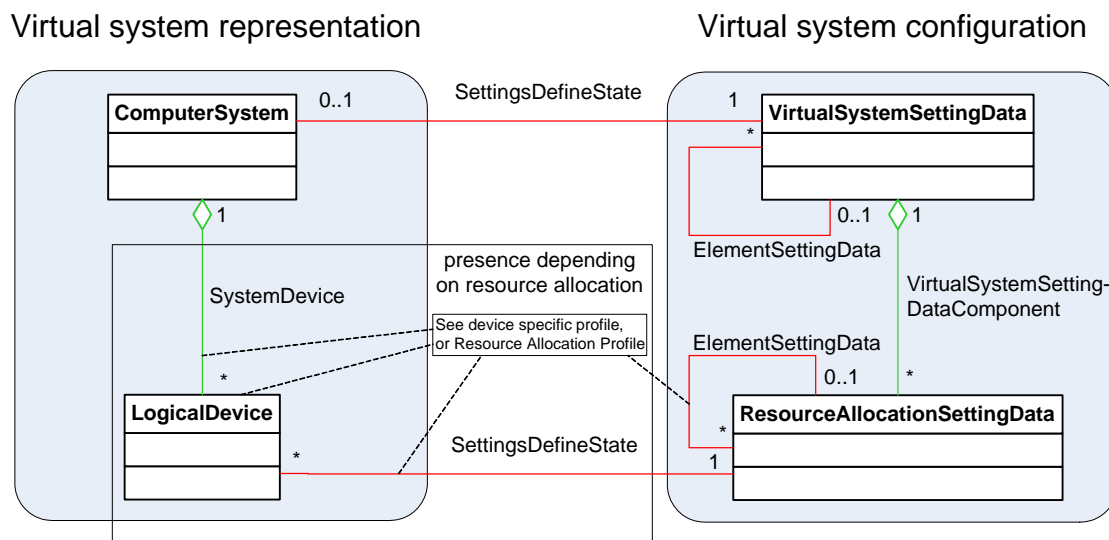
613 Resources in system representations are specified by resource-type-specific profiles such as [DSP1052](#) or
614 [DSP1026](#). These resource-type-specific profiles may be implemented for one or more types of virtual re-
615 sources, omitting optional elements that model physical aspects.

616 Most resource-type-specific profiles specify that logical resources are represented by instances of the
617 CIM_LogicalDevice class, and are aggregated into a virtual system representation using the
618 CIM_SystemDevice association. This profile specifies the use of virtual system configurations for the ex-
619 tension of virtual system representations with virtualization-specific properties.

620 7.3 Virtual system configuration

621 7.3.1 Structure

622 A virtual system configuration shall consist of one instance of the CIM_VirtualSystemSettingData class as
623 the top-level object, and zero or more instances of the CIM_ResourceAllocationSettingData class. The
624 CIM_VirtualSystemSettingDataComponent association shall be used to associate the instance of the
625 CIM_VirtualSystemSettingData class with aggregated instances of the CIM_ResourceAllocationSetting-
626 Data class (see Figure 4).



627

628 **Figure 4 – Virtual system representation and virtual system configuration**

629 7.3.2 The "state" virtual system configuration

630 There shall be exactly one "state" virtual system configuration representing the virtualization specific state
631 of the virtual system. Elements of the "state" virtual system configuration add virtualization-specific prop-
632 erties to related elements in the virtual system representation. Elements of the "state" virtual system con-
633 figuration shall have the same lifecycle as their counterparts in the virtual system representation.

634 The top-level instance of the CIM_VirtualSystemSettingData class in the "state" virtual system configura-
635 tion shall be associated to the instance of the CIM_ComputerSystem class that represents the virtual sys-
636 tem through an instance of the CIM_SettingsDefineState association.

637 NOTE 1 See A.3 for a description of how the presence of instances of CIM classes and of property values within
638 instances may depend on the virtual system state.

639 NOTE 2 If [DSP1041](#) is implemented for a particular resource type, it may require additional instances of the
640 CIM_SettingsDefineState association connecting instances of the CIM_ResourceAllocationSettingData class in the

641 "State" virtual system configuration to related instances of the CIM_LogicalDevice class in the virtual system repre-
642 sentation.

643 **7.3.3 The "defined" virtual system configuration**

644 There shall exactly be one "defined" virtual system configuration representing the virtual system definition.
645 The top-level instance of the CIM_VirtualSystemSettingData class in the "defined" virtual system configura-
646 tion shall be associated to the top-level instance of the CIM_VirtualSystemSettingData class in the
647 "state" virtual system configuration through the CIM_ElementSettingData association with the IsDefault
648 property set to a value of 1 (Is Default).

649 The "Defined" virtual system configuration shall be present at all times regardless of the virtual system
650 state.

651 **NOTE** An implementation may coincide the "defined" virtual system configuration and the "state" vir-
652 tual system configuration; see 7.3.4 .

653 If [DSP1041](#) is implemented for a particular resource type, it may require additional instances of the
654 CIM_ElementSettingData association to connect instances of the CIM_ResourceAllocationSettingData
655 class in the "State" virtual system configuration with their counterparts in the "defined" virtual system con-
656 figuration. The presence of these association instances is not required or defined by this profile
657 (DSP1057). However, this profile requires that any instances of the CIM_ElementSettingData association
658 that are required by [DSP1041](#) shall have an attribute set that is consistent with the attribute set of the in-
659 stance of the CIM_ElementSettingData association that associates the top-level instances of the
660 CIM_VirtualSystemSettingData class.

661 **7.3.4 Implementation approaches for "state" and "defined" virtual system configura-** 662 **tion**

663 Implementations are not required to support separate virtual system configurations for the representation
664 of virtual system definition and virtual system instance: Implementations may apply either a dual-configu-
665 ration implementation approach (see 7.3.4.1) or a single-configuration implementation approach (see
666 7.3.4.2); an implementation shall not mix the two implementation approaches. For a detailed instance-
667 based description, see Annex B.

668 **7.3.4.1 Dual-configuration implementation approach**

669 This approach is applicable for implementations that support separate configurations for the representa-
670 tion of the virtual system definition and the virtual system instance. This approach allows the modeling of
671 divergent modifications on definition and instance.

672 With this dual-configuration approach, the "defined" and the "state" virtual system configurations shall be
673 composed of unique instances of the CIM_VirtualSystemSettingData class and the CIM_ResourceAlloca-
674 tionSettingData class in each configuration.

675 For the top-level instance of the CIM_VirtualSystemSettingData class in the "state" virtual system configu-
676 ration the following provisions apply:

- 677 • It shall be associated to the instance of the CIM_ComputerSystem class in the virtual system
678 representation through an instance of the CIM_SettingsDefineState association
- 679 • It shall be associated to its counterpart in the "defined" virtual system configuration through an
680 instance of the CIM_ElementSettingData association where
 - 681 – the value of the IsDefault property shall be set to according to 7.3.11
 - 682 – the value of the IsNext property shall be set to according to 7.3.12
- 683 • It shall be associated to any instance of the CIM_ResourceAllocationSettingData class that is
684 part of the "state" virtual system configuration via an instance of the CIM_VirtualSystemSetting-
685 DataComponent association

686 [DSP1041](#) or profiles based on [DSP1041](#) may require compliance to similar conditions with respect to in-
687 stances of the CIM_ResourceAllocationSettingData class and the CIM_LogicalDevice class. If resources
688 are allocated or de-allocated, respective instances of the CIM_ResourceAllocationSettingData class shall
689 be added to or removed from the "State" virtual system configuration along with the associations referring
690 to them.

691 NOTE The values of the properties within the instances of the CIM_ElementSettingData association depend on the
692 virtual system state and/or on the resource allocation state.

693 **7.3.4.2 Single-configuration implementation approach**

694 This approach is applicable for implementations that do not support separate configurations for the repre-
695 sentation of the virtual system definition and virtual system instance.

696 With this approach, instances of the CIM_VirtualSystemSettingData class and the CIM_ResourceAlloca-
697 tionSettingData class are shared for both the "defined" virtual system configuration and the "state" virtual
698 system configuration.

699 For the top-level instance of the CIM_VirtualSystemSettingData class in the single virtual system configu-
700 ration the following provisions apply:

- 701 • It shall be associated to the instance of the CIM_ComputerSystem class in the virtual system
702 representation through an instance of the CIM_SettingsDefineState association
- 703 • It shall be associated to itself through an instance of the CIM_ElementSettingData association
704 where
 - 705 – the value of the IsDefault property shall be set to according to 7.3.11
 - 706 – the value of the IsNext property shall be set to according to 7.3.12
- 707 • It shall be associated to any instance of the CIM_ResourceAllocationSettingData class that is
708 part of the single virtual system configuration via an instance of the CIM_VirtualSystemSetting-
709 DataComponent association

710 [DSP1041](#) or profiles based on [DSP1041](#) may require compliance to similar conditions with respect to in-
711 stances of the CIM_ResourceAllocationSettingData class and the CIM_LogicalDevice class, such that as
712 resources are allocated or de-allocated, respective instances of the CIM_SettingsDefineState association
713 and the CIM_ElementSettingData association are required to be added to or removed from instances of
714 the CIM_ResourceAllocationSettingData class.

715 NOTE The values of the properties within the instances of the CIM_ElementSettingData association depend on the
716 virtual system state and/or on the resource allocation state.

717 **7.3.5 Other types of virtual system configurations**

718 Additional virtual system configurations may be associated to the "state" virtual system configuration
719 through the CIM_ElementSettingData association. For details about the "next" configuration (the configu-
720 ration that will be used during the next activation of the virtual system), see 7.3.12.

721 **7.3.6 CIM_VirtualSystemSettingData.Caption property**

722 The implementation of the Caption property is optional.

723 If the Caption property is implemented, the provisions in this subclause apply.

724 If the Caption property is implemented for the CIM_ComputerSystem class, the value of the Caption
725 property in the instance of the CIM_VirtualSystemSettingData class in the "state" virtual system configura-
726 tion of a virtual system shall be identical to the value of the Caption property in the instance of the
727 CIM_ComputerSystem class representing the virtual system.

7.3.7 CIM_VirtualSystemSettingData.Description property

729 The implementation of the Description property is optional.

730 If the Description property is implemented, the provisions in this subclause apply.

731 The value of the Description property in the instance of the CIM_VirtualSystemSettingData class in the
732 "state" virtual system configuration of a virtual system shall be identical to the value of the description
733 property in the instance of the CIM_ComputerSystem class representing the virtual system.

7.3.8 CIM_VirtualSystemSettingData.ElementName property

735 The value of the ElementName property reflects a name for the virtual system configuration assigned by
736 an end-user or administrator.

737 If the ElementName property is implemented for the CIM_ComputerSystem class, the value of the Ele-
738 mentName property in the instance of the CIM_VirtualSystemSettingData class in the "state" virtual sys-
739 tem configuration of a virtual system shall be identical to the value of the ElementName property in the
740 instance of the CIM_ComputerSystem class representing the virtual system.

7.3.9 CIM_VirtualSystemSettingData.VirtualSystemIdentifier property

742 The implementation of the VirtualSystemIdentifier property is optional.

743 If the VirtualSystemIdentifier property is implemented, the provisions in this subclause apply.

744 The value of the VirtualSystemIdentifier property reflects a name for the virtual system assigned by the
745 implementation during virtual system creation. A typical example is a human-readable user ID.

746 The value of the VirtualSystemIdentifier property shall be unique for each instance of the
747 CIM_VirtualSystemSettingData class that represents a virtual system (or its definition) within the scope of
748 a host system.

7.3.10 CIM_VirtualSystemSettingData.VirtualSystemType property

750 The implementation of the VirtualSystemType property is optional.

751 If the VirtualSystemType property is implemented, the provisions in this subclause apply.

752 The value of the VirtualSystemType property reflects a specific type for the virtual system.

753 NOTE The VirtualSystemType property is defined primarily for programmatic use rather than for conveying a virtual
754 system type to end-users.

755 Restrictive conditions may be implied by a virtual system type; these conditions are implementation-
756 dependent and are not specified in this profile. For example, a system type of "OS1 Container" might be
757 defined indicating that a virtual system of that type is used to run an operating system named "OS1". An-
758 other example might be a system type of "CommunicationController", indicating that the virtual system
759 runs special-purpose software enabling it to act as a communication server.

760 The virtual system type may change during the lifetime of the virtual system. For example, a change may
761 be effected through the use of inherent management facilities available with the virtualization platform or
762 through facilities defined by [DSP1042](#) that enable a client to modify virtual system configurations.

7.3.11 CIM_ElementSettingData.IsDefault property

764 The IsDefault property shall be implemented. Each top-level CIM_VirtualSystemSettingData instance in a
765 "state" virtual system configuration and the top-level CIM_VirtualSystemSettingData instance in the re-
766 lated "defined" virtual system configuration shall be associated through an instance of the CIM_Elements-
767 SettingData association. The value of the IsDefault property shall be used to designate the "defined" vir-
768 tual system configuration among all configurations associated with the "state" virtual system configuration.

769 The value of the IsDefault property shall be set as follows:

- 770 • The IsDefault property shall have a value of 1 (Is Default) if the related virtual system configura-
771 tion is the "defined" virtual system configuration.
- 772 • In all other cases, the IsDefault property shall have a value of 2 (Is Not Default).
- 773 • The IsDefault property shall not have a value of 0 (Unknown).
- 774 • In the set of all virtual system configurations that are associated to a top-level instance of the
775 CIM_VirtualSystemSettingData class in a "state" virtual system configuration exactly one con-
776 figuration shall be referenced by an instance of the CIM_ElementSettingData association with a
777 value of 1 (Is Default) for the IsDefault property.

778 The "defined" virtual system configuration is the fall-back default that shall be used for virtual system acti-
779 vation if no other configuration is marked through the IsNext property.

780 **7.3.12 CIM_ElementSettingData.IsNext property**

781 The implementation of the IsNext property is optional.

782 If the IsNext property is implemented, the provisions in this subclause apply.

783 The IsNext property may be used to designate the "next" virtual system configuration. The "next" virtual
784 system configuration is the virtual system configuration that will be used for the next activation of the vir-
785 tual system; if no configuration is marked as the "next" virtual system configuration, the "default" virtual
786 system configuration is used for the next activation.

787 If the IsNext property is implemented, the value of the IsNext instances of the CIM_ElementSettingData
788 association associating a top-level instance of the CIM_VirtualSystemSettingData class in a "state" virtual
789 system configuration and a top-level instance of the CIM_VirtualSystemSettingData class in a related vir-
790 tual system configuration shall be set as follows:

- 791 • The IsNext property shall have one of the following values:
 - 792 – a value of 0 (Unknown) if it is not known whether the referenced virtual system configura-
793 tion will be used for the next activation
 - 794 – a value of 1 (Is Next) if the referenced virtual system configuration is established to be
795 used for subsequent activations of the virtual system
 - 796 – a value of 3 (Is Next For Single Use) if the referenced virtual system configuration is estab-
797 lished to be used for just the next activation of the virtual system in preference of the de-
798 fault and or the persistently established next configuration.
 - 799 – In all other cases the IsNext property shall have a value of 2 (Is Not Next). In this case the
800 "default" virtual system configuration is used for the next virtual system activation.
- 801 • In the set of all virtual system configurations that are associated with a top-level instance of the
802 CIM_VirtualSystemSettingData class in a "state" virtual system configuration, there shall be
 - 803 – at most one configuration that is referenced by an instance of the CIM_ElementSettingData
804 association with a value of 1 (Is Next)
 - 805 – at most one configuration that is referenced by an instance of the CIM_ElementSettingData
806 association with a value of 3 (Is Next For Single Use) for the IsNext property. This configu-
807 ration shall be given preference over one that is designated with a value of 1 (Is Next).

808 7.4 Profile registration

809 7.4.1 This profile

810 The implementation of this profile shall be indicated by an instance of the CIM_RegisteredProfile class in
 811 the CIM Interop namespace. Each instance of the CIM_ComputerSystem class that represents a virtual
 812 system manageable through this profile shall be a central instance of this profile by associating it to the
 813 instance of the CIM_RegisteredProfile class through an instance of the CIM_ElementConformsToProfile
 814 association.

815 7.4.2 Scoped profiles

816 For scoped profiles the following conditions shall be met:

- 817 • The instance of the CIM_RegisteredProfile class that represents the implementation of this pro-
 818 file and instances of the CIM_RegisteredProfile class that represent an implementation of the
 819 scoped profile shall be associated through instances of the CIM_ReferencedProfile association.
- 820 • One of the following conditions shall be met:
 - 821 a) Instances of the CIM_ElementConformsToProfile association shall associate any central
 822 instance of the scoped profile that is associated to the central instance of this profile
 823 through the CIM_SystemDevice association, and the instance of the CIM_RegisteredPro-
 824 file class that represents an implementation of the scoped profile.
 - 825 b) No instances of the CIM_ElementConformsToProfile association shall associate any cen-
 826 tral instance of the scoped profile that is associated to the central instance of this profile
 827 through the CIM_SystemDevice association, and the instance of the CIM_RegisteredPro-
 828 file class that represents an implementation of the scoped profile.

829 7.5 Capabilities

830 7.5.1.1 CIM_EnabledLogicalElementCapabilities.RequestedStatesSupported property

831 The RequestedStatesSupported property shall not have a value of NULL. An empty array indicates that
 832 client state management is not implemented. A non-empty array indicates that client state management is
 833 implemented for a particular virtual system and lists the supported state transitions. The list of supported
 834 state transitions depends on the current virtual system state. The subset of state transitions that are sup-
 835 ported for each state is implementation dependent. The maximal set is defined by Table 3.

836 NOTE The value of this property is volatile. It may change at any time, including the cases where an empty list
 837 changes to a non-empty list and vice versa.

838 7.6 Client state management

839 The implementation of client state management is conditional.

840 Condition: The CIM_ComputerSystem instance that represents a virtual system is associated through the
 841 CIM_ElementCapabilities association to an instance of the CIM_EnabledLogicalElementCapabilities
 842 class, and in that instance the value the RequestedStatesSupported property is a non-empty array.

843 If client state management is implemented, the provisions in this subclause apply.

844 Client state management comprises the facilities provided by the implementation that enable a client to
 845 request virtual system state transitions.

846 If client state management is implemented, an implementation shall do all of the following:

- 847 • implement the CIM_EnabledLogicalElementCapabilities class according to 7.5.1.1 to indicate
 848 the availability of client state management support, and the set of state transitions that are ap-
 849 plicable

- 850 • implement method RequestStateChange()
- 851 • if it implements [DSP1027](#) for virtual systems, implement the RequestPowerStateChange()
- 852 method

853 **7.7 Power state management**

854 The implementation of power state management is optional.

855 If power state management is implemented, the provisions in this subclause apply.

856 The implementation of power state management requires the implementation of [DSP1027](#). [DSP1027](#)

857 specifies

- 858 • how to indicate that [DSP1027](#) is implemented
- 859 • how to implement the CIM_PowerManagementService class and the CIM_Associated-
- 860 PowerManagementService association

861 If the observation of power states is implemented as specified by [DSP1027](#), then the observation of vir-

862 tual system states as defined in 7.1.1 and the observation of virtual system state transitions as defined in

863 7.1.2 shall also be implemented. If power state management is implemented as specified by [DSP1027](#),

864 then client state management as specified in 8.1.1 shall also be implemented.

865 **NOTE** The implementation of [DSP1027](#) in the context of virtual systems is intended to support clients that use fa-

866 cilities specified by [DSP1027](#) in preference of facilities specified in this profile (DSP1057). For example,

867 such clients may use the CIM_AssociatedPowerManagementService.PowerState property in favor of the

868 CIM_ComputerSystem.EnabledState property to determine the virtual system state, or may use the

869 CIM_PowerManagementService.RequestPowerStateChange() method in favor of the CIM_EnabledLogical-

870 Element.RequestStateChange() method to effect virtual system state transitions.

871 **7.7.1 CIM_AssociatedPowerManagementService.PowerState property**

872 The implementation of the PowerState property is conditional.

873 Condition: All of the following

- 874 • Client state management is implemented (see 7.5.1.1)
- 875 • [DSP1027](#) is implemented.

876 If the PowerState property is implemented, the provisions in this subclause apply.

877 The CIM_AssociatedPowerManagementService association shall be used to convey the virtual system

878 state in addition to the CIM_ComputerSystem.EnabledState property. In this case, the PowerState prop-

879 erty shall contain a value that corresponds to the virtual system state as defined in Table 2. For example,

880 if the virtual system state is "active", then the PowerState property shall have a value of 2 (On).

881 A client preferring to use mechanisms defined by [DSP1027](#) may translate the value of the PowerState

882 property of an instance of the CIM_AssociatedPowerManagementService association that is referring to

883 an instance of the CIM_ComputerSystem class representing a virtual system by translating that value

884 according to Table 2. For example, if the PowerState property has a value of 2 (On), then a client shall

885 conclude that the virtual system state is "active".

886 **8 Methods**

887 This clause details the requirements for supporting intrinsic CIM operations and extrinsic methods for the

888 CIM elements defined by this profile.

889 The CIM Schema descriptions for any referenced method and its parameters apply.

890 **8.1 Extrinsic methods**

891 **8.1.1 CIM_ComputerSystem.RequestStateChange() method**

892 The implementation of the RequestStateChange() method is conditional.

893 Condition: Client state management is implemented (see 7.6).

894 If the RequestStateChange() method is implemented, the provisions in this subclause apply.

895 Detailed requirements for the CIM_ComputerSystem.RequestStateChange() method are specified in
896 Table 4.

897 **Table 4 – CIM_ComputerSystem.RequestStateChange() method: Parameters**

Qualifiers	Name	Type	Description/Values
IN	RequestedState	uint16	The requested virtual system state transition according to the transformation defined in Table 3.
OUT	Job	CIM_ConcreteJob REF	A reference to the job that performs the task (NULL if the task is completed on return).
IN	TimeoutPeriod	datetime	A timeout period that specifies the maximum amount of time that the client expects the transition to the new state to take.

898 For return code values, see the CIM Schema description of this method in the CIM_EnabledLogical-
899 Element class.

900 No standard messages are defined.

901 **8.1.2 CIM_PowerManagementService.RequestPowerStateChange() method**

902 The implementation of the RequestPowerStateChange() method is conditional.

903 Condition: All of the following

- 904 • Client state management is implemented (see 7.5.1.1)
- 905 • [DSP1027](#) is implemented.

906 If the RequestPowerStateChange() method is implemented, the provisions in this subclause apply.

907 The RequestPowerStateChange() method shall enable the request of virtual system state transitions
908 through this alternative method. Detailed requirements for the CIM_PowerManagementService.Request-
909 StateChange() method are specified in Table 5.

910 **Table 5 – CIM_PowerManagementService.RequestPowerStateChange() method: Parameters**

Qualifiers	Name	Type	Description/Values
IN	PowerState	uint16	See 8.1.2.1 .
IN	ManagedElement	CIM_ComputerSystem REF	See 8.1.2.2 .
IN	Time	datetime	See 8.1.2.3 .
OUT	Job	CIM_ConcreteJob REF	A reference to the job that performs the task (null if the task is completed on return). For details, see the CIM Schema description of this parameter.

911 For return code values, see the CIM Schema description of this method in the CIM_PowerManagement-
912 Service class.

913 No standard messages are defined.

914 **8.1.2.1 PowerState parameter**

915 The PowerState parameter encodes the requested new virtual system state.

916 The translation defined by Table 3 shall be used to interpret values of the PowerState parameter of the
917 CIM_PowerManagementService.RequestPowerStateChange() method as a request for a virtual system
918 state transition. For example, if value "On" is specified on a particular power state change request for a
919 virtual system, then an "activate" state transition shall be performed.

920 **8.1.2.2 ManagedElement parameter**

921 The value of the ManagedElement parameter shall be used to identify the virtual system to which the op-
922 eration applies.

923 **8.1.2.3 Time parameter**

924 The implementation of the Time parameter is optional.

925 If the Time parameter is implemented, the provisions in this subclause apply.

926 The Time parameter shall indicate the point in time when the power state shall be set.

927 **8.2 Profile conventions for operations**

928 The default list of operations for all classes is:

929 GetInstance()

930 EnumerateInstances()

931 EnumerateInstanceNames()

932 For classes that are referenced by an association, the default list also includes

933 Associators()

934 AssociatorNames()

935 References()

936 ReferenceNames()

937 **8.2.1 CIM_ComputerSystem**

938 All operations in the default list in 8.2 shall be implemented as defined in [DSP0200](#).

939 NOTE Related profiles may define additional requirements on operations for the profile class.

940 **8.2.2 CIM_ConcreteJob**

941 All operations in the default list in 8.2 shall be implemented as defined in [DSP0200](#).

942 NOTE Related profiles may define additional requirements on operations for the profile class.

943 **8.2.3 CIM_ElementSettingData**

944 All operations in the default list in 8.2 shall be implemented as defined in [DSP0200](#).

945 NOTE Related profiles may define additional requirements on operations for the profile class.

946 **8.2.4 CIM_EnabledLogicalElementCapabilities**

947 All operations in the default list in 8.2 shall be implemented as defined in [DSP0200](#).

948 NOTE Related profiles may define additional requirements on operations for the profile class.

949 **8.2.5 CIM_ReferencedProfile**

950 All operations in the default list in 8.2 shall be implemented as defined in [DSP0200](#).

951 NOTE Related profiles may define additional requirements on operations for the profile class.

952 **8.2.6 CIM_RegisteredProfile**

953 All operations in the default list in 8.2 shall be implemented as defined in [DSP0200](#).

954 NOTE Related profiles may define additional requirements on operations for the profile class.

955 **8.2.7 CIM_VirtualSystemSettingData**

956 All operations in the default list in 8.2 shall be implemented as defined in [DSP0200](#).

957 NOTE Related profiles may define additional requirements on operations for the profile class.

958 **8.2.8 CIM_VirtualSystemSettingDataComponent**

959 All operations in the default list in 8.2 shall be implemented as defined in [DSP0200](#).

960 NOTE Related profiles may define additional requirements on operations for the profile class.

961 **9 Use-cases**

962 The following use-cases and object diagrams illustrate use of this profile. They are for informational pur-
963 poses only and do not introduce behavioral requirements for implementations of the profile.

964 **9.1 Virtual system detection and inspection**

965 This set of use cases describes how a client can

- 966 • discover virtual systems
- 967 • determine the state and properties of a virtual system
- 968 • determine the "defined" virtual system configuration
- 969 • determine the virtual system structure
- 970 • determine resource type support
- 971 • detect and inspect the boot configuration for the virtual system

972 **9.1.1 Discover conformant virtual systems using SLP**

973 This use case describes how to locate instances of the CIM_ComputerSystem class that represent virtual
974 systems that are central instances of this profile. This is a two-step process:

- 975 1) The service location protocol (SLP) is used to locate WBEM services where this profile is im-
 976 plemented. A WBEM service using SLP facilities provides information about itself to SLP in form
 977 of an SLP service template. The service template may contain information about the set of pro-
 978 files that is implemented at the WBEM service.
- 979 2) Normal CIM enumeration and association resolution is used to find instances of the CIM_Com-
 980 puterSystem class that represent central instances of this profile.

981 **Assumption:** This profile is registered at least one WBEM service that maintains a registration with a
 982 SLP Directory Agent; the registration included information about registered profiles. The client is able to
 983 make SLP calls and invoke intrinsic CIM operations.

984 A client can locate instances of the CIM_ComputerSystem class that represent virtual systems that are
 985 central instances of this profile as follows:

- 986 1) The client invokes the SLPFindSrvs() SLP function:
- 987 – The value of the srvtype parameter is set to "service:wbem"
 - 988 – The value of the scopelist parameter is set to "default"
 - 989 – The value of the filter parameter is set to "(RegisteredProfilesSupported=DMTF:Virtual
 990 System)"

991 The result is a list of URLs that identify WBEM services where this profile is implemented.

- 992 2) The client contacts each of the WBEM services and enumerates or queries the CIM_Regis-
 993 teredProfile class.
- 994 • As input, the client needs to use the address information of one server obtained in step 1)
 995 and issue the intrinsic EnumerateInstanceNames() CIM operation on the CIM_Registered-
 996 Profile class. Alternatively, the client may issue the intrinsic ExecuteQuery CIM operation
 997 and specify a where clause that, for example, limits the value ranges for the Registered-
 998 Name and RegisteredVersion properties of the CIM_RegisteredProfile class.
 - 999 • As a result, the client receives a list of references to instances of the CIM_RegisteredPro-
 1000 file class that represent implementations of this profile at the intended target location. On a
 1001 query operation this list already is limited according to the input selection criteria.
- 1002 3) The client selects one reference and resolves the CIM_ElementConformsToProfile association
 1003 from the instance of the CIM_RegisteredProfile class to instances of the CIM_ComputerSystem
 1004 class.
- 1005 • As input, the client needs to provide the reference to an instance of the CIM_Registered-
 1006 Profile class that was selected from the result set obtained in step 2.
 - 1007 • As a result, the client receives a list of references referencing instances of the CIM_
 1008 ComputerSystem class that represents virtual systems.

1009 **Result:** The result is that the client knows a set of references referencing instances of the CIM_Compu-
 1010 terSystem class that represent virtual systems that are central instances of this profile.

1011 9.1.2 Determine a virtual system's state and other properties

1012 **Assumption:** The client has a reference referring to an instance of the CIM_ComputerSystem class that
 1013 represents a virtual system that is a central instance of this profile.

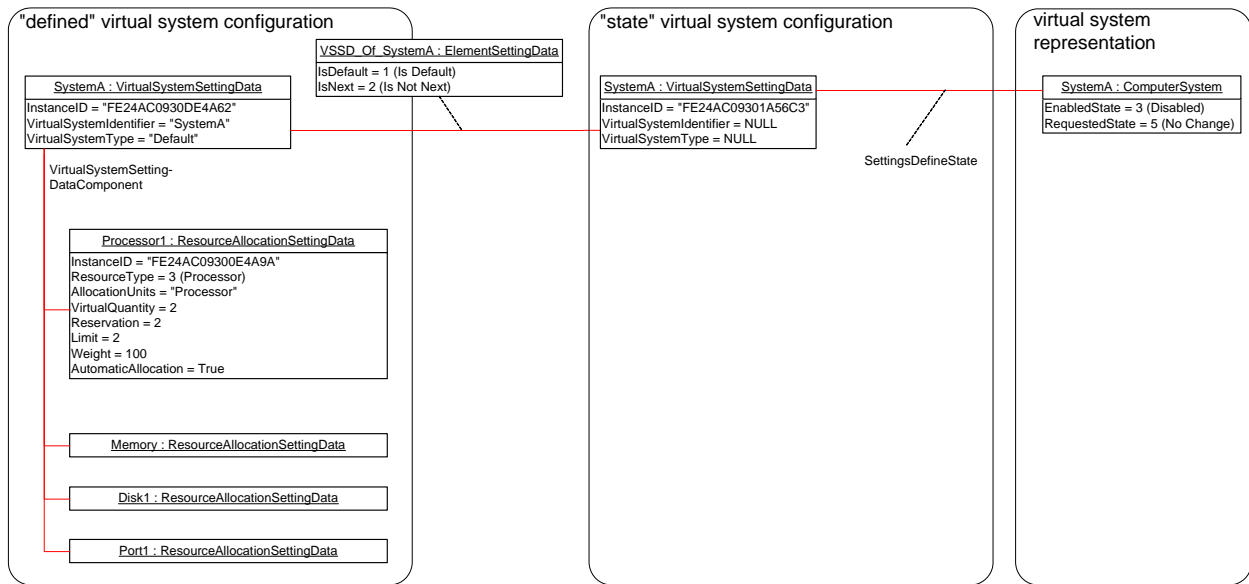
1014 The client can determine the virtual system's state and other properties as follows:

- 1015 1) The client calls the intrinsic GetInstance() CIM operation with the InstanceName parameter ref-
 1016 erencing the instance of the CIM_ComputerSystem class that represents the virtual system as
 1017 the input parameter. As a result the client receives an instance of the CIM_ComputerSystem
 1018 class that describes the virtual system.
- 1019 2) The client uses the value of the EnabledState property to determine the virtual system state ac-
 1020 cording to the translation rules specified in 7.1.1.

1021 **Result:** The client knows the property set defined by the CIM_ComputerSystem class describing the af-
 1022 fected virtual system, in particular the virtual system state. Many virtual system properties and in particu-
 1023 lar the virtual system state may change any time; consequently, the result only describes the virtual sys-
 1024 tem at the moment it was provided by the instrumentation.

1025 **9.1.3 Determine the "defined" virtual system configuration**

1026 **Assumption:** The client has a reference referring to an instance of the CIM_ComputerSystem class that
 1027 represents a virtual system that is a central instance of this profile. The virtual system is assumed to be
 1028 configured as shown in Figure 5 with the "Virtual system configuration ("defined")" configuration. In this
 1029 example the implementation applies the dual-configuration implementation approach (see 7.3.4.1) as de-
 1030 scribed in Annex B.



1031

1032 **Figure 5 – Sample virtual system configuration**

1033 The client can determine the "defined" virtual system configuration as follows:

- 1034 1) The client resolves the CIM_SettingsDefineState association from the instance of the
 1035 CIM_ComputerSystem class representing the virtual system to the top-level instance of the
 1036 CIM_VirtualSystemSettingData class in the "state" Virtual System configuration.
- 1037 2) The client resolves the CIM_ElementSettingData association from the "state" instance of the
 1038 CIM_VirtualSystemSettingData class that represents the virtual aspects of the virtual system to
 1039 instances of the CIM_VirtualSystemSettingData class with the constraint that the CIM_Element-
 1040 SettingData.IsDefault property has a value of 2 (IsDefault). The result is a reference referring to
 1041 an instance of the CIM_VirtualSystemSettingData class that represents the top-level object of
 1042 the desired virtual system configuration.
- 1043 3) The client obtains the referenced instance of the CIM_VirtualSystemSettingData class using the
 1044 intrinsic getInstance() CIM operation and analyzes its properties. For example, the client might
 1045 analyze the VirtualSystemIdentifier property, which reflects the (end-user interpretable) name
 1046 used for the virtual system ("SystemA" in Figure 5), or the VirtualSystemType property, which
 1047 reflects a particular virtual system type that the virtualization platform assigned for the respec-
 1048 tive virtual system ("Default" in Figure 5). Note that the InstanceID property contains an opaque
 1049 ID for the instance; the structure of InstanceID values is implementation dependent and not
 1050 known to clients.

- 1051 4) The client resolves the CIM_VirtualSystemSettingDataComponent association from the instance
 1052 of the CIM_VirtualSystemSettingData class to instances of the CIM_ResourceAllocationSetting-
 1053 Data class.
- 1054 5) The client obtains instances of the CIM_ResourceAllocationSettingData class using the intrinsic
 1055 getInstance() CIM operation and analyzes properties of these instances. For example, the cli-
 1056 ent might analyze the Reservation property. The Reservation property reflects the amount of
 1057 host resource that is allocated for the virtual resource while the virtual system is instantiated.

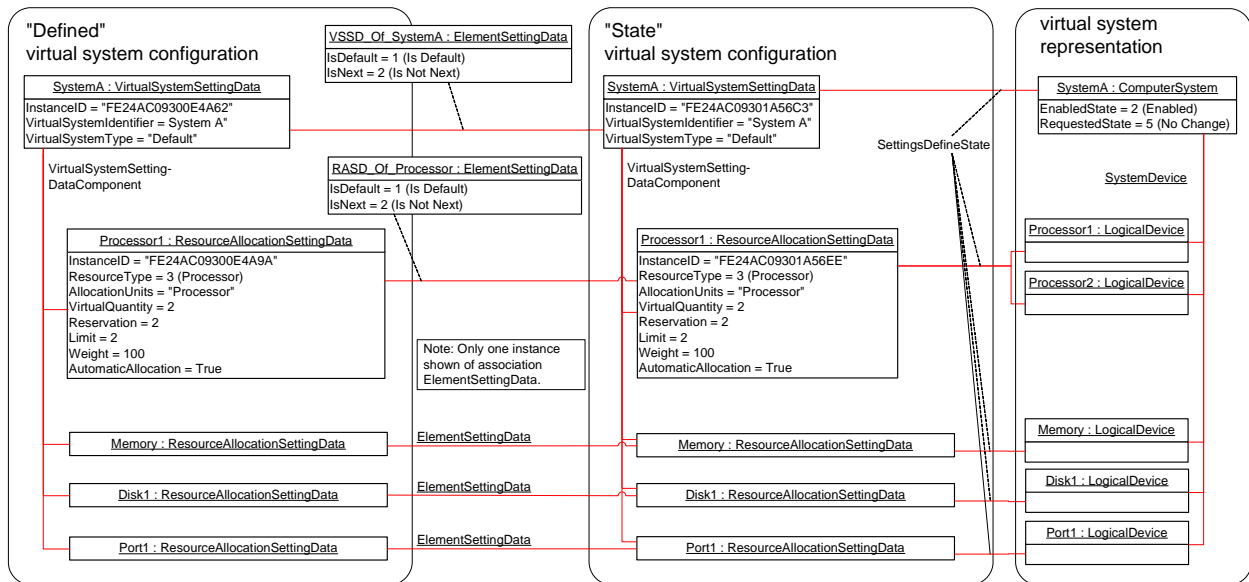
1058 **Result:** The client knows the virtual system configuration in terms of one instance of the CIM_Virtual-
 1059 SystemSettingData class and a set of aggregated instances of the CIM_ResourceAllocationSettingData
 1060 class.

1061 **9.1.4 Determine the virtual system structure**

1062 **Assumption:** The client has a reference referring to an instance of the CIM_ComputerSystem class that
 1063 represents a virtual system that is a central instance of this profile.

- 1064 • The virtual system configuration is assumed to be the same as for use case described in 9.1.3.
- 1065 • The virtual system is assumed to be in the "active" state.
- 1066 • The virtual system is assumed to be structured as shown in Figure 6.
- 1067 • The set of attributes for each logical resource is not shown; this set of attributes depends on the
 1068 type of logical resource and may be specified in the context of respective resource-type-specific
 1069 profiles.

1070 To avoid cluttering the diagram, an instance of the CIM_ElementSettingData association between the
 1071 "defined" and the "state" instance of the CIM_ResourceAllocationSettingData class is shown for proces-
 1072 sor configurations only.



1073

1074 **Figure 6 – Sample virtual system in "active" state**

1075 A client can determine the virtual system structure as follows:

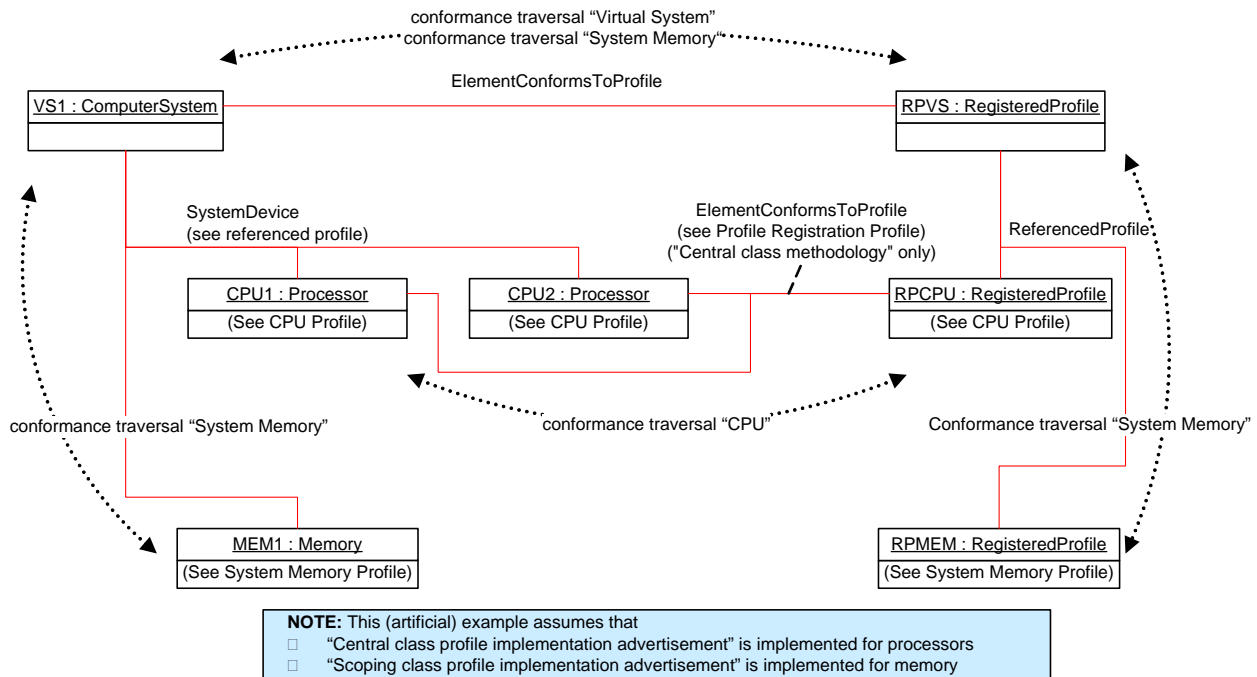
- 1076 1) The client may apply the use case described in 9.1.2 to obtain state information and other prop-
1077 erties of the CIM_ComputerSystem instance that represents the virtual system.
- 1078 2) The client may apply the use case described in 9.1.3 to obtain information about the virtual sys-
1079 tem configuration.
- 1080 3) The client resolves the CIM_SystemDevice association from the instance of the CIM_Computer-
1081 System class that represents the virtual system to instances of the CIM_LogicalDevice class.
- 1082 4) The client obtains instances of the CIM_LogicalDevice class that were returned in step 3) and
1083 analyzes properties of interest.

1084 **Result:** The client knows the virtual system structure as expressed through the virtual system configura-
1085 tions ("defined" and "state") and through the set of objects representing the virtual system and its compo-
1086 nents.

1087 **9.1.5 Determine resource type support**

1088 This subset of use cases describes how to determine whether implementations of resource-type-specific
1089 profiles are present for logical devices in scope of a virtual system. Examples are the [DSP1022](#) for the
1090 management of virtual processors with the CIM_Processor class as the central class, or [DSP1026](#) for the
1091 management of virtual memory with the CIM_Memory class as the central class.

1092 [DSP1033](#) defines how an implementation of a profile advertises conformance to the profile. For example,
1093 Figure 7 shows an instance of the CIM_ComputerSystem class named VS1 that is associated to an in-
1094 stance of the CIM_RegisteredProfile class named RPVS.



1095

1096 **Figure 7 – Instance diagram: Profile conformance of scoped resources**

1097 If profiles addressing the management of scoped resources are implemented, then [DSP1033](#) specifies to
1098 implement either the "central class profile implementation advertisement methodology" or the "scoped
1099 class profile implementation advertisement methodology".

1100 With the "central class profile implementation advertisement methodology" the approach is straight for-
1101 ward: Any central instance of a profile is associated with the respective instance of the
1102 CIM_RegisteredProfile class through the CIM_ElementConformsToProfile association.

1103 With the "Scoped Class Profile Implementation Advertisement Methodology" the CIM_ElementConforms-
1104 ToProfile association is not implemented for scoped profiles and resources; instead, conformance of
1105 scoped resources to respective scoped profiles is implied by the presence of scoped instances of the
1106 CIM_RegisteredProfile class.

1107 **9.1.5.1 Determine resource type support of scoped resources (central class methodology)**

1108 **Assumption:** The client has a reference referring to an instance of the CIM_ComputerSystem class that
1109 represents a virtual system that is a central instance of this profile; see 9.1.1. A situation as shown in
1110 Figure 7 for processors is assumed.

1111 The first part of this use case determines the profile implementation advertisement methodology for proc-
1112 essors.

- 1113 1) The client resolves the CIM_ElementConformsToProfile association to locate associated in-
1114 stances of the CIM_RegisteredProfile class, invoking the intrinsic AssociatorNames() CIM op-
1115 eration as follows:
 - 1116 – The value of the ObjectName parameter references the instance of the CIM_Computer-
1117 System class.
 - 1118 – The value of the AssocClass parameter is set to "CIM_ElementConformsToProfile".
 - 1119 – The value of the ResultClass parameter is set to "CIM_RegisteredProfile".
 - 1120 – The result is a list of references referring to instances of the CIM_RegisteredProfile class
1121 representing implementations of this profile; if the operation is successful, the size of the
1122 result set is 1.
- 1123 2) The client resolves the CIM_ReferencedProfile association to locate scoped instances of the
1124 CIM_RegisteredProfile class, invoking the intrinsic Associators() CIM operation as follows:
 - 1125 – The value of the ObjectName parameter is set to the reference referring to the instance of
1126 the CIM_RegisteredProfile class obtained in step 1).
 - 1127 – The value of the AssocClass parameter is set to "CIM_ReferencedProfile".
 - 1128 – The value of the ResultClass parameter is set to "CIM_RegisteredProfile".
 - 1129 – The result is a list of instances of the CIM_RegisteredProfile class representing implemen-
1130 tations of scoped profiles.
- 1131 3) The client iterates over the list obtained in step 2), selecting only instances where the Regis-
1132 teredName property has a value of "CPU".
 - 1133 – The result is a list of instances of the CIM_RegisteredProfile class that represents imple-
1134 mentations of scoped profiles implementing [DSP1022](#) (CPU Profile).
- 1135 4) The client resolves the CIM_ElementConformsToProfile association for each of the instances of
1136 the CIM_RegisteredProfile class from step 3) to locate at least one associated instance of the
1137 CIM_Processor class, invoking the intrinsic Associators() CIM operation as follows:
 - 1138 – The value of the ObjectName parameter is set to the reference taken from the instance of
1139 the CIM_RegisteredProfile class obtained in step 3).
 - 1140 – The value of the AssocClass parameter is set to "CIM_ReferencedProfile".
 - 1141 – The value of the ResultClass parameter is set to "CIM_Processor".
 - 1142 – The result is a list of instances of the CIM_Processor class that are central instances of
1143 [DSP1022](#); the list may be empty.

1144 If for any of the results from step 4) at least one instance of the CIM_Processor class was detected, then
 1145 the central class profile implementation advertisement methodology is applied by the implementation with
 1146 respect to implementations of [DSP1022](#); this is the case in this example. If no such instances were de-
 1147 tected, then the scoping class profile implementation advertisement methodology would have been ap-
 1148 plied.

1149 At this point the client has validated that [DSP1022](#) is implemented as a scoped profile of this profile, and
 1150 that the central class profile implementation advertisement methodology is applied by the implementation
 1151 with respect to [DSP1022](#).

1152 In the second part of this use case it is now the responsibility of the client for any detected scoped in-
 1153 stance of the CIM_Processor class to validate that [DSP1022](#) is indeed implemented. The use case de-
 1154 scribes how to locate such instances, and perform the validation:

- 1155 1) Client resolves the CIM_SystemDevice association from the central instance to associated vir-
 1156 tual resources, invoking the intrinsic AssociatorNames() CIM operation as follows:
 - 1157 – The value of the ObjectName parameter is set referring to the instance of the CIM_Compu-
 1158 terSystem class.
 - 1159 – The value of the AssocClass parameter is set to "CIM_SystemDevice".
 - 1160 – The value of the ResultClass parameter is set to "CIM_Processor".
 - 1161 – The result is a list of references referring to scoped instances of the CIM_Processor class
 1162 representing virtual processors.
- 1163 2) For each reference returned by step 1) the client resolves the CIM_ElementConformsToProfile
 1164 association to locate associated instances of the CIM_RegisteredProfile class, invoking the in-
 1165 trinsic Associators() CIM operation as follows:
 - 1166 – The value of parameter ObjectName is set referring to an instance of the CIM_Processor
 1167 class.
 - 1168 – The value of the AssocClass parameter is set to "CIM_ElementConformsToProfile".
 - 1169 – The value of the ResultClass parameter is set to "CIM_RegisteredProfile".
 - 1170 – The result is a list of instances of the CIM_RegisteredProfile class; if the operation is suc-
 1171 cessful, the size of the list is either 0 or 1. A size of 1 indicates that a version of [DSP1022](#)
 1172 is implemented for the particular processor; a size of 0 indicates that [DSP1022](#) is not im-
 1173 plemented for the particular processor.

1174 **Result:** The client knows the set of scoped instances of the CIM_Processor class that represents proces-
 1175 sors of the assumed virtual system, and whether the instances are central instances of [DSP1022](#), that is,
 1176 whether [DSP1022](#) is implemented in the context of these instances.

1177 9.1.5.2 Determine resource type support of scoped resources (scoping class methodology)

1178 **Assumption:** The client has a reference referring an instance of the CIM_ComputerSystem class that
 1179 represents a virtual system that is a central instance of this profile; see 9.1.1. A situation as shown in
 1180 Figure 7 for the "Memory" resource type is assumed.

1181 The first part of this use case determines the profile implementation advertisement methodology for
 1182 memory.

- 1183 3) The client resolves the CIM_ElementConformsToProfile association to locate associated in-
 1184 stances of the CIM_RegisteredProfile class, invoking the intrinsic AssociatorNames() CIM op-
 1185 eration as follows:
 - 1186 – The value of the ObjectName parameter is set to the reference referring to the instance of
 1187 the CIM_ComputerSystem class.
 - 1188 – The value of the AssocClass parameter is set to "CIM_ElementConformsToProfile".

- 1189 – The value of the ResultClass parameter is set to "CIM_RegisteredProfile".
- 1190 – The result is a list of references referring to instances of the CIM_RegisteredProfile class
- 1191 representing implementations of this profile; if the operation is successful, the size of the
- 1192 result set is 1.
- 1193 4) The client resolves the CIM_ReferencedProfile association to locate scoped instances of the
- 1194 CIM_RegisteredProfile class, invoking the intrinsic Associators() CIM operation as follows:
- 1195 – The value of parameter ObjectName is set to the reference referring to the instance of the
- 1196 CIM_RegisteredProfile class obtained in step 1).
- 1197 – The value of the AssocClass parameter is set to "CIM_ReferencedProfile".
- 1198 – The value of the ResultClass parameter is set to "CIM_RegisteredProfile".
- 1199 – The result is a list of instances of the CIM_RegisteredProfile class that represent imple-
- 1200 mentations of scoped profiles.
- 1201 5) The client iterates over the list obtained in step 2), selecting only instances where the Regis-
- 1202 teredName property has a value of "System Memory".
- 1203 – The result is a list of instances of the CIM_RegisteredProfile class that represents imple-
- 1204 mentations of scoped profiles implementing [DSP1026](#) (*System Memory Profile*).
- 1205 6) The client resolves the CIM_ElementConformsToProfile association for each of the instances of
- 1206 the CIM_RegisteredProfile class from step 3) to locate at least one associated instance of the
- 1207 CIM_Memory class, invoking the intrinsic Associators() CIM operation as follows:
- 1208 – The value of the ObjectName parameter is set to the reference taken from the instance of
- 1209 the CIM_RegisteredProfile class obtained in step 3).
- 1210 – The value of the AssocClass parameter is set to "CIM_ElementConformsToProfile".
- 1211 – The value of the ResultClass parameter is set to "CIM_Memory".
- 1212 – The result is a list of instances of the CIM_Memory class that are central instances of the
- 1213 scoped [DSP1026](#). The list may be empty.

1214 If for any of the results from step 6) no instance of the CIM_Memory class was detected, then the scoping

1215 class profile implementation advertisement methodology is applied by the implementation with respect to

1216 implementations of [DSP1026](#); this is the case in this example. If any such instances were detected, then

1217 the central class profile implementation advertisement methodology would have been applied.

1218 At this point the client has validated that [DSP1026](#) is implemented as a scoped profile of this profile, and

1219 that the scoping class profile implementation advertisement methodology is applied by the implementati-

1220 on with respect to [DSP1026](#).

1221 In the second part of this use case the client now may assume for any detected scoped instance of the

1222 CIM_Memory class that [DSP1026](#) is implemented. The use case describes how to locate such instances:

- 1223 1) The client resolves the CIM_SystemDevice association from the central instance to associated
- 1224 virtual resources, invoking the intrinsic AssociatorNames() CIM operation as follows:
- 1225 – The value of the ObjectName parameter is set to the reference referring to the instance of
- 1226 the CIM_ComputerSystem class.
- 1227 – The value of the AssocClass parameter is set to "CIM_SystemDevice".
- 1228 – The value of the ResultClass parameter is set to "CIM_Memory".
- 1229 – The result is a list of references referring to scoped instances of the CIM_Memory class
- 1230 that represents virtual memory.

1231 **Result:** The client knows the set of scoped instances of the CIM_Memory class that represents memory

1232 in the assumed virtual system, and that these are central instances of [DSP1026](#).

1233 9.1.6 Determine the next boot configuration

1234 **Assumption:** The client has a reference referring to an instance of the CIM_ComputerSystem class that
1235 represents a virtual system that is a central instance of this profile.

1236 1) The client resolves the CIM_ElementSettingData association to find instances of the CIM_Boot-
1237 ConfigSetting class that describe the boot configuration of the virtual system, invoking the intrinsic
1238 References() CIM operation as follows:

1239 – the ObjectName parameter referring to the instance of the CIM_ComputerSystem class
1240 that represents the virtual system

1241 – the ResultClass parameter set to a value of "CIM_ElementSettingData"

1242 – the Role parameter set to a value of "ManagedElement"

1243 The result of this step is a set of instances of the CIM_ElementSettingData association.

1244 2) The client analyzes the result set of the previous step and selects that instance of the CIM_Ele-
1245 mentSettingData association that has the IsNext property set to a value of 3 (Is Next For Single
1246 Use) or, if there is no such instance, that has the IsNext property set to a value of 1 (Is Next).

1247 The result of this step is an instance of the CIM_ElementSettingData association where the Set-
1248 tingData property refers to the instance of the CIM_BootConfigSetting class that is used for the
1249 next boot process.

1250 3) The client obtains the instance of the CIM_BootConfigSetting class, using the intrinsic GetIn-
1251 stance() CIM operation with the InstanceName parameter referring to that instance.

1252 **Result:** The client knows the boot configuration that is used during the next "Activate" virtual system state
1253 transition.

1254 9.2 Virtual system operation

1255 This set of use cases describes how a client can perform basic operations on virtual system, like activat-
1256 ing, deactivating, pausing or resuming a virtual system.

1257 9.2.1 Change virtual system state

1258 This use case is a generic use case that describes the generic procedure to effect a virtual system state
1259 change. A number of use cases follow that describe the effects on objects and association instances rep-
1260 resenting virtual systems, their components, and relationships as defined in this profile.

1261 **Assumption:** The client has a reference referring to an instance of the CIM_ComputerSystem class that
1262 represents a virtual system that is a central instance of this profile. The client intends to effect a virtual
1263 system state transition. (For a list of virtual system state transitions defined by this profile, see Table 3.)

1264 1) The client applies the rules outlined in 7.1.2 to determine a value for the RequestedState pa-
1265 rameter of the CIM_EnabledLogicalElement.RequestStateChange() method that designates
1266 the intended state transition.

1267 2) The client resolves the CIM_ElementCapabilities association from the instance of the
1268 CIM_ComputerSystem class to find the instance of the CIM_EnabledLogicalElementCapabilities
1269 class that describes capabilities of the virtual system; if there is no associated instance of
1270 CIM_EnabledLogicalElementCapabilities, then client state management is not supported for the
1271 virtual system.

1272 3) The client analyzes the RequestedStatesSupported property to check whether it contains an
1273 element that designates the intended state transition as determined by step 1). If the Re-
1274 questedStatesSupported property does not contain a respective element, then the intended
1275 state transition is not supported for the virtual system as a client state management activity.
1276 This may be a temporary situation. Also it might still be possible to effect the state transition us-
1277 ing other means, such as the native capabilities of the virtualization platform.

- 1278 4) The client invokes the RequestStateChange method on the instance of the CIM_Computer-
 1279 System class that represents the virtual system, using a value for the RequestedState paramete-
 1280 r as determined in step 1).
- 1281 5) The client checks the return code.
- 1282 – If the return code is zero, the virtual system state transition was performed as requested.
 - 1283 – If the return code is 1, the RequestStateChange method is not implemented by the imple-
 1284 mentation. This should not occur if the checks above were performed.
 - 1285 – If the return code is 2, an error occurred.
 - 1286 – If the return code is 0x1000, the implementation has decided to perform the state transition
 1287 as an asynchronous task. The client may monitor progress by analyzing the instance of the
 1288 CIM_ConcreteJob class returned through the Job parameter.

1289 If the operation is performed as an asynchronous task, the client may obtain intermediate instances of the
 1290 CIM_ComputerSystem class representing the virtual system (see 9.1.2). These would show values for the
 1291 EnabledState and RequestedState properties that indicate an ongoing state transition. For example, dur-
 1292 ing an "activate" virtual system state transition the EnabledState property might show a value of 10 (Start-
 1293 ing) and the RequestedState property might have a value of 2 (Enabled).

1294 **Result:** The virtual system performs the intended virtual system state transition. The client may next ob-
 1295 tain the actual virtual system state by, for example, following the procedures outlined the use case in
 1296 9.1.2.

1297 9.2.2 Activate virtual system

1298 **Assumption:** This use case is predicated on the assumptions described in 9.2.1 and the same starting
 1299 point described in 9.1.3.

- 1300 1) The client applies the steps in the use case described in 9.2.1 to perform an "activate" transi-
 1301 tion, for example using a value of 2 (Enabled) for the RequestedState parameter.
- 1302 2) The client verifies that the operation was executed successfully, making sure that either a return
 1303 code of 0 results or, if the state change is performed as an asynchronous task, by checking that
 1304 the result of the respective instance of the CIM_ConcreteJob class indicates a successful com-
 1305 pletion.

1306 If the operation is performed as an asynchronous task, a client may obtain intermediate elements of the
 1307 virtual system structure (see 9.1.4). This structure might be incomplete during the state transition. For
 1308 example, if a client resolves associations to instances of the CIM_LogicalDevice class that represent the
 1309 virtual resources as shown in Figure 6 (such as, for example, the CIM_SystemDevice association from
 1310 the instance of the CIM_ComputerSystem class representing the virtual system, or the CIM_ElementSet-
 1311 tingData association from the instance of the CIM_ResourceAllocationSettingData class representing the
 1312 virtual resource allocation), then the client might observe that some virtual resources are already allo-
 1313 cated and represented through instances of the CIM_LogicalDevice class, while other virtual resources
 1314 are not yet allocated to the virtual system and not yet represented through instances of the CIM_Logical-
 1315 Device class.

1316 **Result:** The virtual system is in the "active" state as shown in the use case described in Figure 6 and in
 1317 9.1.4.

1318 10 CIM elements

1319 Table 6 lists CIM elements that are defined or specialized for this profile. Each CIM element shall be im-
 1320 plemented as described in Table 6. The CIM Schema descriptions for any referenced element and its
 1321 sub-elements apply.

1322 Sections 7 ("Implementation") and 8 ("Methods") may impose additional requirements on these elements.

1323

Table 6 – CIM elements: Virtual System Profile

Element	Requirement	Notes
Classes		
CIM_AffectedJobElement	Conditional	See 10.1.
CIM_ComputerSystem	Mandatory	See 10.2.
CIM_ConcreteJob	Conditional	See 10.3.
CIM_ElementCapabilities	Conditional	See DSP1052 , clause 10.
CIM_ElementConformsToProfile	Mandatory	See 10.4.
CIM_ElementSettingData	Mandatory	See 10.5.
CIM_EnabledLogicalElementCapabilities	Optional	See 10.6.
CIM_PowerManagementService	Optional	See 10.7.
CIM_ReferencedProfile	Conditional	See 10.8.
CIM_RegisteredProfile	Mandatory	See 10.9.
CIM_SettingsDefineState	Mandatory	See 10.10.
CIM_VirtualSystemSettingData	Mandatory	See 10.11.
CIM_VirtualSystemSettingDataComponent	Conditional	See 10.12.
Indications		
None defined in this profile		

1324 **10.1 CIM_AffectedJobElement**

1325 The implementation of the CIM_AffectedJobElement association is conditional.

1326 Condition: The CIM_ConcreteJob class is implemented; see 10.3.

1327 If the CIM_AffectedJobElement association is implemented, the provisions in this subclause apply.

1328 The CIM_AffectedJobElement association shall associate an instance of the CIM_ComputerSystem class
 1329 representing a virtual system and an instance of the CIM_ConcreteJob class representing an ongoing
 1330 virtual system state transition.

1331 Table 7 lists the requirements for this association.

1332

Table 7 – Association: CIM_AffectedJobElement

Elements	Requirement	Notes
AffectedElement	Mandatory	Key: Reference to an instance of the CIM_ComputerSystem class that represents a virtual system Cardinality: 1
AffectingElement	Mandatory	Key: Reference to an instance of the CIM_ConcreteJob class that represents an ongoing virtual system state transition task Cardinality: *

1333 10.2 CIM_ComputerSystem

1334 The use of the CIM_ComputerSystem class is specialized in [DSP1052](#) and further refined in this profile.

1335 The requirements in Table 8 are in addition to those mandated by [DSP1052](#).

1336 **Table 8 – Class: CIM_ComputerSystem**

Elements	Requirement	Notes
Caption	Optional	None.
Description	Optional	None
ElementName	Optional	None
EnabledState	Mandatory	See 7.1.1.
RequestedState	Mandatory	See 7.1.2.
RequestStateChange()	Conditional	See 8.1.1.

1337 10.3 CIM_ConcreteJob

1338 The implementation of the CIM_ConcreteJob class is conditional.

1339 Condition: Asynchronous execution of methods is implemented; see 8.1.

1340 If the CIM_ConcreteJob class is implemented, the provisions in this subclause apply.

1341 An implementation shall use an instance of the CIM_ConcreteJob class to represent an asynchronous task.

1343 Table 9 lists requirements for elements of this class.

1344 **Table 9 – Class: CIM_ConcreteJob**

Element	Requirement	Description
JobState	Mandatory	See CIM Schema.
TimeOfLastStateChange	Mandatory	See CIM Schema.

1345 10.4 CIM_ElementConformsToProfile

1346 The CIM_ElementConformsToProfile association shall associate each instance of the CIM_Registered-
 1347 Profile class representing an implementation of this profile with each instance of the CIM_Computer-
 1348 System class representing a virtual system that is manageable through that profile implementation.

1349 Table 10 lists the requirements for this association.

1350 **Table 10 – Association: CIM_ElementConformsToProfile**

Element	Requirement	Notes
ConformantStandard	Mandatory	Key: Reference to an instance of the CIM_RegisteredProfile class that represents an implementation of this profile Cardinality: *
ManagedElement	Mandatory	Key: Reference to an instance of the CIM_ComputerSystem class that represents a conformant virtual system Cardinality: *

1351 **10.5 CIM_ElementSettingData**

1352 The CIM_ElementSettingData association associates the top-level instance of the CIM_VirtualSystemSet-
1353 tingData class in a "state" virtual system configuration and top-level instances of the CIM_VirtualSystem-
1354 SettingData class in other virtual system configurations.

1355 Table 11 lists the requirements for this association.

1356 **Table 11 – Association: CIM_ElementSettingData**

Element	Requirement	Notes
ManagedElement	Mandatory	Key: Reference to an instance of the CIM_VirtualSystemSettingData class that represents the virtualization-specific properties of the virtual system Cardinality: 0..1 See 7.3.3 for additional restrictions on the cardinality.
SettingData	Mandatory	Key: Reference to an instance of the CIM_VirtualSystemSettingData class that represents a virtual system configuration Cardinality: * See 7.3.3 for additional restrictions on the cardinality.
IsDefault	Mandatory	See 7.3.11.
IsCurrent	Unspecified	
IsNext	Mandatory	See 7.3.12.
IsMinimum	Mandatory	Shall be set to 1 (Not Applicable)
IsMaximum	Mandatory	Shall be set to 1 (Not Applicable)
NOTE 1 The cardinality of the ManagedElement role is 0..1 (and not 1) because there are instances of the CIM_VirtualSystemSettingData class that do not have an associated instance of the CIM_VirtualSystemSettingData class through the CIM_ElementSettingData association.		
NOTE 2 The cardinality of the SettingData role is * (and not 1) because there are instances of the CIM_VirtualSystemSettingData class that do not have an associated instance of the CIM_VirtualSystemSettingData class through the CIM_ElementSettingData association.		

1357 10.6 CIM_EnabledLogicalElementCapabilities

1358 The use of the CIM_EnabledLogicalElementCapabilities class is specialized in [DSP1052](#).

1359 The requirements denoted in Table 12 are in addition to those mandated by [DSP1052](#).

1360 **Table 12 – Class: CIM_EnabledLogicalElementCapabilities**

Element	Requirement	Notes
RequestedStatesSupported[]	Mandatory	See 7.5.1.1.

1361 10.7 CIM_PowerManagementService

1362 The CIM_PowerManagementService class is specialized by [DSP1027](#). This profile (DSP1057) specifies
1363 additional optional (see 7.7) and conditional (see 8.1.2) elements.

1364 10.8 CIM_ReferencedProfile

1365 The implementation of the CIM_ReferencedProfile association is conditional.

1366 Condition: A scoped resource allocation profile is implemented; see 7.4.2.

1367 If the CIM_ReferencedProfile association is implemented, the provisions in this subclause apply.

1368 An instance of the CIM_ReferencedProfile association shall associate each instance of the CIM_Regis-
1369 teredProfile class representing an implementation of this profile with instances of the
1370 CIM_RegisteredProfile class representing implementations of profiles that model the management of
1371 logical elements in scope of virtual systems.

1372 This profile (DSP1057) refines requirements of [DSP1033](#) by establishing conditions for the support of the
1373 CIM_ReferencedProfile association.

1374 The implementation of the CIM_ReferencedProfile association is conditional with respect to the presence
1375 of an instance of the CIM_RegisteredProfile class representing a profile that is scoped by this profile.

1376 Table 13 contains the requirements for this association.

1377 **Table 13 – Association: CIM_ReferencedProfile**

Element	Requirement	Notes
Antecedent	Mandatory	Key: Reference to an instance of the CIM_RegisteredProfile class that represents an instance of a resource profile describing logical elements Cardinality: 1
Dependent	Mandatory	Key: Reference to an instance of the CIM_RegisteredProfile class that represents an implementation of this profile Cardinality: 0..*

1378 10.9 CIM_RegisteredProfile

1379 The use of the CIM_RegisteredProfile class is specialized by [DSP1033](#).

1380 The requirements denoted in Table 14 are in addition to those mandated by [DSP1033](#).

1381

Table 14 – Class: CIM_RegisteredProfile

Elements	Requirement	Notes
RegisteredOrganization	Mandatory	Shall be set to 2 (DMTF)
RegisteredName	Mandatory	Shall be set to "Virtual System"
RegisteredVersion	Mandatory	Shall be set to the version of this profile: "1.0"

1382 **10.10 CIM_SettingsDefineState**

1383 An instance of the CIM_SettingsDefineState association shall associate each instance of the
 1384 CIM_ComputerSystem class representing a virtual system with the instance of the
 1385 CIM_VirtualSystemSettingData class that represents the virtualization-specific properties of that virtual
 1386 system and is the top-level instance of the "state" virtual system configuration.

1387 Table 15 contains the requirements for this association.

1388

Table 15 – Association: CIM_SettingsDefineState

Elements	Requirement	Notes
ManagedElement	Mandatory	Key: Reference to an instance of the CIM_ComputerSystem class that represents a virtual system Cardinality: 0..1 See 7.3.2 for additional restrictions on the cardinality.
SettingData	Mandatory	Key: Reference to an instance of the CIM_VirtualSystemSettingData class that represents the virtualization-specific properties of a virtual system. Cardinality: 1
NOTE The cardinality of the ManagedElement role is 0..1 (and not 1) because there are instances of the CIM_VirtualSystemSettingData class that do not have an associated instance of the CIM_ComputerSystem class through the CIM_SettingsDefineState association.		

1389 **10.11 CIM_VirtualSystemSettingData**

1390 The CIM_VirtualSystemSettingData class models virtualization-specific aspects of a virtual system.

1391 Table 16 contains the requirements for this class.

1392

Table 16 – Class: CIM_VirtualSystemSettingData

Element	Requirement	Notes
InstanceID	Mandatory	Key
Caption	Optional	See 7.3.6.
Description	Optional	See 7.3.7.
ElementName	Mandatory	See 7.3.8.
VirtualSystemIdentifier	Optional	See 7.3.9.
VirtualSystemType	Optional	See 7.3.10.

1393 **10.12 CIM_VirtualSystemSettingDataComponent**

1394 The implementation of the CIM_VirtualSystemSettingData component association is conditional.

1395 Condition: Component profiles of this profile are implemented, such as [DSP1044](#), [DSP1045](#) or [DSP1059](#).

1396 If the CIM_VirtualSystemSettingDataComponent association is implemented, the provisions in this sub-
1397 clause apply.

1398 An instance of the CIM_VirtualSystemSettingDataComponent association shall associate each instance
1399 of the CIM_VirtualSystemSettingData class representing the virtual aspects of a virtual system with in-
1400 stances of the CIM_ResourceAllocationSettingData class representing virtual aspects of virtual resources
1401 of that virtual system.

1402 Table 17 contains the requirements for this association.

1403 **Table 17 – Association: CIM_VirtualSystemSettingDataComponent**

Elements	Requirement	Notes
GroupComponent	Mandatory	Key: Reference to an instance of the CIM_VirtualSystemSettingData class that represents the virtual aspects of a virtual system Cardinality: 1
PartComponent	Mandatory	Key: Reference to an instance of the CIM_ResourceAllocationSettingData class that represents virtual aspects of a virtual resource Cardinality: 0..*

1404

Annex A (Informative)

Virtual system modeling — background information

1405
1406
1407
1408

1409 **A.1 Concepts: Model, view, controller**

1410 This profile (like any profile) specifies only an interface or view to an otherwise opaque internal model
1411 maintained by an implementation. This profile does not specify how a virtual system is modeled within an
1412 implementation; this profile specifies only a view of that internal model and some control elements. The
1413 view enables a client to *observe* the internal model; the control elements enable a client to *effect* model
1414 *changes* that in turn become visible through the view.

1415 The view is specified in terms of CIM classes and CIM associations; the control elements are specified in
1416 terms of CIM methods. For that reason the term *CIM model* is frequently used instead of view. This is ac-
1417 ceptable as long as it is understood that a CIM model in fact just represents an interface or view to the
1418 internal model maintained by the implementation.

1419 The implementation presents instances of CIM classes and associations on request from clients. These
1420 instances are fed with data that the implementation obtains from the internal model, using implementa-
1421 tion-specific means. The implementation executes CIM methods on request from clients. CIM methods
1422 are realized using implementation-specific control mechanisms such as program or command-line inter-
1423 faces, for example.

1424 This profile does not specify restrictions on the internal model itself. For example, the implementation is
1425 free to decide which elements of its internal model it exposes through the view defined by this profile, and
1426 in most cases the CIM view exposes only a very limited subset of the internal model.

1427 **A.2 Aspect-oriented modeling approach**

1428 One possible approach to model system virtualization would be to specify virtualization-specific derived
1429 classes for virtual systems and components. For example, to model a virtual system one could model a
1430 CIM_VirtualComputerSystem class extending the CIM_ComputerSystem class with virtualization-specific
1431 properties and methods.

1432 This inheritance-based modeling approach was not applied for various reasons:

- 1433 • A virtual system should appear to a virtualization-unaware client exactly like a non-virtual com-
1434 puter system.
- 1435 • The single-inheritance modeling approach is not suited for various management domains being
1436 modeled on top of the same set of base classes. For example, if the CIM_VirtualComputerSys-
1437 tem and CIM_PartitionedComputerSystem classes were both derived from the CIM_Computer-
1438 System class, then a particular instance could represent either a virtual system or a partitioned
1439 system, but not both.
- 1440 • Many virtualization platforms support the concepts of virtual system definition and virtual system
1441 instance. The definition is a formal description of the virtual system; the instance is the internal
1442 representation of the virtual system in the "active" state. Ideally, both definition and instance are
1443 described using the same set of CIM classes.

1444 Instead, a large part of the model specified by this profile is based on classes derived from CIM_Setting-
1445 Data:

- 1446 • Settings allow virtualization-specific information to be modeled separately from the target class.
- 1447 • Settings are ideally suited to model descriptive data, such as virtual resource definitions.

- 1448 • Settings are easily aggregated into larger configurations, such as virtual system configuration
- 1449 covering the virtual system itself and all of its resources.
- 1450 • Settings allow extending the property set of existing classes in an aspect-oriented way. Various
- 1451 aspects, such as "virtualization" and "partitioning," can exist in parallel for the same managed
- 1452 element.

1453 **A.3 Presence of model information**

1454 [DSP1001](#) (the *Management Profile Specification Usage Guide*) requires an autonomous profile to specify

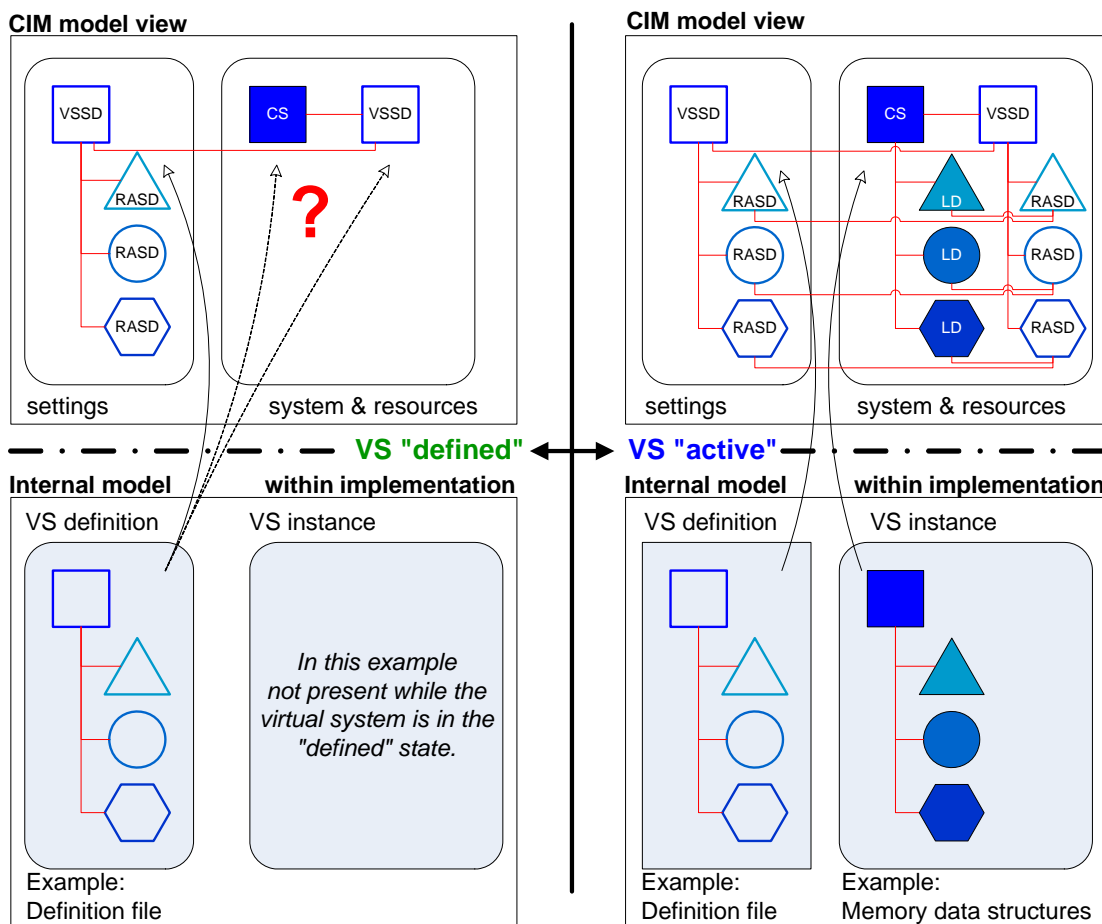
1455 a central class and a scoping class. [DSP1052](#) specifies the CIM_ComputerSystem class for both the central

1456 and scoping class. This profile (DSP1057) specializes [DSP1052](#), and thus is required to use the

1457 CIM_ComputerSystem class (or a derived class) for central and scoping class as well.

1458 [DSP1001](#) further requires that an instance of that class must be present at all times. Figure 8 illustrates

1459 that this requirement in some cases causes a potential model representation problem.



1460

1461

Figure 8 – State-dependent presence of model elements

1462 The left side of Figure 8 shows a virtual system in the "defined" state. In this example the virtualization

1463 platform distinguishes between virtual system definition and virtual system instance; the virtual system

1464 instance does not exist while the virtual system is in the "defined" state. Nevertheless, the implementation

1465 is required to represent a (virtual) computer system through an instance of the CIM_ComputerSystem

1466 class during its complete lifecycle, including periods when the virtual system is only defined but not active

1467 and instantiated at the virtualization platform. This causes a model representation problem: Many proper-

1468 ties of the CIM_ComputerSystem class (with instances labeled "CS" in Figure 8) model information about
1469 a stateful virtual system instance, but not about a stateless virtual system definition.

1470 For that reason the property set of the CIM_ComputerSystem class can only be completely presented by
1471 the implementation while the virtual system is instantiated. While the virtual system is in the "defined"
1472 state, respective properties of the instance of the CIM_ComputerSystem class representing the virtual
1473 system are one of the following:

- 1474 • undefined and have a value of NULL
- 1475 • fed from the virtual system definition instead of from the (in this state, non-existent) virtual sys-
1476 tem instance (This is indicated by the dashed curved arrows in Figure 8.)

1477 The right side of Figure 8 shows the same virtual system in the "active" state. Because in this state the
1478 virtual system instance exists in addition to the virtual system definition, data is directly fed from the virtual
1479 system instance into the system and resources part of the CIM model.

1480 Note that the situation is different for virtual resources. [DSP1041](#) does not require an instance of the
1481 CIM_LogicalDevice class to be present at all times; consequently, instances of the CIM_LogicalDevice
1482 class appear only as long as their scoping virtual system is instantiated.

1483 **A.4 Model extension through settings**

1484 The right side of Figure 8 illustrates another modeling approach applied by this profile: The extension of
1485 the virtual system representation with virtualization-specific properties through settings. The upper right
1486 part of Figure 8 shows how the virtual system itself is represented by an instance of the CIM_Computer-
1487 System class (labeled "CS") and virtual resources are represented by instances of the
1488 CIM_LogicalDevice class (labeled "LD"). On the right side these instances are associated with setting
1489 classes that extend the property set of computer system and resource representations with virtualization-
1490 specific information (labeled VSSD for the virtual system extension and RASD for the set of virtual re-
1491 source extensions). This profile specifies an approach where these extensions are modeled by the same
1492 set of classes that are used to represent a virtual system definition.

Annex B (Informative)

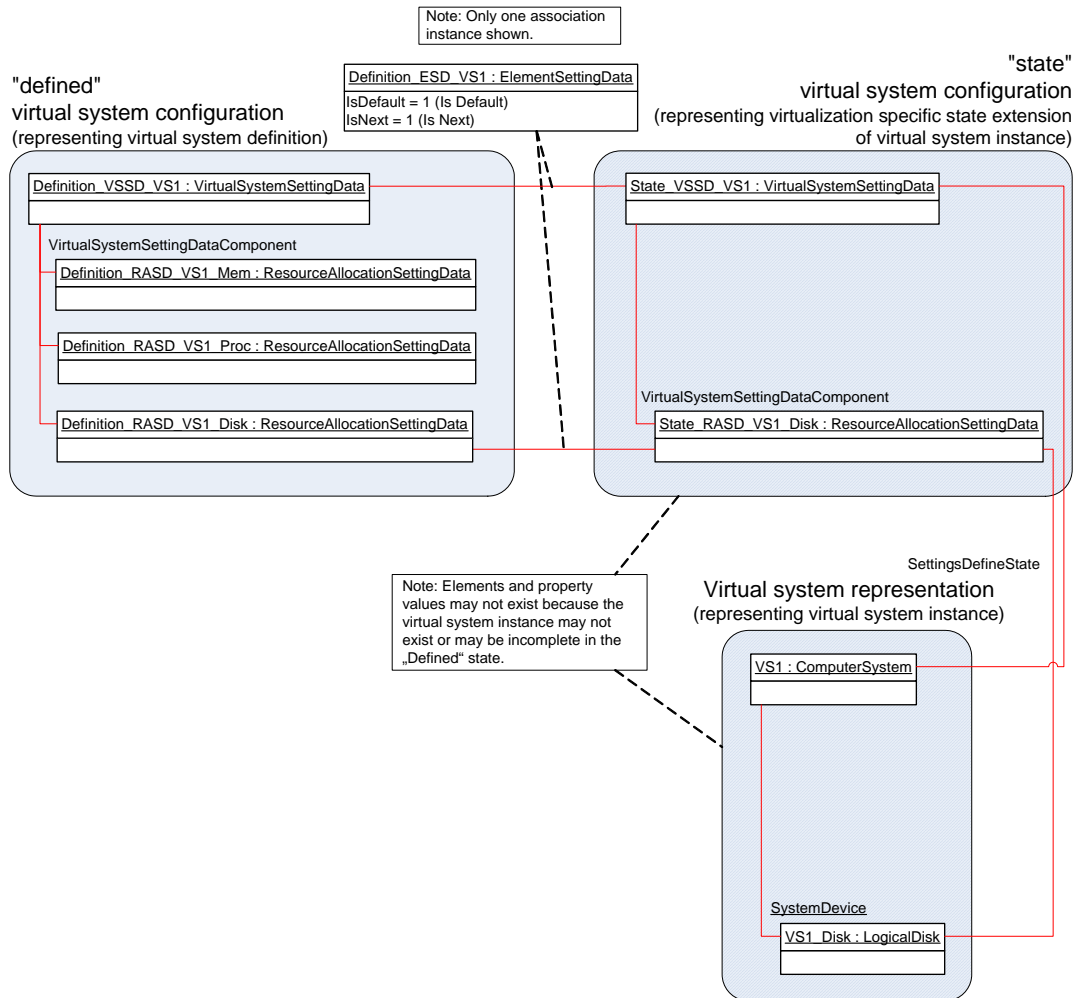
Implementation details

1493
1494
1495
1496

B.1 Dual-configuration implementation approach

1498 Figure 9 shows an example of a virtual system in the "defined" state. There are two virtual system configurations: The virtual system configuration on the left is the "Defined" virtual system configuration: the
1499 virtual system configuration on the right is the "state" virtual system configuration.
1500

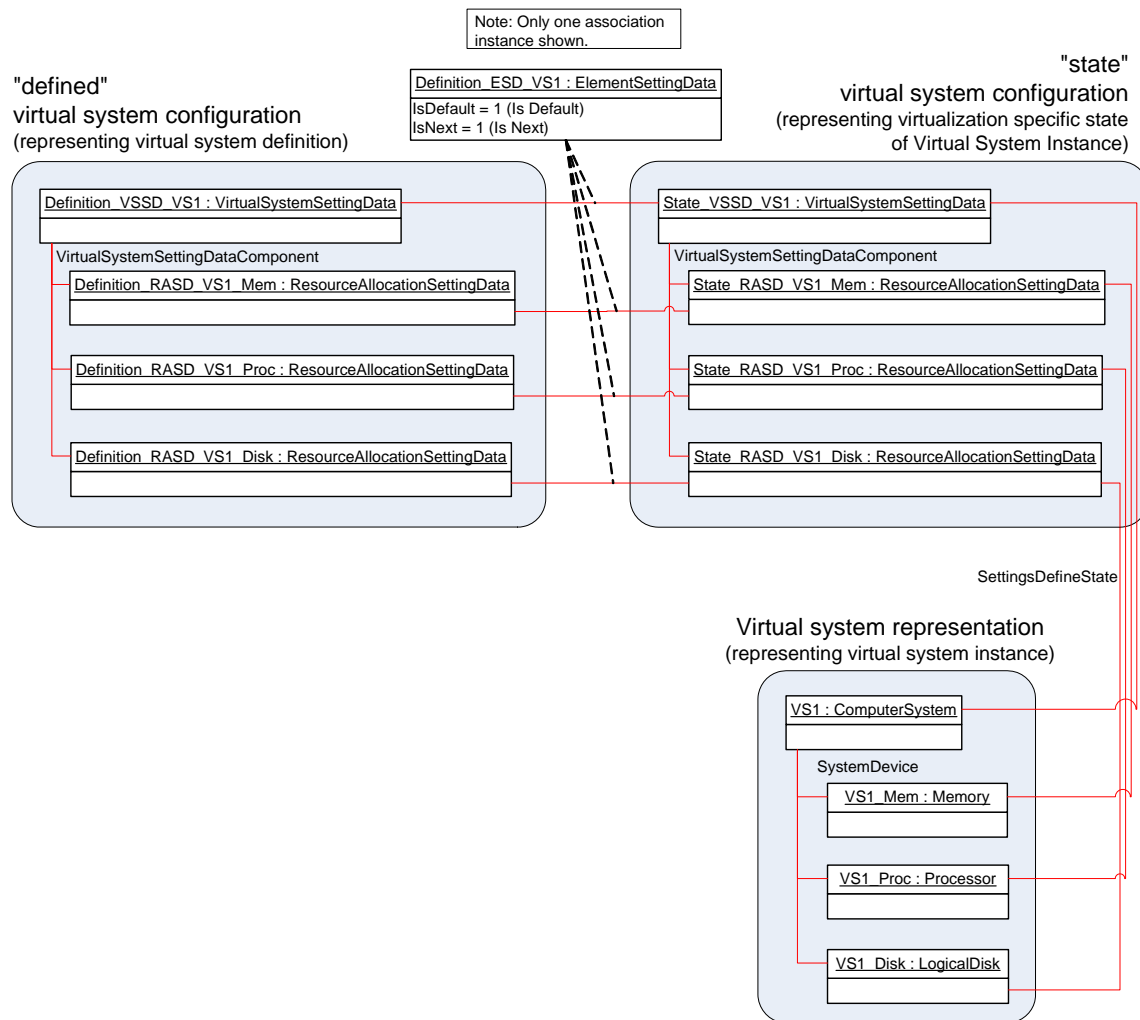
1501 Note that in this example virtual resource VS1_Disk has a persistently allocated resource that remains
1502 allocated regardless of the virtual system state. Consequently, an instance of the CIM_LogicalDisk class
1503 (tagged VS1_Disk) represents the disk in the "defined" state already, and virtualization-specific properties
1504 are represented by an instance of the CIM_ResourceAllocationSettingData class (tagged State_-
1505 RASD_VS1_Disk) in the "state" virtual system configuration that is associated through the CIM_Set-
1506 tingsDefineState association.



1507
1508

Figure 9 – Sample virtual system in "defined" state (Dual-configuration approach)

1509 The same system is shown in Figure 10 in a state other than the "defined" state.



1510
1511 **Figure 10 – Sample virtual system in a state other than "defined" (Dual-configuration approach)**

1512 Resources for virtual resources were allocated, and virtual resources are represented by instances of the
1513 CIM_LogicalDevice class. Virtualization-specific properties are represented as instances of the CIM_Re-
1514 sourceAllocationSettingData class in the "state" virtual system configuration that are associated through
1515 instances of the CIM_SettingsDefineState association.

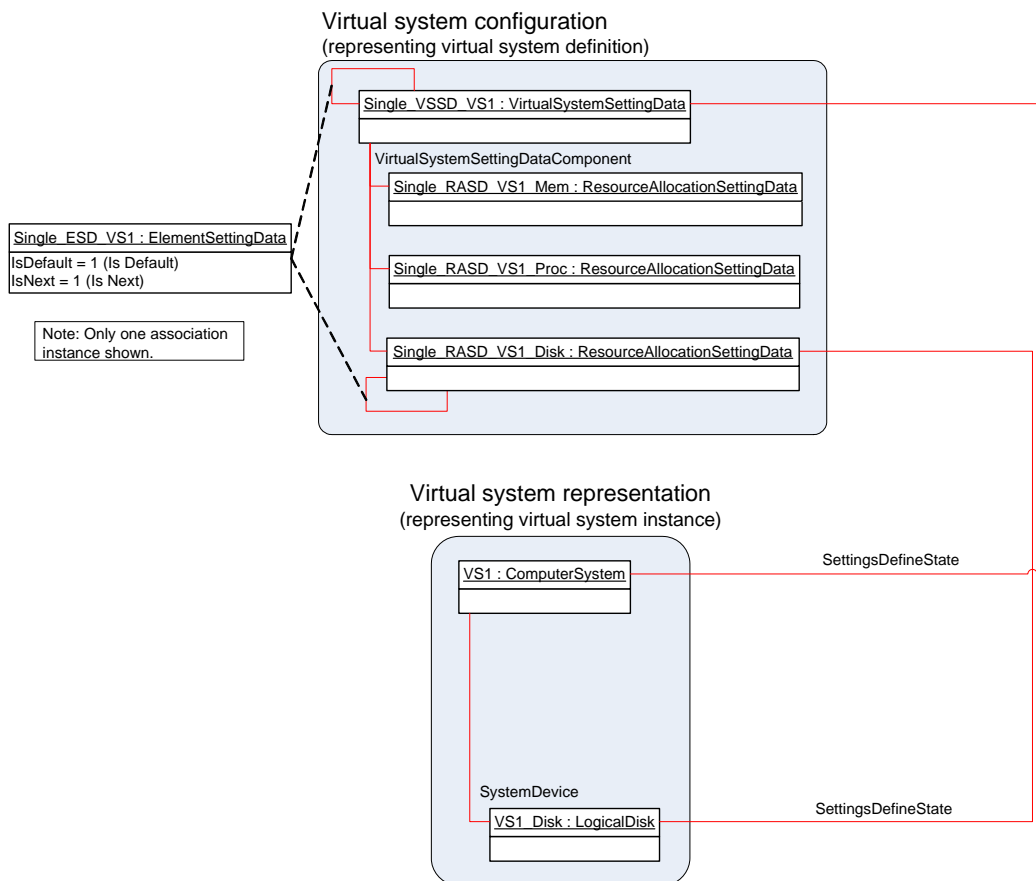
1516 NOTE 1 This profile specifies a CIM view of virtual systems. This profile does not specify restrictions on the internal
1517 model maintained by the implementation to ensure that all resources are allocated during system activation;
1518 instead, the implementation is free to decide whether activation is successful or fails if some virtual re-
1519 sources are not able to be allocated.

1520 NOTE 2 If [DSP1041](#) is implemented for a particular resource type, it may require that, as virtual resources are allo-
1521 cated or de-allocated, respective instances of the CIM_LogicalDevice class are created or destroyed in the
1522 virtual system representation, and that these instances are connected to their counterpart in the "state" vir-
1523 tual system configuration through respective instances of the CIM_SettingsDefineState association, and that
1524 the instances in the "state" virtual system configuration are connected to their counterpart in the "defined"
1525 virtual system configuration through respective instances of the CIM_ElementSettingData association with
1526 the IsDefault property set to 1 (Is Default).

1527 **B.2 Single-configuration implementation approach**

1528 Figure 11 shows an example in which a virtual system is in the "defined" state. Only one set of instances
 1529 of the CIM_VirtualSystemSettingData class and the CIM_ResourceAllocationSettingData class compose
 1530 a single virtual system configuration instance that acts as the "defined" and as the "state" virtual system
 1531 configuration. The single configuration instance is associated to the instance of the CIM_ComputerSys-
 1532 tem class representing the virtual system through an instance of the CIM_SettingsDefineState associa-
 1533 tion.

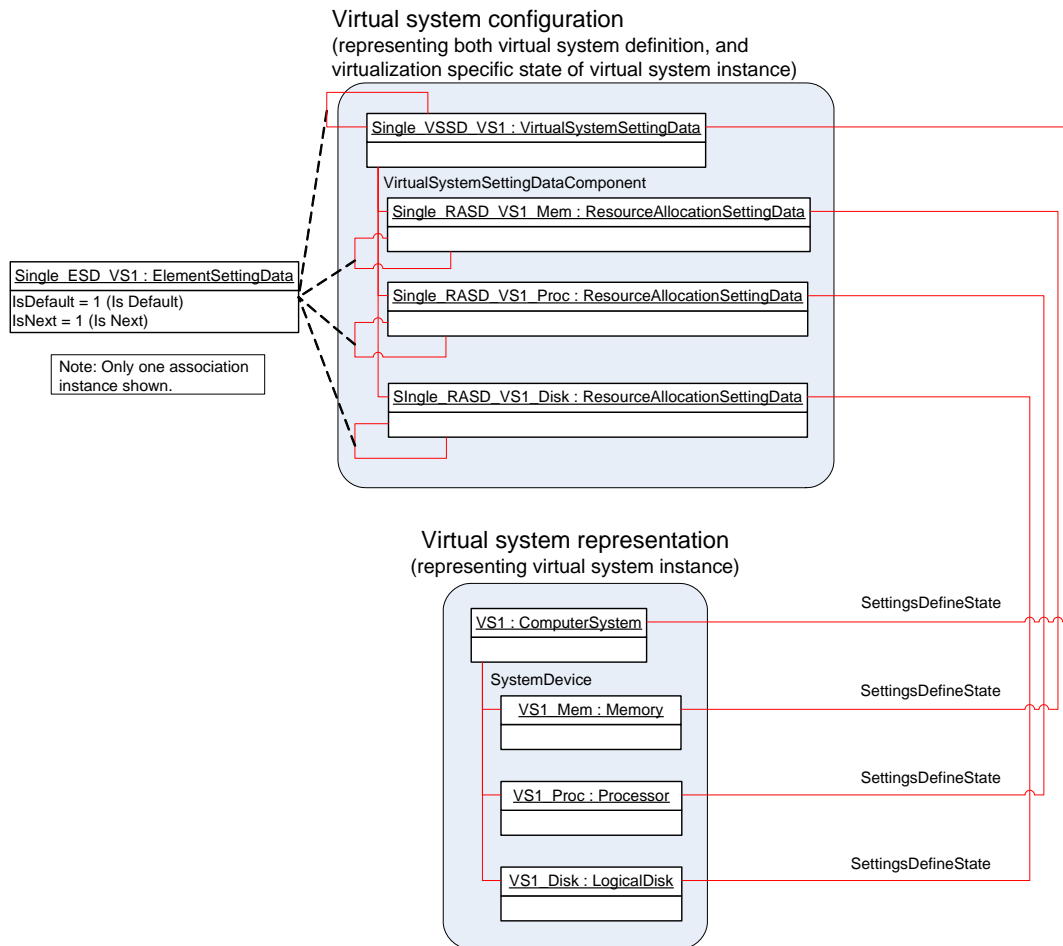
1534 Note that in this example virtual resource VS1_Disk has a persistently allocated resource that remains
 1535 allocated regardless of the virtual system state. Consequently, an instance of the CIM_LogicalDisk class
 1536 tagged VS1_Disk represents the disk in the "defined" state already, and virtualization-specific properties
 1537 are represented by an instance of the CIM_ResourceAllocationSettingData class tagged State_RASD-
 1538 _VS1_Disk in the "state" virtual system configuration that is associated through the CIM_SettingsDefine-
 1539 State association.



1540

1541 **Figure 11 – Sample virtual system in the "defined" state (Single-configuration approach)**

1542 In Figure 12 the same virtual system is shown in a state other than "defined".



1543

1544 **Figure 12 – Sample virtual system in a state other than "defined" (Single-configuration approach)**

1545 Resources for virtual resources were allocated. Virtual resources are represented by instances of the
 1546 CIM_LogicalDevice class, with virtualization-specific properties represented as instances of the CIM_Re-
 1547 sourceAllocationSettingData class in the "state" virtual system configuration and associated through in-
 1548 stances of the CIM_SettingsDefineState association.

1549 **NOTE** If [DSP1041](#) is implemented for a particular resource type, it may require that, as virtual resources are allo-
 1550 cated or de-allocated and respective instances of the CIM_LogicalDevice class are created or destroyed in
 1551 the virtual system representation, these instances are connected to their counterpart in the "state" virtual
 1552 system configuration through respective instances of the CIM_SettingsDefineState association. [DSP1041](#)
 1553 may also require that the instances in the "state" virtual system configuration are connected to their coun-
 1554 terpart in the "defined" virtual system configuration through respective instances of the
 1555 CIM_ElementSettingData association with the `IsDefault` property set to 1 (Is Default); in the single-
 1556 configuration implementation approach, these association instances connect elements of the single virtual
 1557 system configuration to themselves.

1558

**Annex C
(Informative)**

Change Log

1559
1560
1561
1562

1563

Version	Date	Description
1.0.0a	2007-05-07-	Released as preliminary standard.
1.0.0	2010-04-22	Released as DMTF Standard.

1564