



1  
2

3

4

5

**Document Number: DSP1057**

**Date: 2007-05-11**

**Version: 1.0.0a**

6 **Virtual System Profile**

7 **Document Type: Specification**

8 **Document Status: Preliminary Standard**

9 **Document Language: E**

10 [Copyright Notice](#)

11 [Copyright © 2007 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 for uses consistent with this purpose, provided that correct attribution is given. As DMTF  
15 specifications may be revised from time to time, the particular version and release date should always be  
16 noted.

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26 claimant, and shall have no liability or responsibility for costs or losses incurred if a standard is withdrawn  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

## CONTENTS

30	1	Scope .....	9
31	2	Normative References .....	9
32	2.1	Approved References .....	9
33	2.2	References under Development .....	9
34	2.3	Other References .....	9
35	3	Terms and Definitions .....	10
36	4	Symbols and Abbreviated Terms .....	11
37	5	Synopsis .....	12
38	6	Description .....	12
39	6.1	DMTF Management Profile Relationships .....	13
40	6.2	Virtual System Class Schema .....	15
41	6.3	Virtual System Concepts: Definition, Instance, Representation, and Configuration .....	16
42	6.4	Virtual System States and Transitions .....	17
43	6.4.1	Virtual System States .....	17
44	6.4.2	Virtual System State Transitions .....	18
45	6.4.3	Summary of Virtual System States and Virtual System State Transitions .....	19
46	7	Implementation .....	21
47	7.1	Virtual System .....	21
48	7.1.1	CIM_ComputerSystem.EnabledState Property .....	21
49	7.1.2	CIM_ComputerSystem.RequestedState Property .....	22
50	7.2	Virtual Resource .....	24
51	7.3	Virtual System Configuration .....	24
52	7.3.1	Structure .....	24
53	7.3.2	“State” Virtual System Configuration .....	24
54	7.3.3	“Defined” Virtual System Configuration .....	25
55	7.3.4	Implementation Approaches for “State” and “Defined” Virtual System Configuration .....	25
56	7.3.5	Other Types of Virtual System Configurations (Optional) .....	26
58	7.3.6	CIM_VirtualSystemSettingData.Caption (Optional) .....	26
59	7.3.7	CIM_VirtualSystemSettingData.Description (Optional) .....	27
60	7.3.8	CIM_VirtualSystemSettingData.ElementName Property (Optional) .....	27
61	7.3.9	CIM_VirtualSystemSettingData.VirtualSystemIdentifier Property (Optional) .....	27
62	7.3.10	CIM_VirtualSystemSettingData.VirtualSystemType Property (Optional) .....	27
63	7.3.11	CIM_ElementSettingData.IsDefault Property .....	27
64	7.3.12	CIM_ElementSettingData.IsNext Property .....	28
65	7.4	Profile Registration .....	28
66	7.4.1	This Profile (Virtual System Profile) .....	28
67	7.4.2	Scoped DMTF Management Profiles .....	28
68	7.5	Capabilities .....	29
69	7.6	Client State Management (Conditional) .....	29
70	7.7	Power State Management (Optional) .....	29
71	7.7.1	CIM_AssociatedPowerManagementService.PowerState Property (Conditional) .....	30
72	8	Methods .....	30
73	8.1	Extrinsic Methods .....	30
74	8.1.1	CIM_ComputerSystem.RequestStateChange( ) (Conditional) .....	30
75	8.1.2	CIM_PowerManagementService.RequestPowerStateChange( ) (Conditional) .....	31
76	8.2	Profile Conventions for Operations .....	31
77	8.2.1	CIM_ComputerSystem .....	32
78	8.2.2	CIM_ConcreteJob .....	32
79	8.2.3	CIM_ElementSettingData .....	32
80	8.2.4	CIM_EnabledLogicalElementCapabilities .....	32
81	8.2.5	CIM_ReferencedProfile .....	32
82	8.2.6	CIM_RegisteredProfile .....	33
83	8.2.7	CIM_VirtualSystemSettingData .....	33
84	8.2.8	CIM_VirtualSystemSettingDataComponent .....	33

## Virtual System Profile

85	9	Use Cases.....	33
86	9.1	Virtual System Detection and Inspection.....	33
87	9.1.1	Discover Conformant Virtual Systems Using SLP.....	34
88	9.1.2	Determine a Virtual System's State and Other Properties.....	34
89	9.1.3	Determine the "Defined" Virtual System Configuration.....	35
90	9.1.4	Determine the Virtual System Structure.....	36
91	9.1.5	Determine Resource Type Support.....	37
92	9.1.6	Determine the Next Boot Configuration.....	41
93	9.2	Virtual System Operation.....	41
94	9.2.1	Change Virtual System State.....	41
95	9.2.2	Activate Virtual System.....	42
96	10	CIM Elements.....	43
97	10.1	CIM_AffectedJobElement (Conditional).....	43
98	10.2	CIM_ComputerSystem.....	44
99	10.3	CIM_ConcreteJob (Conditional).....	44
100	10.4	CIM_ElementConformsToProfile.....	44
101	10.5	CIM_ElementSettingData.....	45
102	10.6	CIM_EnabledLogicalElementCapabilities.....	46
103	10.7	CIM_PowerManagementService.....	46
104	10.8	CIM_ReferencedProfile (Conditional).....	46
105	10.9	CIM_RegisteredProfile.....	47
106	10.10	CIM_SettingsDefineState.....	47
107	10.11	CIM_VirtualSystemSettingData.....	47
108	10.12	CIM_VirtualSystemSettingDataComponent (Conditional).....	48

109

## 110 Figures

111	Figure 1 –DMTF Management Profiles Related to System Virtualization.....	14
112	Figure 2 – Virtual System Profile: Class Diagram.....	15
113	Figure 3 – Virtual System States.....	20
114	Figure 4 – Virtual System Representation and Virtual System Configuration.....	24
115	Figure 5 – Sample Virtual System Configuration.....	35
116	Figure 6 – Sample Virtual System in "Active" State.....	36
117	Figure 7 – Instance Diagram: Profile Conformance of Scoped Resources.....	37
118	Figure 8 – State-Dependent Presence of Model Elements.....	50
119	Figure 9 – Sample Virtual System in "Defined" State (Dual-Configuration Approach).....	52
120	Figure 10 – Sample Virtual System in a State Other Than "Defined" (Dual-Configuration Approach).....	53
121	Figure 11 – Sample Virtual System in a "Defined" State (Single-Configuration Approach).....	54
122	Figure 12 – Sample Virtual System in a State Other Than "Defined" (Single-Configuration Approach)....	55

123

## 124 Tables

125	Table 1 – Related Profiles.....	12
126	Table 2 – Observation of Virtual System States.....	21
127	Table 3 – Observation of Virtual System State Transitions.....	23
128	Table 4 – CIM_ComputerSystem.RequestStateChange( ) Method: Parameters.....	30
129	Table 5 – CIM_PowerManagementService.RequestPowerStateChange( ) Method: Parameters.....	31
130	Table 6 – Operations: CIM_ElementSettingData.....	32
131	Table 7 – Operations: CIM_ReferencedProfile.....	32
132	Table 8 – Operations: CIM_VirtualSystemSettingDataComponent.....	33
133	Table 9 – CIM Elements: Virtual System Profile.....	43

134	Table 10 – Association: CIM_AffectedJobElement .....	44
135	Table 11 – Class: CIM_ComputerSystem .....	44
136	Table 12 – Class: CIM_ConcreteJob.....	44
137	Table 13 – Association: CIM_ElementConformsToProfile .....	45
138	Table 14 – Association: CIM_ElementSettingData.....	45
139	Table 15 – Class: CIM_EnabledLogicalElementCapabilities .....	46
140	Table 16 – Association: CIM_ReferencedProfile.....	46
141	Table 17 – Class: CIM_RegisteredProfile .....	47
142	Table 18 – Association: CIM_SettingsDefineState.....	47
143	Table 19 – Class: CIM_VirtualSystemSettingData .....	47
144	Table 20 – Association: CIM_VirtualSystemSettingDataComponent.....	48
145		



146

## Foreword

147 This profile - the *Virtual System Profile* (DSP1057) - was prepared by the System Virtualization, Partition-  
148 ing and Clustering Working Group of the DMTF.

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

151

## Introduction

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



157

# Virtual System Profile

## 158 1 Scope

159 This profile - the *Virtual System Profile* - is an autonomous DMTF management profile that defines the  
160 minimum object model needed to provide for the inspection of a virtual system and its components. In  
161 addition, it defines optional basic control operations for activating, deactivating, pausing, or suspending a  
162 virtual system.

## 163 2 Normative References

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

### 167 2.1 Approved References

- 168 DMTF [DSP0004](#), *CIM Infrastructure Specification 2.3.0*
- 169 DMTF [DSP0200](#), *CIM Operations over HTTP 1.2.0*
- 170 DMTF [DSP1000](#), *Management Profile Specification Template 1.0*
- 171 DMTF [DSP1001](#), *Management Profile Specification Usage Guide 1.0*
- 172 DMTF [DSP1012](#), *Boot Control Profile 1.0*
- 173 DMTF [DSP1022](#), *CPU Profile 1.0*
- 174 DMTF [DSP1026](#), *System Memory Profile 1.0*
- 175 DMTF [DSP1027](#), *Power State Management Profile 1.0*
- 176 DMTF [DSP1033](#), *Profile Registration Profile 1.0*
- 177 DMTF [DSP1041](#), *Resource Allocation Profile 1.0*
- 178 DMTF [DSP1043](#), *Allocation Capabilities Profile 1.0*
- 179 DMTF [DSP1052](#), *Computer System Profile 1.0*

### 180 2.2 References under Development

- 181 DMTF [DSP1042](#), *System Virtualization Profile 0.7.4*
- 182 DMTF [DSP1059](#), *Generic Device Resource Virtualization Profile 0.9.0*

### 183 2.3 Other References

- 184 ISO/IEC [Directives, Part 2](#), *Rules for the structure and drafting of International Standards*
- 185 OMG [UMLINF2.0](#), *Unified Modeling Language: Infrastructure*

186 OMG [UMLSUP2.0](#), *Unified Modeling Language: Superstructure*

187 OPENSLLP [RFC2608](#), *RFC Service Location Protocol Version 2*

### 188 **3 Terms and Definitions**

189 For the purposes of this document, the following terms and definitions apply. For the purposes of this  
190 document, the terms and definitions given in DSP1033, DSP1001, and DSP1052 also apply.

#### 191 **3.1**

##### 192 **can**

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

#### 194 **3.2**

##### 195 **cannot**

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

#### 197 **3.3**

##### 198 **conditional**

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

#### 201 **3.4**

##### 202 **mandatory**

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

#### 205 **3.5**

##### 206 **may**

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

#### 208 **3.6**

##### 209 **need not**

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

#### 211 **3.7**

##### 212 **optional**

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

#### 214 **3.8**

##### 215 **referencing profile**

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

#### 218 **3.9**

##### 219 **shall**

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

- 222 **3.10**  
223 **shall not**  
224 indicates requirements strictly to be followed in order to conform to the document and from which no de-  
225 viation is permitted
- 226 **3.11**  
227 **should**  
228 indicates that among several possibilities, one is recommended as particularly suitable, without mention-  
229 ing or excluding others, or that a certain course of action is preferred but not necessarily required
- 230 **3.12**  
231 **should not**  
232 indicates that a certain possibility or course of action is deprecated but not prohibited
- 233 **3.13**  
234 **unspecified**  
235 indicates that this profile does not define any constraints for the referenced CIM element
- 236 **3.14**  
237 **implementation**  
238 a set of CIM providers that realize the classes specified by this profile
- 239 **3.15**  
240 **client**  
241 an application that exploits facilities specified by this profile
- 242 **3.16**  
243 **virtualization platform**  
244 virtualizing infrastructure provided by a host system enabling the deployment of virtual systems
- 245 **4 Symbols and Abbreviated Terms**
- 246 **4.1**  
247 **CIM**  
248 Common Information Model
- 249 **4.2**  
250 **CIMOM**  
251 CIM object manager
- 252 **4.3**  
253 **RASD**  
254 CIM\_ResourceAllocationSettingData
- 255 **4.4**  
256 **SLP**  
257 service location protocol

258 **4.5**  
 259 **VS**  
 260 virtual system

261 **4.6**  
 262 **VSSD**  
 263 CIM\_VirtualSystemSettingData

## 264 5 Synopsis

265 **Profile Name:** *Virtual System Profile*

266 **Version:** 1.0.0a

267 **Organization:** DMTF

268 **CIM Schema Version:** 2.16

269 **Central Class:** CIM\_ComputerSystem

270 **Scoping Class:** CIM\_ComputerSystem

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

274 The instance of the CIM\_ComputerSystem class representing a virtual system shall be the central in-  
 275 stance and the scoping instance of this profile.

276 Table 1 lists DMTF management profiles that this profile depends on, or that may be used in context of  
 277 this profile. The *Computer System Profile* lists additional related DMTF management profiles; these rela-  
 278 tionships are not further specified in this profile.

279

**Table 1 – Related Profiles**

Profile Name	Organization	Version	Relationship	Description
<i>Profile Registration Profile</i>	DMTF	1.0	Mandatory	The profile that specifies registered profiles.
<i>Computer System Profile</i>	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 Profile</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 Profile</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.

## 280 6 Description

281 This profile - the *Virtual System Profile* - specializes the abstract autonomous *Computer System Profile*  
 282 that defines the minimum top-level object model needed to define a basic computing platform. The pri-

283 many design objective applied by this profile is that a virtual system and its components appear to a client  
 284 in the same way as a non-virtual system. Typical management tasks such as enumerating, analyzing,  
 285 controlling, or configuring a system should be enabled without requiring the client to understand specific  
 286 aspects of virtual systems.

## 287 6.1 DMTF Management Profile Relationships

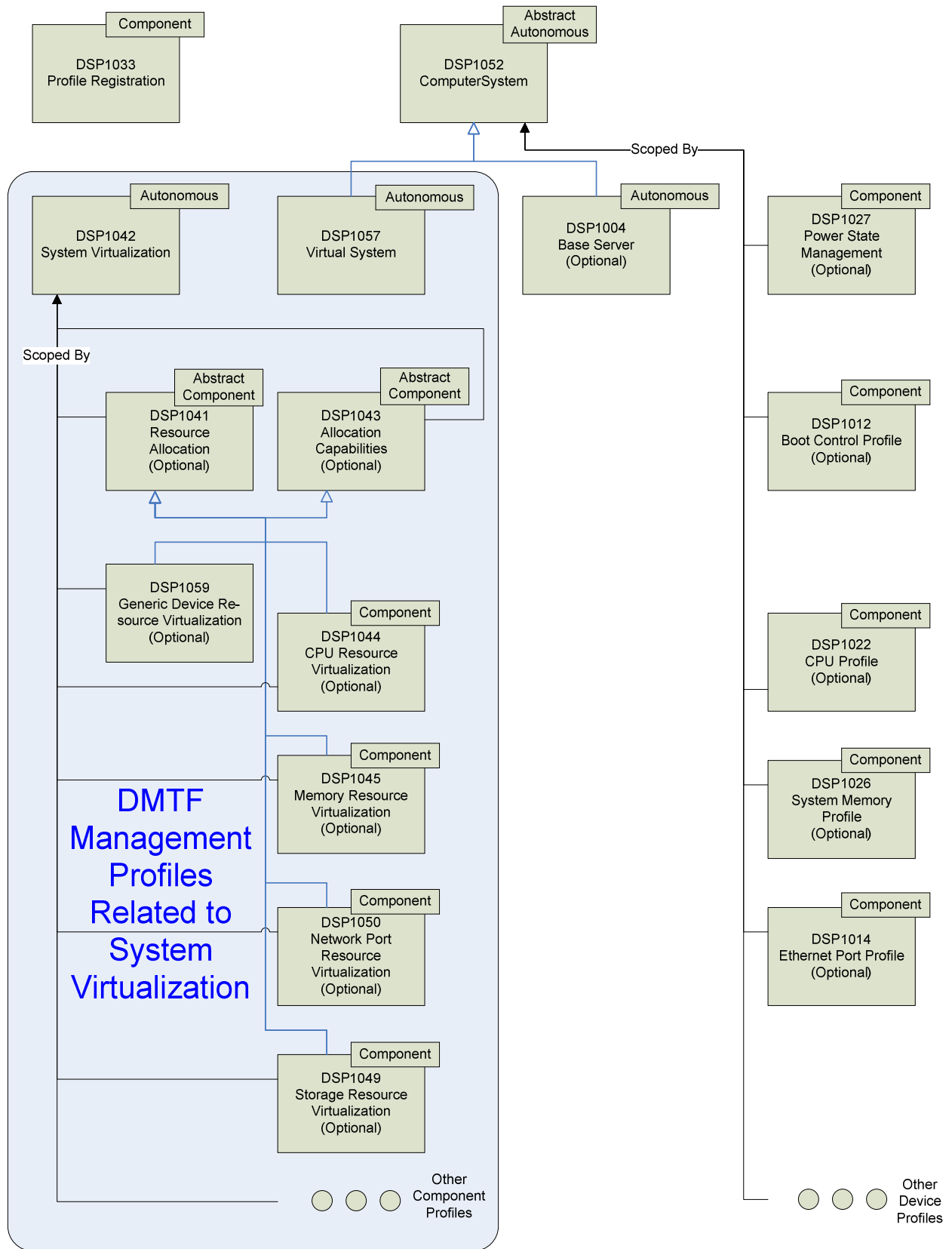
288 This profile - the *Virtual System Profile* - is complementary to the *System Virtualization Profile*:

- 289 • The *Virtual System Profile* focuses on virtualization aspects that relate to virtual systems and  
 290 their virtual resources, such as modeling the *structure* of virtual systems and their resources.  
 291 The profile introduces the concept of virtual system configurations allowing the inspection of vir-  
 292 tual system configuration and state information.
- 293 • The *System Virtualization Profile* focuses on virtualization aspects that relate to host systems  
 294 and their resources, such as modeling the *relationships* between host resources and virtual re-  
 295 sources. Further it addresses virtualization-specific tasks such as the creation or modification of  
 296 virtual systems and their configurations.

297 Figure 1 shows a structure of DMTF management profiles. For example, an implementation that instru-  
 298 ments a virtualization platform may implement some of the following DMTF management profiles:

- 299 • This profile - the *Virtual System Profile*  
 300 The *Virtual System Profile* enables the inspection of and basic operations on virtual systems.
- 301 • *System Virtualization Profile*  
 302 The *System Virtualization Profile* enables the inspection of host systems, their capabilities, and  
 303 their services for creation and manipulation of virtual systems.
- 304 • Resource-type-specific profiles  
 305 Resource-type-specific profiles enable the inspection and operation of resources for one par-  
 306 ticular resource type. They apply to both virtual and host resources; they do not cover virtualiza-  
 307 tion-specific aspects of resources. A client may exploit resource-type-specific management pro-  
 308 files for the inspection and manipulation of virtual and host resources in a similar manner.
- 309 • Resource allocation profiles  
 310 Resource allocation profiles enable the inspection of existing resource allocations and of host  
 311 and other resources available for allocation. Resource allocation profiles are based on the ab-  
 312 stract *Resource Allocation Profile* and the abstract *Allocation Capabilities Profile*, and they are  
 313 scoped by the *System Virtualization Profile*. A client may exploit resource allocation profiles to  
 314 inspect all of the following:
  - 315 – the allocation of resources
  - 316 – the allocation dependencies that virtual resources have on host resources and resource  
 317 pools
  - 318 – the capabilities describing possible values for resource allocations
  - 319 – the capabilities describing the mutability of resource allocations
- 320 The *Generic Device Resource Virtualization Profile* is a resource-type-independent resource al-  
 321 location profile that specifies the management of the allocation of basic virtual resources. For  
 322 some resource types, specific resource allocation profiles are defined that address resource-  
 323 type-specific allocation aspects and capabilities.

# Virtual System Profile



324

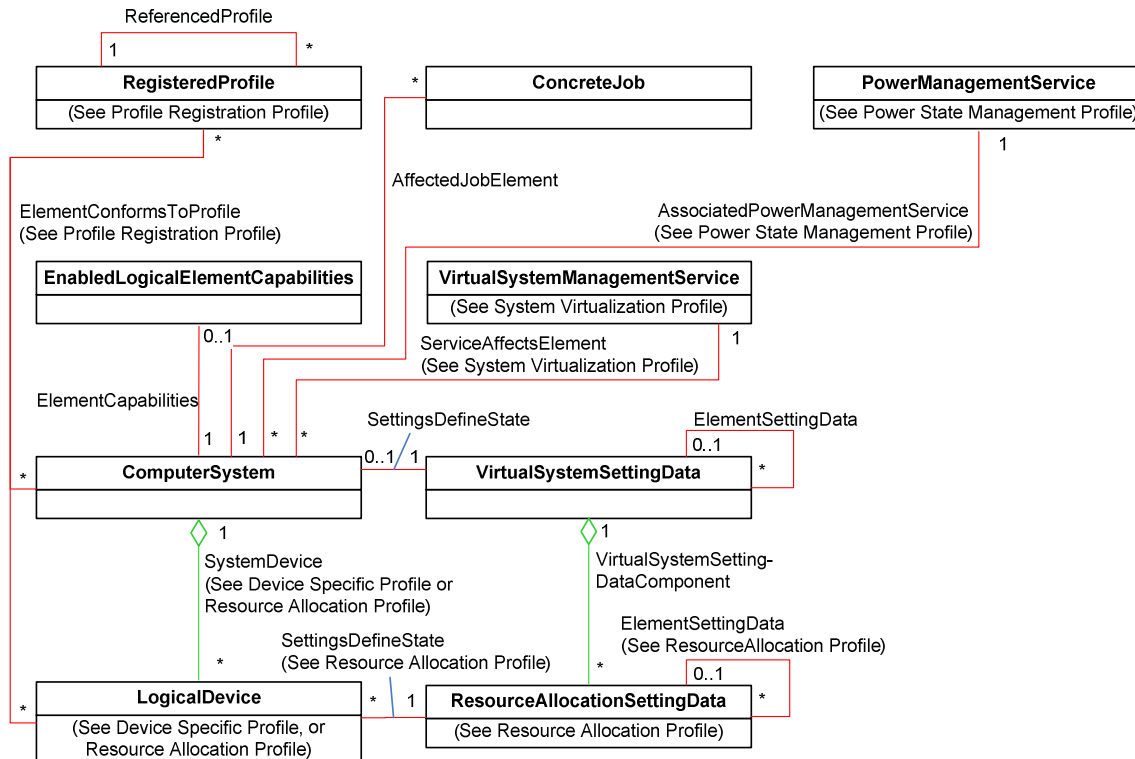
325

**Figure 1 –DMTF Management Profiles Related to System Virtualization**

## 326 6.2 Virtual System Class Schema

327 Figure 2 shows the class schema of this profile – the *Virtual System Profile*. It outlines the elements that  
 328 are owned or specialized by this profile, as well as the dependency relationships between elements of  
 329 this profile and other profiles. For simplicity in diagrams the prefix CIM\_ has been removed from class and  
 330 association names.

331 The *Computer System Profile* references additional classes in its class diagram that outline relationships  
 332 with certain resources, services, and protocol endpoints. This profile - the *Virtual System Profile* – pro-  
 333 vides no specialization of these dependencies. For that reason they are not shown in the class diagram.  
 334 For details, refer to the *Computer System Profile* and to the component profiles referenced there.



335

336

**Figure 2 – Virtual System Profile: Class Diagram**

337 This profile - the *Virtual System Profile* - specifies the use of the following classes and associations:

- 338 • the CIM\_ComputerSystem class to represent virtual systems
- 339 • the CIM\_RegisteredProfile class and the CIM\_ElementConformsToProfile association to model  
 340 conformance with this profile
- 341 • the CIM\_ReferencedProfile association to model dependencies between this profile and  
 342 resource-type-specific resource allocation profiles
- 343 • the CIM\_EnabledLogicalElementCapabilities class and the CIM\_ElementCapabilities  
 344 association to model capabilities of a virtual system such as characteristics of certain properties  
 345 or the set of potential state transitions
- 346 • the CIM\_VirtualSystemSettingData class to model virtualization-specific aspects of a virtual  
 347 system

- 348 • the CIM\_VirtualSystemSettingDataComponent association to model the aggregation of instan-  
349 ces of the CIM\_ResourceAllocationSettingData class to one instance of the CIM\_VirtualSystem-  
350 SettingData class, forming a virtual system configuration
- 351 • the CIM\_SettingsDefineState association to model the relationship between an instance of the  
352 CIM\_ComputerSystem class representing a virtual system and an instance of the CIM\_Virtual-  
353 SystemSettingData class representing virtualization specific aspects of that virtual system
- 354 • the CIM\_ElementSettingData association to model the relationship between an element and  
355 configuration data applicable to the element
- 356 • the CIM\_ConcreteJob class and the CIM\_AffectedJobElement association to model a mecha-  
357 nism that allows tracking of asynchronous tasks resulting from operations such as the optional  
358 RequestStateChange( ) method applied to instances of the CIM\_ComputerSystem class

359 In general, any mention of a class in this document means the class itself or its subclasses. For example,  
360 a statement such as “an instance of the CIM\_LogicalDevice class” implies an instance of the CIM\_Logi-  
361 calDevice class or a subclass of the CIM\_LogicalDevice class.

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

### 363 **6.3 Virtual System Concepts: Definition, Instance, Representation, and Con-** 364 **figuration**

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

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

375 The term *virtual system representation* refers to the set of CIM class instances that represent the current  
376 “State” of a virtual system instance. A virtual system representation consists of one top-level instance of  
377 the CIM\_ComputerSystem class and a set of aggregated instances of the CIM\_LogicalDevice class. The  
378 “State” of the system and logical devices is thus represented by the set of property values in these in-  
379 stances. Virtualization specific “State” is not yet represented; for that purpose the next paragraph intro-  
380 duces a virtualization specific “State” extension to the virtual system representation. The presence of in-  
381 stances of the CIM\_LogicalDevice class within the virtual system representation is controlled by speciali-  
382 zations of the *Resource Allocation Profile*. The specializations describe how instances of the CIM\_Logi-  
383 calDevice class are added or removed from the virtual system representation as virtual resources are al-  
384 located or de-allocated.

385 The term *virtual system configuration* refers to an aggregation of instances of the CIM\_SettingData class:  
386 One top-level instance of the CIM\_VirtualSystemSettingData class and a set of aggregated instances of  
387 the CIM\_ResourceAllocationSettingData class. This profile – the *Virtual System Profile* - specifies the use  
388 of virtual system configurations for two principal purposes:

- 389 • Virtual system configurations are used for the representation of configuration information, in par-  
390 ticular for the representation of virtual system definitions.
- 391 • Virtual system configurations are used for the representation of virtualization specific “State” that  
392 extends the virtual system representation. A single “State” virtual system configuration is associ-  
393 ated to a virtual system. Elements of the “State” virtual system configuration extend corresponding  
394 elements of the virtual system representation with virtualization-specific properties. A variety of vir-  
395 tual system configurations may be associated with the “State” configuration via the CIM\_Element-



396 SettingData association. An example is the representation of the virtual system definition by a  
397 separate “Defined” virtual system configuration.

398 Virtualization platforms may support modifications on virtual system definitions or virtual system instances  
399 through various means, for example through direct configuration file editing, through a command-line in-  
400 terface, through a program interface, or through a CIM-based interface as modeled in the *System Virtual-*  
401 *ization Profile*. Regardless of the mechanism used to effect a modification on a virtual system definition or  
402 a virtual system instance that modification becomes visible to clients through the CIM model view defined  
403 in this profile - the *Virtual System Profile*, as expressed by the respective virtual system configuration or  
404 the virtual system representation.

## 405 **6.4 Virtual System States and Transitions**

406 This subsection informally describes virtual system states and virtual system state transitions. The norma-  
407 tive part of this profile in section 7 specifies how states and state transitions are observed, and a mecha-  
408 nism for the initiation of state transitions.

### 409 **6.4.1 Virtual System States**

410 Subsequent subsections describe various virtual system states and their semantics. Normative require-  
411 ments for the observation of virtual system states are specified in section 7.1.1.

#### 412 **6.4.1.1 Semantics of “Defined” State**

413 In the “Defined” state a virtual system is defined at the virtualization platform, but the virtual system and  
414 its virtual resources need not be instantiated by the virtualization platform. A virtual system in the “De-  
415 fined” state is not enabled to perform tasks. In this state the virtual system does not consume any re-  
416 sources of the virtualization platform, with the exception of persistent resource allocations that remain  
417 allocated regardless of the virtual system state. An example is virtual disk allocations.

#### 418 **6.4.1.2 Semantics of “Active” State**

419 In the “Active” state a virtual system is instantiated at the virtualization platform. Generally the virtual re-  
420 sources are enabled to perform tasks. For example, virtual processors of the virtual system are enabled  
421 to execute instructions. Other virtual resources are enabled to perform respective resource-type-specific  
422 tasks. Nevertheless some virtual resources may not be enabled to perform tasks for various reasons like  
423 for example missing resource allocation. A virtual system is considered to be in the “Active” state as soon  
424 a transition is initiated from another state, and as long as a transition from the “Active” state to another  
425 state is not yet complete. Examples are the activation and deactivation of virtual systems.

#### 426 **6.4.1.3 Semantics of “Paused” State**

427 In the “Paused” state the virtual system and its virtual resources remain instantiated and resources re-  
428 main allocated as in the “Active” state, but the virtual system and its virtual resources are not enabled to  
429 perform tasks.

#### 430 **6.4.1.4 Semantics of “Suspended” State**

431 In the “Suspended” state the state of the virtual system and its virtual resources are stored on non-volatile  
432 storage. The system and its resources are not enabled to perform tasks. It is implementation-dependent  
433 whether virtual resources continue to be represented by instances of the CIM\_LogicalDevice class even if  
434 some or all resources allocated to the virtual resources were de-allocated.

#### 435 **6.4.1.5 Vendor Defined States**

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

438 **6.4.1.6 Semantics of “Unknown” State**

439 “Unknown” is a pseudo-virtual system state indicating that the present virtual system state cannot be de-  
440 termined. For example, the implementation may not be able to contact the virtualization platform hosting  
441 the virtual system because of networking problems.

442 **6.4.2 Virtual System State Transitions**

443 Subsequent subsections describe various virtual system state transitions and their semantics. Normative  
444 requirements for the observation of virtual system state transitions are specified in section 7.1.2.

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

448 **6.4.2.1 “Define” State Transition**

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

451 **6.4.2.2 Semantics of the “Activate” State Transition**

452 While performing the “Activate” state transition from the “Defined” state, missing resources are allocated  
453 according to the virtual system definition, the virtual system and its virtual resources are instantiated and  
454 enabled to perform tasks.

455 While performing an “Activate” state transition from the “Suspended” state back to the “Active” state any  
456 resources that were de-allocated during the transition to and while the system was in the “Suspended”  
457 state are re-allocated, all virtual resources are restored to their previous state and the virtual system is re-  
458 enabled to perform tasks, continuing from the point before the system was suspended.

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

463 While performing an “Activate” state transition from the “Paused” state back to the “Active” state the vir-  
464 tual system and its resources are re-enabled to perform tasks continuing from the point before the system  
465 was paused.

466 **6.4.2.3 Semantics of the “Deactivate” State Transition**

467 While performing the “Deactivate” state transition the virtual system and its virtual resources are disabled  
468 to perform tasks, non-persistent virtual resources are released, their backing resources are de-allocated,  
469 and the virtual system instance is removed from the virtualization platform. If a “Deactivate” state transi-  
470 tion originates from the “Suspended” state, previously saved state information of virtual system and re-  
471 sources is removed. The virtual system remains defined at the virtualization platform.

472 NOTE The “Deactivate” transition is assumed to be disruptive with respect to the virtual system and its components  
473 performing tasks.

474 **6.4.2.4 Semantics of the “Pause” State Transition**

475 While performing the “Pause” state transition the virtual system and its virtual resources are disabled to  
476 perform tasks. The virtual system and its virtual resources remain instantiated with their backing re-  
477 sources allocated.

**478 6.4.2.5 Semantics of the “Suspend” State Transition**

479 While performing the “Suspend” state transition the virtual system and its virtual resources are disabled to  
480 perform tasks and the state of the virtual system and its resources are saved to non-volatile storage. Re-  
481 sources may be de-allocated.

**482 6.4.2.6 Semantics of the “Shut Down” State Transition**

483 While performing the “Shut Down” state transition from the “Active” state, the software that is executed by  
484 the virtual system is notified to shut down. It is assumed that the software then terminates all its tasks and  
485 terminates itself. Subsequent steps of the “Shut Down” state transition should be the same as for the  
486 “Deactivate” state transition.

**487 6.4.2.7 Semantics of the “Reboot” State Transition**

488 While performing the “Reboot” state transition, the software that is executed by the virtual system is noti-  
489 fied to re-cycle or re-boot. Virtual resources remain instantiated with their backing resources allocated.

**490 6.4.2.8 Semantics of the “Reset” State Transition**

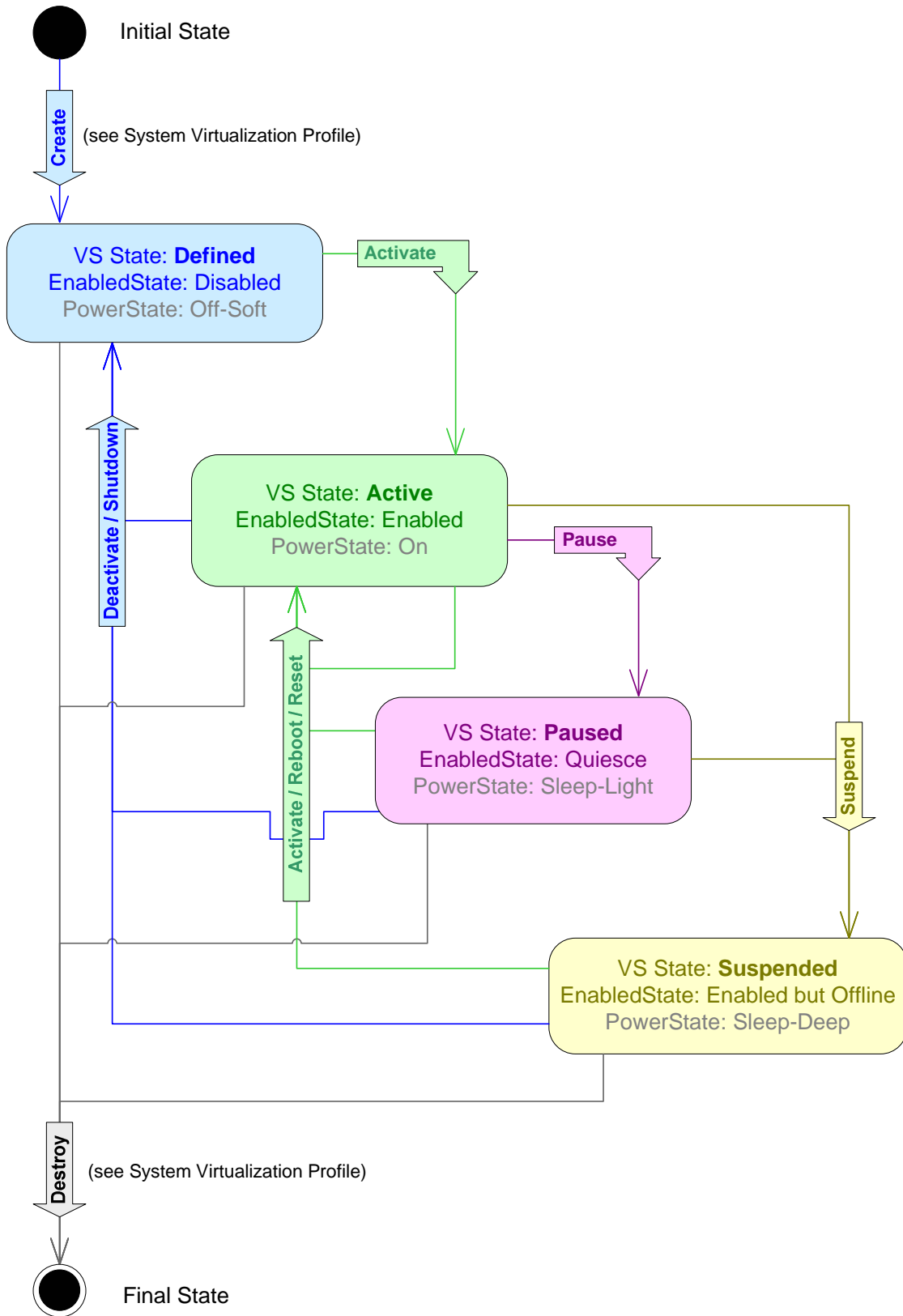
491 Logically the “Reset” state transition consists of a “Deactivate” state transition followed by an “Activate”  
492 state transition, except that resource are not de-allocated during deactivation and thus need not be re-  
493 allocated during activation..

494 NOTE The “Reset” transition is assumed to be disruptive with respect to the virtual system and its components  
495 performing tasks, and state information of the virtual system and its resources may be lost, including state information  
496 saved during a previous “Suspend” state transition.

**497 6.4.3 Summary of Virtual System States and Virtual System State Transitions**

498 Figure 3 summarizes virtual system states that are assumed by this profile and possible state transitions  
499 between those states. Further, Figure 3 shows the mapping of virtual system states to properties of the  
500 CIM\_ComputerSystem class and the CIM\_AssociatedPowerManagementService association.

# Virtual System Profile



501

502

**Figure 3 – Virtual System States**

## 503 7 Implementation

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

506 The list of all methods covered by this profile is in section 8. The list of all properties covered by this pro-  
507 file is in section 10.

508 In references to CIM Schema properties that enumerate values, the numeric value is normative and the  
509 descriptive text following it in parenthesis is informational. For example, in the statement “If an instance of the  
510 CIM\_VirtualSystemManagementCapabilities class contains the value 3 (DestroySystemSupported) in  
511 an element of the SynchronousMethodsSupported[ ] array property”, the “value 3” is normative text and  
512 “(DestroySystemSupported)” is descriptive text.

### 513 7.1 Virtual System

514 The CIM\_ComputerSystem class shall be used to represent virtual systems. One instance of the  
515 CIM\_ComputerSystem class shall exist for each virtual system that is conformant to this profile, regard-  
516 less of its state.

517 This subsection and all secondary subsections apply to instances of the CIM\_ComputerSystem class that  
518 represent virtual systems.

#### 519 7.1.1 CIM\_ComputerSystem.EnabledState Property

520 The EnabledState property shall be supported and used as the primary means to support the observation  
521 of virtual system state (see 6.4.1). Note that as a particular virtual system state is observed through the  
522 value of the EnabledState property a state transition to a different state may already be in progress; this  
523 issue is resolved by modeling the observation of state transitions through the value of the Requested-  
524 State property as defined in section 7.1.2.

525 The “Defined” and “Active” states as defined in section 6.4.1 shall be supported; support of additional  
526 states is optional.

527 Table 2 provides the normative mapping of virtual system states to values of the EnabledState property.  
528 The value of the EnabledState property shall be set depending on the state of the virtual system. For ex-  
529 ample, if a virtual system is in the “Active” state then the EnabledState property should have a value of 2  
530 (Enabled), but may have a value of 8 (Deferred) or 4 (Shutting Down) if respective conditions apply, as  
531 defined by the description of the CIM\_EnabledLogicalElement class in the CIM Schema.

532 **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	<b>3 (Disabled)</b>	<b>8 (Off – Soft)</b> 6 (Off – Hard)
“Active” (See 6.4.1.2)	Mandatory	<b>2 (Enabled)</b> 4 (Shutting Down) 8 (Deferred) 10 (Starting)	<b>2 (On)</b>
“Paused” (Optional) (See 6.4.1.3)	Optional	<b>9 (Quiesce)</b>	<b>3 (Sleep – Light)</b>

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	<b>6 (Enabled but Offline)</b>	<b>4 (Sleep – Deep)</b> 7 (Hibernate (Off – Soft))
Vendor Defined (Optional) (See 6.4.1.5)	Optional	<b>1 (Other)</b>	<b>1 (Other)</b> or (0x7FFF-0xFFFF)
"Unknown" (Optional) (See 6.4.1.6)	Optional	<b>0 (Unknown)</b>	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.

533 The use of the values in the "CIM\_AssociatedPowerManagementService.PowerState Property Value  
534 (Optional)" column listed in Table 2 is described in section 7.7.1.

535 NOTE This profile – the *Virtual System Profile* – clearly distinguishes between the observation of virtual system  
536 state (as defined in this section) and client state management (as defined in section 7.6). In particular with respect to  
537 the observation of virtual system state no mechanism is specified for determining a supported subset of virtual sys-  
538 tem states; instead any virtual system state as defined by Table 2 is possible. Opposed to that the set of state transi-  
539 tions that may be effected through client state management is modeled in section 7.6 through the CIM\_EnabledLogi-  
540 calElementCapabilities class.

### 541 7.1.2 CIM\_ComputerSystem.RequestedState Property

542 The RequestedState property shall be supported. The RequestedState property shall be used to indicate  
543 whether the observation of virtual system state transitions is supported, and if the observation of virtual  
544 system state transitions is supported the property shall indicate ongoing virtual system state transitions.

545 The following provisions apply:

- 546 • If the observation of virtual system state transitions is not supported, the RequestedState prop-  
547 erty shall be set to a value of 12 (Not Applicable).
- 548 • If the observation of one or more virtual system state transitions is supported, the value of the  
549 RequestedState property shall be used to facilitate the observation of virtual system state transi-  
550 tions. The following provisions apply:
  - 551 – The RequestedState property shall not have a value of 12 (Not Applicable).
  - 552 – The RequestedState property shall have a value designating the most recently requested  
553 state transition according to Table 3. For example, if a virtual system is performing an "Ac-  
554 tivate" state transition, then the RequestedState property shall have a value of 2 (Enabled).
  - 555 – If a state transition completes successfully, the value of the EnabledState property shall re-  
556 flect the "To" virtual system state as defined by Table 3, using values as defined by Table  
557 2. For example, if a virtual system has successfully performed an "Activate" state transition,  
558 then it shall be in the "Active" virtual system state and show a value of 2 (Enabled) for the  
559 EnabledState property. The RequestedState property shall maintain the value designating  
560 the most recently requested state transition according to Table 3.

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

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

566 NOTE State transitions may be observed even if client state management as described in section 7.6 is not sup-  
 567 ported. For example, a state transition might be initiated by means inherent to the virtualization platform, or it might  
 568 be triggered during activation of the virtualization platform itself.

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

571 **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	RequestPower StateChange( ): Property Value
Observation of state transitions not supported	n/a	n/a	n/a	<b>12 (Not Applicable)</b>	n/a
“Define” (Optional) (See 6.4.2.1)	Optional	No CIM_ComputerSystem instance	“Defined”	Not applicable. For definition of virtual systems see <i>System Virtualization Profile</i> .	
“Activate” (Optional) (See 6.4.2.2)	Optional	“Defined” “Paused” “Suspended”	“Active”	<b>2 (Enabled)</b>	2 (On)
“Deactivate” (Optional) (See 6.4.2.3)	Optional	“Active” “Paused” “Suspended”	“Defined”	<b>3 (Disabled)</b>	8 (Off – Soft)
“Pause” (Optional) (See 6.4.2.4)	Optional	“Active”	“Paused”	<b>9 (Quiesce)</b>	3 (Sleep–Light)
“Suspend” (Optional) (See 6.4.2.5)	Optional	“Active” “Paused”	“Suspended”	<b>6 (Offline)</b>	4 (Sleep –Deep)
“Shut Down” (Optional) (See 6.4.2.6)	Optional	“Active” “Paused” “Suspended”	“Defined”	<b>4 (Shut Down)</b>	8 (Off – Soft)
“Reboot” (Optional) (See 6.4.2.7)	Optional	“Active” “Paused” “Suspended”	“Active”	<b>10 (Reboot)</b>	5 (Power Cycle (Off – Soft))
“Reset” (Optional) (See 6.4.2.8)	Optional	“Active” “Paused” “Suspended”	“Active”	<b>11 (Reset)</b>	9 (Power Cycle (Off – Hard))
Information about recent or pending state transitions not available	Optional	n/a	n/a	<b>5 (No Change)</b>	n/a

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

572 NOTE This profile – the *Virtual System Profile* – clearly distinguishes between the observation of virtual system  
 573 state transitions (as defined in this section) and client state management (as defined in section 7.6). In particular with  
 574 respect to the observation of virtual system state transitions no mechanism is specified for determining a supported  
 575 subset of virtual system state transitions; instead any virtual system state transition as defined by Table 3 is possible.  
 576 Opposed to that the set of state transitions that may be effected through client state management is modeled in sec-  
 577 tion 7.6 through the CIM\_EnabledLogicalElementCapabilities class.

578 **7.2 Virtual Resource**

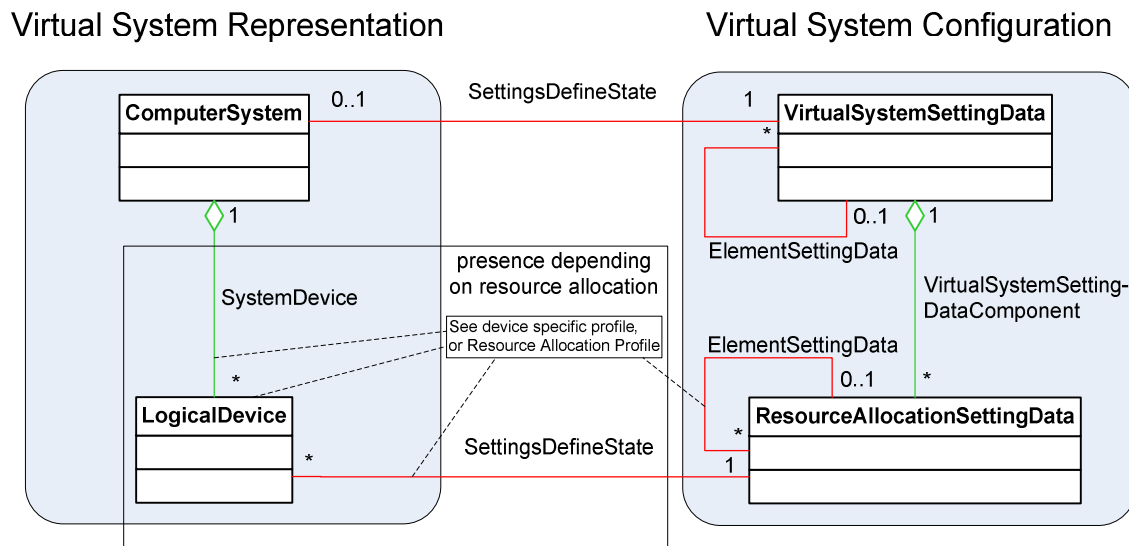
579 Resources in system representations are specified by resource-type-specific DMTF management profiles  
 580 such as the *CPU Profile* or the *System Memory Profile*. These resource-type-specific DMTF management  
 581 profiles may be implemented for one or more types of virtual resources, omitting optional elements that  
 582 model physical aspects.

583 Most resource-type-specific DMTF management profiles specify that logical resources are represented by  
 584 instances of the *CIM\_LogicalDevice* class, and are aggregated into a virtual system representation using  
 585 the *CIM\_SystemDevice* association. This profile – the *Virtual System Profile* – specifies the use of virtual  
 586 system configurations for the extension of virtual system representations with virtualization-specific prop-  
 587 erties.

588 **7.3 Virtual System Configuration**

589 **7.3.1 Structure**

590 A virtual system configuration shall consist of one instance of the *CIM\_VirtualSystemSettingData* class as  
 591 the top-level object, and zero or more instances of the *CIM\_ResourceAllocationSettingData* class. The  
 592 *CIM\_VirtualSystemSettingDataComponent* association shall be used to associate the instance of the  
 593 *CIM\_VirtualSystemSettingData* class with aggregated instances of the *CIM\_ResourceAllocationSetting-*  
 594 *Data* class (see Figure 4).



595

596 **Figure 4 – Virtual System Representation and Virtual System Configuration**

597 **7.3.2 “State” Virtual System Configuration**

598 There shall be exactly one “State” virtual system configuration representing the virtualization specific state  
 599 of the virtual system. Elements of the “State” virtual system configuration add virtualization-specific prop-  
 600 erties to related elements in the virtual system representation. Elements of the “State” virtual system con-  
 601 figuration shall have the same lifecycle as their counterparts in the virtual system representation.

602 The top-level instance of the *CIM\_VirtualSystemSettingData* class in the “State” virtual system configura-  
 603 tion shall be associated to the instance of the *CIM\_ComputerSystem* class that represents the virtual sys-  
 604 tem through an instance of the *CIM\_SettingsDefineState* association.

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



607 NOTE 2: If the *Resource Allocation Profile* is implemented for a particular resource type, it may require additional  
 608 instances of the CIM\_SettingsDefineState association connecting instances of the CIM\_ResourceAllocationSetting-  
 609 Data class in the “State” virtual system configuration to related instances of the CIM\_LogicalDevice class in the virtual  
 610 system representation.

### 611 7.3.3 “Defined” Virtual System Configuration

612 There shall exactly be one “Defined” virtual system configuration representing the virtual system defini-  
 613 tion. The top-level instance of the CIM\_VirtualSystemSettingData class in the “Defined” virtual system  
 614 configuration shall be associated to the top-level instance of the CIM\_VirtualSystemSettingData class in  
 615 the “State” virtual system configuration through the CIM\_ElementSettingData association with the IsDe-  
 616 fault property set to a value of 1 (Is Default).

617 The “Defined” virtual system configuration shall be present at all times regardless of the virtual system  
 618 state.

619 NOTE An implementation may coincide the “Defined” virtual system configuration and the “State” vir-  
 620 tual system configuration; see section 7.3.4.

621 If the *Resource Allocation Profile* is implemented for a particular resource type, it may require additional  
 622 instances of the CIM\_ElementSettingData association to connect instances of the CIM\_ResourceAllocati-  
 623 onSettingData class in the “State” virtual system configuration with their counterparts in the “Defined” vir-  
 624 tual system configuration. The presence of these association instances is not required or defined by this  
 625 profile - the *Virtual System Profile*. However, this profile – the *Virtual System Profile* - requires that any  
 626 instances of the CIM\_ElementSettingData association that are required by the *Resource Allocation Profile*  
 627 shall have an attribute set that is consistent with the attribute set of the instance of the CIM\_ElementSet-  
 628 tingData association that associates the top-level instances of the CIM\_VirtualSystemSettingData class.

### 629 7.3.4 Implementation Approaches for “State” and “Defined” Virtual System Configura- 630 tion

631 Implementations are not required to support separate virtual system configurations for the representation  
 632 of virtual system definition and virtual system instance: Implementations may apply either a dual-configu-  
 633 ration implementation approach (section 7.3.4.1) or a single-configuration implementation approach (sec-  
 634 tion 7.3.4.2); an implementation shall not mix the two implementation approaches. For a detailed instan-  
 635 ce-based description, see Annex B.

#### 636 7.3.4.1 Dual-Configuration Implementation Approach

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

640 With this dual-configuration approach, the “Defined” and the “State” virtual system configurations shall be  
 641 composed of unique instances of the CIM\_VirtualSystemSettingData class and the CIM\_ResourceAlloca-  
 642 tionSettingData class in each configuration.

643 For the top-level instance of the CIM\_VirtualSystemSettingData class in the “State” virtual system con-  
 644 figuration the following provisions apply:

- 645 • It shall be associated to the instance of the CIM\_ComputerSystem class in the virtual system  
 646 representation through an instance of the CIM\_SettingsDefineState association
- 647 • It shall be associated to its counterpart in the “Defined” virtual system configuration through an  
 648 instance of the CIM\_ElementSettingData association where
  - 649 – the value of the IsDefault property shall be set to according to 7.3.11
  - 650 – the value of the IsNext property shall be set to according to 7.3.12

- 651           • It shall be associated to any instance of the CIM\_ResourceAllocationSettingData class that is  
652           part of the “State” virtual system configuration via an instance of the CIM\_VirtualSystemSetting-  
653           DataComponent association

654   The *Resource Allocation Profile* may require compliance to similar conditions with respect to instances of  
655   the CIM\_ResourceAllocationSettingData class and the CIM\_LogicalDevice class. If resources are alloca-  
656   ted or de-allocated, respective instances of the CIM\_ResourceAllocationSettingData class shall be added  
657   to or removed from the “State” virtual system configuration along with the associations referring to them.

658   NOTE   The values of the properties within the instances of the CIM\_ElementSettingData association depend on the  
659   virtual system state and/or on the resource allocation state.

#### 660   **7.3.4.2    Single-Configuration Implementation Approach**

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

663   With this approach, instances of the CIM\_VirtualSystemSettingData class and the CIM\_ResourceAlloca-  
664   tionSettingData class are shared for both the “Defined” virtual system configuration and the “State” virtual  
665   system configuration.

666   For the top-level instance of the CIM\_VirtualSystemSettingData class in the single virtual system configu-  
667   ration the following provisions apply:

- 668           • It shall be associated to the instance of the CIM\_ComputerSystem class in the virtual system  
669           representation through an instance of the CIM\_SettingsDefineState association
- 670           • It shall be associated to itself through an instance of the CIM\_ElementSettingData association  
671           where
- 672           –   the value of the IsDefault property shall be set to according to 7.3.11
- 673           –   the value of the IsNext property shall be set to according to 7.3.12
- 674           • It shall be associated to any instance of the CIM\_ResourceAllocationSettingData class that is  
675           part of the single virtual system configuration via an instance of the CIM\_VirtualSystemSetting-  
676           DataComponent association

677   The *Resource Allocation Profile* may require compliance to similar conditions with respect to instances of  
678   the CIM\_ResourceAllocationSettingData class and the CIM\_LogicalDevice class, such that as resources  
679   are allocated or de-allocated, respective instances of the CIM\_SettingsDefineState association and the  
680   CIM\_ElementSettingData association are required to be added to or removed from instances of the  
681   CIM\_ResourceAllocationSettingData class.

682   NOTE   The values of the properties within the instances of the CIM\_ElementSettingData association depend on the  
683   virtual system state and/or on the resource allocation state.

#### 684   **7.3.5    Other Types of Virtual System Configurations (Optional)**

685   Additional virtual system configurations may be associated to the “State” virtual system configuration  
686   through the CIM\_ElementSettingData association. For details about the “Next” configuration (the configu-  
687   ration that will be used during the next activation of the virtual system), see section 7.3.12.

#### 688   **7.3.6    CIM\_VirtualSystemSettingData.Caption (Optional)**

689   The Caption property may be supported.

690   If the Caption property is supported for the CIM\_ComputerSystem class, the value of the Caption property  
691   in the instance of the CIM\_VirtualSystemSettingData class in the “State” virtual system configuration of a  
692   virtual system shall be identical to the value of the Caption property in the instance of the CIM\_Computer-  
693   System class representing the virtual system.

### 694 **7.3.7 CIM\_VirtualSystemSettingData.Description (Optional)**

695 The Description property may be supported.

696 If the Description property is supported for the CIM\_ComputerSystem class, the value of the Description  
697 property in the instance of the CIM\_VirtualSystemSettingData class in the “State” virtual system configu-  
698 ration of a virtual system shall be identical to the value of the Description property in the instance of the  
699 CIM\_ComputerSystem class representing the virtual system.

### 700 **7.3.8 CIM\_VirtualSystemSettingData.ElementName Property (Optional)**

701 The ElementName property may be supported. The value of the ElementName property reflects a name  
702 for the virtual system configuration assigned by an end-user or administrator.

703 If the ElementName property is supported for the CIM\_ComputerSystem class, the value of the Element-  
704 Name property in the instance of the CIM\_VirtualSystemSettingData class in the “State” virtual system  
705 configuration of a virtual system shall be identical to the value of the ElementName property in the in-  
706 stance of the CIM\_ComputerSystem class representing the virtual system.

### 707 **7.3.9 CIM\_VirtualSystemSettingData.VirtualSystemIdentifier Property (Optional)**

708 The VirtualSystemIdentifier property may be supported. The value of the VirtualSystemIdentifier property  
709 reflects a name for the virtual system assigned by the implementation during virtual system creation. A  
710 typical example is a human-readable user ID.

### 711 **7.3.10 CIM\_VirtualSystemSettingData.VirtualSystemType Property (Optional)**

712 The VirtualSystemType property may be supported. The value of the VirtualSystemType property reflects  
713 a specific type for the virtual system.

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

719 The virtual system type may change during the lifetime of the virtual system. For example, a change may  
720 be effected through the use of inherent management facilities available with the virtualization platform or  
721 through facilities defined by the *System Virtualization Profile* that enable a client to modify virtual system  
722 configurations.

### 723 **7.3.11 CIM\_ElementSettingData.IsDefault Property**

724 The IsDefault property shall be supported for instances of the CIM\_ElementSettingData association be-  
725 tween a top-level instance of the CIM\_VirtualSystemSettingData class in a “State” virtual system configu-  
726 ration and a top-level instance of the CIM\_VirtualSystemSettingData class in a related virtual system con-  
727 figuration. The IsDefault property shall be used to designate the “Defined” virtual system configuration  
728 among all configurations associated with the “State” virtual system configuration.

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

- 730 • The IsDefault property shall have a value of 1 (Is Default) if the related virtual system configura-  
731 tion is the “Defined” virtual system configuration.
- 732 • In all other cases, the IsDefault property shall have a value of 2 (Is Not Default).
- 733 • The IsDefault property shall not have a value of 0 (Unknown).
- 734 • In the set of all virtual system configurations that are associated to a top-level instance of the  
735 CIM\_VirtualSystemSettingData class in a “State” virtual system configuration exactly one con-

736           figuration shall be referenced by an instance of the CIM\_ElementSettingData association with a  
737           value of 1 (Is Default) for the IsDefault property.

738           The “Defined” virtual system configuration is the fall-back default that shall be used for virtual system acti-  
739           vation if no other configuration is marked through the IsNext property.

### 740   **7.3.12   CIM\_ElementSettingData.IsNext Property**

741           The IsNext property shall be supported for instances of the CIM\_ElementSettingData association be-  
742           tween a top-level instance of the CIM\_VirtualSystemSettingData class in a “State” virtual system configu-  
743           ration and a top-level instance of the CIM\_VirtualSystemSettingData class in a related virtual system con-  
744           figuration. The IsNext property may be used to designate the “Next” virtual system configuration. The  
745           “Next” virtual system configuration is the virtual system configuration that will be used for the next activa-  
746           tion of the virtual system; if no configuration is marked as the “Next” virtual system configuration, the “De-  
747           fault” virtual system configuration is used for the next activation.

748           The value of the IsNext property shall be set as follows:

- 749           •   The IsNext property shall have one of the following values:
  - 750               –   a value of 0 (Unknown) if it is not known whether the referenced virtual system configura-  
751               –   tion will be used for the next activation
  - 752               –   a value of 1 (Is Next) if the referenced virtual system configuration is established to be  
753               –   used for subsequent activations of the virtual system
  - 754               –   a value of 3 (Is Next For Single Use) if the referenced virtual system configuration is estab-  
755               –   lished to be used for just the next activation of the virtual system in preference of the de-  
756               –   fault and or the persistently established next configuration.
  - 757               –   In all other cases the IsNext property shall have a value of 2 (Is Not Next). In this case the  
758               –   “Default” virtual system configuration is used for the next virtual system activation.
- 759           •   In the set of all virtual system configurations that are associated with a top-level instance of the  
760           –   CIM\_VirtualSystemSettingData class in a “State” virtual system configuration, there shall be
  - 761               –   at most one configuration that is referenced by an instance of the CIM\_ElementSettingData  
762               –   association with a value of 1 (Is Next)
  - 763               –   at most one configuration that is referenced by an instance of the CIM\_ElementSettingData  
764               –   association with a value of 3 (Is Next For Single Use) for the IsNext property. This configu-  
765               –   ration shall be given preference over one that is designated with a value of 1 (Is Next).

## 766   **7.4   Profile Registration**

### 767   **7.4.1   This Profile (Virtual System Profile)**

768           The implementation of this profile – the *Virtual System Profile* - shall be indicated by an instance of the  
769           CIM\_RegisteredProfile class in the CIM Interop namespace. Each instance of the CIM\_ComputerSystem  
770           class that represents a virtual system manageable through this profile shall be a central instance of this  
771           profile by associating it to the instance of the CIM\_RegisteredProfile class through an instance of the  
772           CIM\_ElementConformsToProfile association.

### 773   **7.4.2   Scoped DMTF Management Profiles**

774           For a scoped DMTF management profile the following conditions shall be met:

- 775           •   The instance of the CIM\_RegisteredProfile class that represents the implementation of this pro-  
776           –   file - the *Virtual System Profile* - and instances of the CIM\_RegisteredProfile class that repre-  
777           –   sent an implementation of the scoped DMTF management profile shall be associated through  
778           –   instances of the CIM\_ReferencedProfile association.

- 779       • One of the following conditions shall be met:
- 780       a) Instances of the CIM\_ElementConformsToProfile association shall associate any central  
781       instance of the scoped DMTF management profile that is associated to the central instance  
782       of this profile through the CIM\_SystemDevice association, and the instance of the CIM\_Re-  
783       gisteredProfile class that represents an implementation of the scoped DMTF management  
784       profile.
- 785       b) No instances of the CIM\_ElementConformsToProfile association shall associate any cen-  
786       tral instance of the scoped DMTF management profile that is associated to the central in-  
787       stance of this profile through the CIM\_SystemDevice association, and the instance of the  
788       CIM\_RegisteredProfile class that represents an implementation of the scoped DMTF man-  
789       agement profile.

## 790   **7.5   Capabilities**

### 791   **7.5.1.1   CIM\_EnabledLogicalElementCapabilities.RequestedStatesSupported Property**

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

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

## 799   **7.6   Client State Management (Conditional)**

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

802   Client state management is conditional: If the instance of the CIM\_ComputerSystem class that represents  
803   a virtual system is associated through the CIM\_ElementCapabilities association to an instance of the  
804   CIM\_EnabledLogicalElementCapabilities class where the value the RequestedStatesSupported property  
805   is a non-empty array, then client state management is supported for that virtual system.

806   If client state management is supported, an implementation shall do all of the following:

- 807       • implement the CIM\_EnabledLogicalElementCapabilities class according to section 7.5.1.1 to in-  
808       dicate the availability of client state management support, and the set of state transitions that  
809       are applicable
- 810       • implement method RequestStateChange( )
- 811       • if it implements the *Power State Management Profile* for virtual systems, implement the Re-  
812       questPowerStateChange( ) method

## 813   **7.7   Power State Management (Optional)**

814   An implementation may support the *Power State Management Profile*; the *Power State Management Pro-*  
815   *file* specifies

- 816       • how to indicate that the Power State Management Profile is supported
- 817       • how to implement the CIM\_PowerManagementService class and the CIM\_Associated-  
818       PowerManagementService association

819   If the observation of power states is supported as specified by the *Power State Management Profile*, then  
820   the observation of virtual system states as defined in section 7.1.1 and the observation of virtual system  
821   state transitions as defined in section 7.1.2 shall also be supported. If power state management is sup-

822 ported as specified by the *Power State Management Profile*, then client state management as specified in  
823 section 8.1.1 shall also be supported.

824 NOTE The support of the *Power State Management Profile* in the context of virtual systems is intended to support  
825 clients that use facilities specified by the *Power State Management Profile* in preference of facilities specified in the  
826 *Virtual System Profile*. For example, such clients may use the CIM\_AssociatedPowerManagementService.Power-  
827 State property in favor of the CIM\_ComputerSystem.EnabledState property to determine the virtual system state, or  
828 may use the CIM\_PowerManagementService.RequestPowerStateChange( ) method in favor of the CIM\_Enabled-  
829 LogicalElement.RequestStateChange( ) method to effect virtual system state transitions.

### 830 **7.7.1 CIM\_AssociatedPowerManagementService.PowerState Property (Conditional)**

831 If state management is supported (see section 7.5.1.1) and the *Power State Management Profile* is sup-  
832 ported for a virtual system, then this profile – the *Virtual System Profile* - specifies additional rules:

- 833 • The CIM\_AssociatedPowerManagementService association shall be used to convey the virtual  
834 system state in addition to the CIM\_ComputerSystem.EnabledState property. In this case, the  
835 PowerState property shall contain a value that corresponds to the virtual system state as de-  
836 fined in Table 2. For example, if the virtual system state is “Active”, then the PowerState prop-  
837 erty shall have a value of 2 (On).
- 838 • A client preferring to use mechanisms defined by the *Power State Management Profile* may  
839 translate the value of the PowerState property of an instance of the CIM\_AssociatedPower-  
840 ManagementService association that is referring to an instance of the CIM\_ComputerSystem  
841 class representing a virtual system by translating that value according to Table 2. For example,  
842 if the PowerState property has a value of 2 (On), then a client shall conclude that the virtual sys-  
843 tem state is “Active”.

## 844 **8 Methods**

845 This section details the requirements for supporting intrinsic CIM operations and extrinsic methods for the  
846 CIM elements defined by this profile.

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

### 848 **8.1 Extrinsic Methods**

#### 849 **8.1.1 CIM\_ComputerSystem.RequestStateChange( ) (Conditional)**

850 If client state management is supported (see section 7.6), the RequestStateChange( ) method shall be  
851 implemented.

852 Detailed requirements for the CIM\_ComputerSystem.RequestStateChange( ) method are specified in  
853 Table 4.

854 **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.

855 For return code values, see the CIM Schema description of this method in the CIM\_EnabledLogical-  
856 Element class.

857 No standard messages are defined.

### 858 **8.1.2 CIM\_PowerManagementService.RequestPowerStateChange( ) (Conditional)**

859 If client state management is supported (see section 7.5.1.1) and the *Power State Management Profile* is  
860 supported for a virtual system, then this profile - the *Virtual System Profile* - specifies that the CIM\_Power-  
861 ManagementService.RequestPowerStateChange( ) method shall be implemented, enabling the request  
862 of virtual system state transitions through this alternative method. Detailed requirements for the  
863 CIM\_PowerManagementService.RequestPowerStateChange( ) method are specified in Table 5.

864 **Table 5 – CIM\_PowerManagementService.RequestPowerStateChange( ) Method: Parameters**

Qualifiers	Name	Type	Description/Values
IN	PowerState	uint16	See section 8.1.2.1.
IN	ManagedElement	CIM_ComputerSystem REF	See section 8.1.2.2.
IN	Time	datetime	See section 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.

865 For return code values, see the CIM Schema description of this method in the CIM\_PowerManagement-  
866 Service class.

867 No standard messages are defined.

#### 868 **8.1.2.1 PowerState Parameter**

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

870 The translation defined by Table 3 shall be used to interpret values of the PowerState parameter of the  
871 CIM\_PowerManagementService.RequestPowerStateChange( ) method as a request for a virtual system  
872 state transition. For example, if value “On” is specified on a particular power state change request for a  
873 virtual system, then an “Activate” state transition shall be performed.

#### 874 **8.1.2.2 ManagedElement Parameter**

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

#### 877 **8.1.2.3 Time Parameter (Optional)**

878 The Time parameter may indicate the point in time that the power state should be set.

## 879 **8.2 Profile Conventions for Operations**

880 Support for operations for each profile class (including associations) is specified in the following subsec-  
881 tions. Each subsection includes either a statement “All operations in the default list in section 8.2 are sup-  
882 ported as described by DSP0200 version 1.2.0” or a table listing all of the operations that are not sup-  
883 ported by this profile or where the profile requires behavior other than that described by DSP0200 version  
884 1.2.0.

## Virtual System Profile

885 The default list of operations is as follows:

- 886 • GetInstance
- 887 • Associators
- 888 • AssociatorNames
- 889 • References
- 890 • ReferenceNames
- 891 • EnumerateInstances
- 892 • EnumerateInstanceNames

893 A compliant implementation shall support all of the operations in the default list for each class, unless the  
894 "Requirement" column states something other than *Mandatory*.

895 This profile defines methods in terms of DSP0200 version 1.2.0.

### 896 **8.2.1 CIM\_ComputerSystem**

897 All operations in the default list in section 8.2 are supported as described by DSP0200 version 1.2.0.

### 898 **8.2.2 CIM\_ConcreteJob**

899 All operations in the default list in section 8.2 are supported as described by DSP0200 version 1.2.0.

### 900 **8.2.3 CIM\_ElementSettingData**

901 Table 6 lists operations that either have special requirements beyond those from DSP0200 version 1.2.0  
902 or shall not be supported.

903 **Table 6 – Operations: CIM\_ElementSettingData**

Operation	Requirement	Messages
Associators	Unspecified	None
AssociatorNames	Unspecified	None
References	Unspecified	None
ReferenceNames	Unspecified	None
EnumerateInstances	Unspecified	None
EnumerateInstanceNames	Unspecified	None

### 904 **8.2.4 CIM\_EnabledLogicalElementCapabilities**

905 All operations in the default list in subsection 8.2 are supported as described by DSP0200 version 1.2.0.

### 906 **8.2.5 CIM\_ReferencedProfile**

907 Table 7 lists operations that either have special requirements beyond those from DSP0200 version 1.2.0  
908 or shall not be supported.

909 **Table 7 – Operations: CIM\_ReferencedProfile**

Operation	Requirement	Messages
-----------	-------------	----------



Operation	Requirement	Messages
Associators	Unspecified	None
AssociatorNames	Unspecified	None
References	Unspecified	None
ReferenceNames	Unspecified	None
EnumerateInstances	Unspecified	None
EnumerateInstanceNames	Unspecified	None

## 910 **8.2.6 CIM\_RegisteredProfile**

911 All operations in the default list in section 8.2 are supported as described by DSP0200 version 1.2.0.

## 912 **8.2.7 CIM\_VirtualSystemSettingData**

913 All operations in the default list in section 8.2 are supported as described by DSP0200 version 1.2.0.

## 914 **8.2.8 CIM\_VirtualSystemSettingDataComponent**

915 Table 8 lists operations that either have special requirements beyond those from DSP0200 version 1.2.0  
916 or shall not be supported.

917 **Table 8 – Operations: CIM\_VirtualSystemSettingDataComponent**

Operation	Requirement	Messages
Associators	Unspecified	None
AssociatorNames	Unspecified	None
References	Unspecified	None
ReferenceNames	Unspecified	None
EnumerateInstances	Unspecified	None
EnumerateInstanceNames	Unspecified	None

918

## 919 **9 Use Cases**

920 The following use cases and object diagrams illustrate use of this profile - the *Virtual System Profile*. They  
921 are for informational purposes only and do not introduce behavioral requirements for implementations of  
922 the profile.

### 923 **9.1 Virtual System Detection and Inspection**

924 This set of use cases describes how a client can

- 925 • discover virtual systems
- 926 • determine the state and properties of a virtual system
- 927 • determine the “Defined” virtual system configuration
- 928 • determine the virtual system structure
- 929 • determine resource type support
- 930 • detect and inspect the boot configuration for the virtual system

### 931 **9.1.1 Discover Conformant Virtual Systems Using SLP**

932 This use case describes how to locate instances of the CIM\_ComputerSystem class that represent virtual  
933 systems that are central instances of this profile – the *Virtual System Profile*. This is a two-step process:

- 934 1) The service location protocol (SLP) is used to locate CIM object managers (CIMOMs) where  
935 this profile is implemented. A CIMOM using SLP facilities provides information about itself to  
936 SLP in form of an SLP service template. The service template may contain information about  
937 the set of DMTF management profiles that is implemented at the CIMOM.
- 938 2) Normal CIM enumeration and association resolution is used to find instances of the CIM\_Com-  
939 puterSystem class that represent central instances of this profile.

940 **Assumption:** This profile is registered at least one CIMOM that maintains a registration with a SLP Direc-  
941 tory Agent; the registration included information about registered DMTF management profiles. The client  
942 is able to make SLP calls and invoke intrinsic CIM operations.

943 A client can locate instances of the CIM\_ComputerSystem class that represent virtual systems that are  
944 central instances of this profile as follows:

- 945 1) The client invokes the SLPFindSrvs( ) SLP function:  
946 – The value of the srvtpe parameter is set to “service:wbem”  
947 – The value of the scopelist parameter is set to “default”  
948 – The value of the filter parameter is set to “(RegisteredProfilesSupported=DMTF:Virtual  
949 System Profile)”

950 The result is a list of URLs that identify CIMOMs where this profile – the Virtual System Profile –  
951 is implemented.

- 952 2) The client contacts each of the CIMOMs and enumerates or queries the CIM\_RegisteredProfile  
953 class.
  - 954 • As input, the client needs to use the address information of one server obtained in step 1)  
955 and issue the intrinsic EnumerateInstanceNames( ) CIM operation on the CIM\_Registered-  
956 Profile class. Alternatively, the client may issue the intrinsic ExecuteQuery CIM operation  
957 and specify a where clause that, for example, limits the value ranges for the Registered-  
958 Name and RegisteredVersion properties of the CIM\_RegisteredProfile class.
  - 959 • As a result, the client receives a list of references to instances of the CIM\_RegisteredPro-  
960 file class that represent implementations of this profile - the *Virtual System Profile* - at the  
961 intended target location. On a query operation this list already is limited according to the  
962 input selection criteria.
- 963 3) The client selects one reference and resolves the CIM\_ElementConformsToProfile association  
964 from the instance of the CIM\_RegisteredProfile class to instances of the CIM\_ComputerSystem  
965 class.
  - 966 • As input, the client needs to provide the reference to an instance of the CIM\_Registered-  
967 Profile class that was selected from the result set obtained in step 2.
  - 968 • As a result, the client receives a list of references referencing instances of the CIM\_  
969 ComputerSystem class that represents virtual systems.

970 **Result:** The result is that the client knows a set of references referencing instances of the CIM\_Compu-  
971 terSystem class that represent virtual systems that are central instances of this profile.

### 972 **9.1.2 Determine a Virtual System’s State and Other Properties**

973 **Assumption:** The client has a reference referring to an instance of the CIM\_ComputerSystem class that  
974 represents a virtual system that is a central instance of this profile.

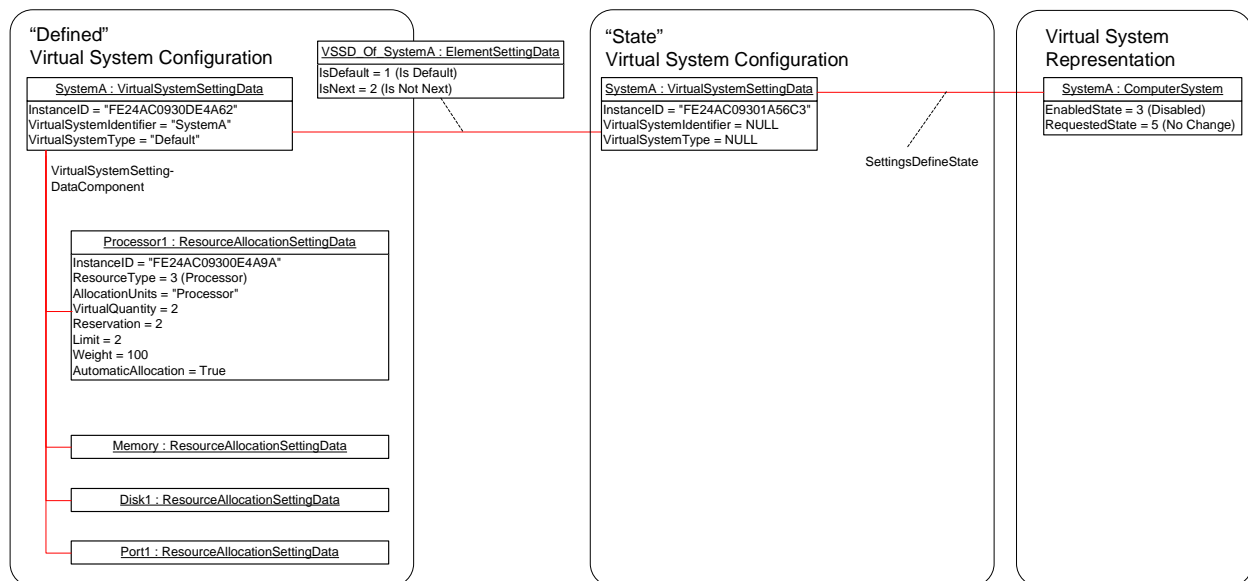
975 The client can determine the virtual system’s state and other properties as follows:

- 976 1) The client calls the intrinsic GetInstance( ) CIM operation with the InstanceName parameter ref-  
 977 referencing the instance of the CIM\_ComputerSystem class that represents the virtual system as  
 978 the input parameter. As a result the client receives an instance of the CIM\_ComputerSystem  
 979 class that describes the virtual system.
- 980 2) The client uses the value of the EnabledState property to determine the virtual system state ac-  
 981 cording to the translation rules specified in section 7.1.1.

982 **Result:** The client knows the property set defined by the CIM\_ComputerSystem class describing the af-  
 983 fected virtual system, in particular the virtual system state. Many virtual system properties and in particu-  
 984 lar the virtual system state may change any time; consequently, the result only describes the virtual sys-  
 985 tem at the moment it was provided by the instrumentation.

### 986 9.1.3 Determine the “Defined” Virtual System Configuration

987 **Assumption:** The client has a reference referring to an instance of the CIM\_ComputerSystem class that  
 988 represents a virtual system that is a central instance of this profile. The virtual system is assumed to be  
 989 configured as shown in Figure 5 with the “Virtual System Configuration (“Defined”)” configuration. In this  
 990 example the implementation applies the dual-configuration implementation approach (section 7.3.4.1) as  
 991 described in Annex B.



992

993

Figure 5 – Sample Virtual System Configuration

994 The client can determine the “Defined” virtual system configuration as follows:

- 995 1) The client resolves the CIM\_SettingsDefineState association from the instance of the  
 996 CIM\_ComputerSystem class representing the virtual system to the top-level instance of the  
 997 CIM\_VirtualSystemSettingData class in the “State” Virtual System configuration.
- 998 2) The client resolves the CIM\_ElementSettingData association from the “State” instance of the  
 999 CIM\_VirtualSystemSettingData class that represents the virtual aspects of the virtual system to  
 1000 instances of the CIM\_VirtualSystemSettingData class with the constraint that the CIM\_Elemen-  
 1001 tSettingData.IsDefault property has a value of 2 (IsDefault). The result is a reference referring to  
 1002 an instance of the CIM\_VirtualSystemSettingData class that represents the top-level object of  
 1003 the desired virtual system configuration.
- 1004 3) The client obtains the referenced instance of the CIM\_VirtualSystemSettingData class using the  
 1005 intrinsic getInstance( ) CIM operation and analyzes its properties. For example, the client might  
 1006 analyze the VirtualSystemIdentifier property, which reflects the (end-user interpretable) name

## Virtual System Profile

1007 used for the virtual system ("SystemA" in Figure 5), or the VirtualSystemType property, which  
 1008 reflects a particular virtual system type that the virtualization platform assigned for the respec-  
 1009 tive virtual system ("Default" in Figure 5). Note that the InstanceID property contains an opaque  
 1010 ID for the instance; the structure of InstanceID values is implementation dependent and not  
 1011 known to clients.

1012 4) The client resolves the CIM\_VirtualSystemSettingDataComponent association from the instance  
 1013 of the CIM\_VirtualSystemSettingData class to instances of the CIM\_ResourceAllocationSetting-  
 1014 Data class.

1015 5) The client obtains instances of the CIM\_ResourceAllocationSettingData class using the intrinsic  
 1016 getInstance() CIM operation and analyzes properties of these instances. For example, the cli-  
 1017 ent might analyze the Reservation property. The Reservation property reflects the amount of  
 1018 host resource that is allocated for the virtual resource while the virtual system is instantiated.

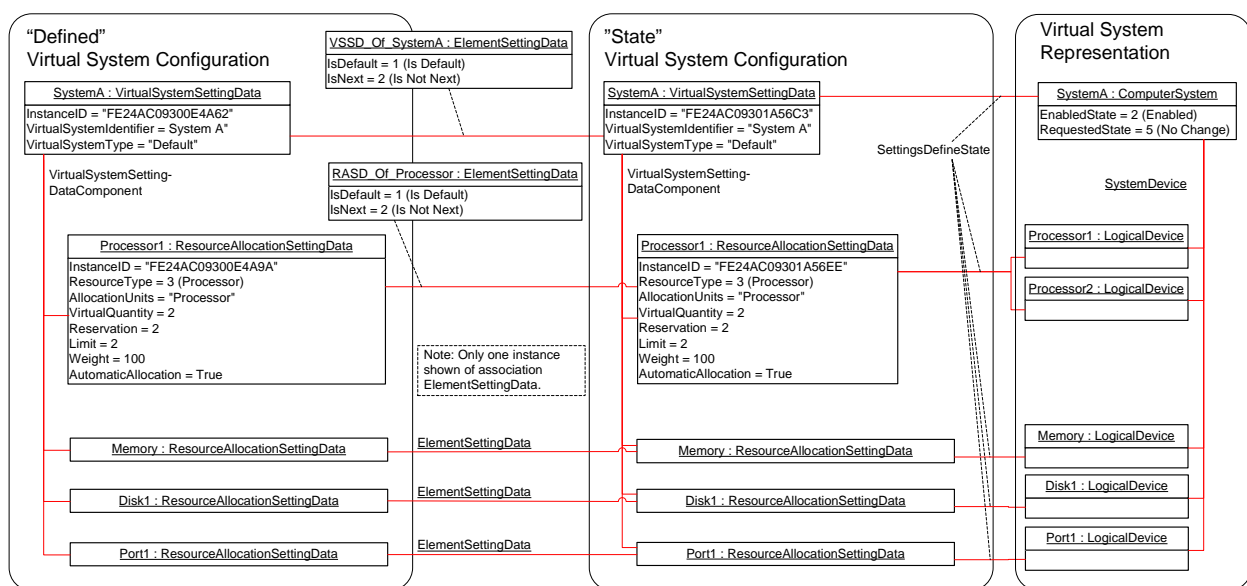
1019 **Result:** The client knows the virtual system configuration in terms of one instance of the CIM\_Virtual-  
 1020 SystemSettingData class and a set of aggregated instances of the CIM\_ResourceAllocationSettingData  
 1021 class.

### 1022 9.1.4 Determine the Virtual System Structure

1023 **Assumption:** The client has a reference referring to an instance of the CIM\_ComputerSystem class that  
 1024 represents a virtual system that is a central instance of this profile.

- 1025 • The virtual system configuration is assumed to be the same as for use case described in sec-  
 1026 tion 9.1.3.
- 1027 • The virtual system is assumed to be in the "Active" state.
- 1028 • The virtual system is assumed to be structured as shown in Figure 6.
- 1029 • The set of attributes for each logical resource is not shown; this set of attributes depends on the  
 1030 type of logical resource and may be specified in the context of respective resource-type-specific  
 1031 DMTF management profiles.

1032 To avoid cluttering the diagram, an instance of the CIM\_ElementSettingData association between the  
 1033 "Defined" and the "State" instance of the CIM\_ResourceAllocationSettingData class is shown for proces-  
 1034 sor configurations only.



1035

1036

**Figure 6 – Sample Virtual System in "Active" State**

1037 A client can determine the virtual system structure as follows:

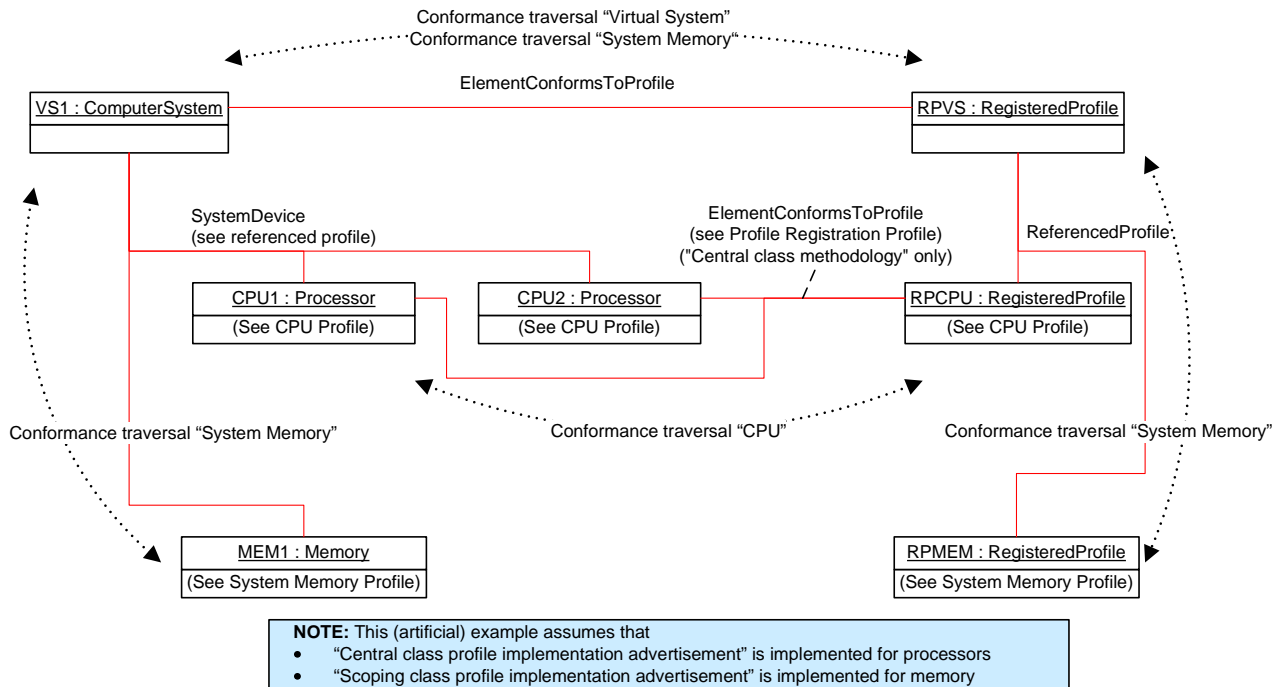
- 1038 1) The client may apply the use case described in section 9.1.2 to obtain state information and  
1039 other properties of the CIM\_ComputerSystem instance that represents the virtual system.
- 1040 2) The client may apply the use case described in section 9.1.3 to obtain information about the vir-  
1041 tual system configuration.
- 1042 3) The client resolves the CIM\_SystemDevice association from the instance of the CIM\_Computer-  
1043 System class that represents the virtual system to instances of the CIM\_LogicalDevice class.
- 1044 4) The client obtains instances of the CIM\_LogicalDevice class that were returned in step 3) and  
1045 analyzes properties of interest.

1046 **Result:** The client knows the virtual system structure as expressed through the virtual system configura-  
1047 tions (“Defined” and “State”) and through the set of objects representing the virtual system and its com-  
1048 ponents.

1049 **9.1.5 Determine Resource Type Support**

1050 This subset of use cases describes how to determine whether implementations of resource-type-specific  
1051 DMTF management profiles are present for logical devices in scope of a virtual system. Examples are the  
1052 *CPU Profile* for the management of virtual processors with the CIM\_Processor class as the central class,  
1053 or the *System Memory Profile* for the management of virtual memory with the CIM\_Memory class as the  
1054 central class.

1055 The *Profile Registration Profile* defines how an implementation of a DMTF management profile adver-  
1056 tises conformance to the DMTF management profile. For example, Figure 7 shows an instance of the  
1057 CIM\_ComputerSystem class named VS1 that is associated to an instance of the CIM\_RegisteredProfile  
1058 class named RPVS.



1059  
1060

1061 **Figure 7 – Instance Diagram: Profile Conformance of Scoped Resources**

1062 If DMTF management profiles for scoped resources are supported, the *Profile Registration Profile* speci-  
1063 fies to support either the “central class profile implementation advertisement methodology” or the “scoped  
1064 class profile implementation advertisement methodology”.

1065 With the “central class profile implementation advertisement methodology” the approach is straight for-  
1066 ward: Any central instance of a DMTF management profile is associated with the respective instance of  
1067 the CIM\_RegisteredProfile class through the CIM\_ElementConformsToProfile association.

1068 With the “Scoped Class Profile Implementation Advertisement Methodology” the CIM\_ElementConforms-  
1069 ToProfile association is not implemented for scoped profiles and resources; instead, conformance of  
1070 scoped resources to respective scoped DMTF management profiles is implied by the presence of scoped  
1071 instances of the CIM\_RegisteredProfile class.

#### 1072 **9.1.5.1 Determine Resource Type Support of Scoped Resources (Central Class Methodology)**

1073 **Assumption:** The client has a reference referring to an instance of the CIM\_ComputerSystem class that  
1074 represents a virtual system that is a central instance of this profile; see section 9.1.1. A situation as  
1075 shown in Figure 7 for processors is assumed.

1076 The first part of this use case determines the profile implementation advertisement methodology for proc-  
1077 essors.

- 1078 1) The client resolves the CIM\_ElementConformsToProfile association to locate associated in-  
1079 stances of the CIM\_RegisteredProfile class, invoking the intrinsic AssociatorNames( ) CIM op-  
1080 eration as follows:
  - 1081 – The value of the ObjectName parameter references the instance of the CIM\_Computer-  
1082 System class.
  - 1083 – The value of the AssocClass parameter is set to “CIM\_ElementConformsToProfile”.
  - 1084 – The value of the ResultClass parameter is set to “CIM\_RegisteredProfile”.
  - 1085 – The result is a list of references referring to instances of the CIM\_RegisteredProfile class  
1086 representing implementations of this profile; if the operation is successful, the size of the  
1087 result set is 1.
- 1088 2) The client resolves the CIM\_ReferencedProfile association to locate scoped instances of the  
1089 CIM\_RegisteredProfile class, invoking the intrinsic Associators( ) CIM operation as follows:
  - 1090 – The value of the ObjectName parameter is set to the reference referring to the instance of  
1091 the CIM\_RegisteredProfile class obtained in step 1).
  - 1092 – The value of the AssocClass parameter is set to “CIM\_ReferencedProfile”.
  - 1093 – The value of the ResultClass parameter is set to “CIM\_RegisteredProfile”.
  - 1094 – The result is a list of instances of the CIM\_RegisteredProfile class representing implemen-  
1095 tations of scoped profiles.
- 1096 3) The client iterates over the list obtained in step 2), selecting only instances where the Regis-  
1097 teredName property has a value of “CPU Profile”.
  - 1098 – The result is a list of instances of the CIM\_RegisteredProfile class that represents imple-  
1099 mentations of scoped profiles implementing the *CPU Profile* .
- 1100 4) The client resolves the CIM\_ElementConformsToProfile association for each of the instances of  
1101 the CIM\_RegisteredProfile class from step 3) to locate at least one associated instance of the  
1102 CIM\_Processor class, invoking the intrinsic Associators( ) CIM operation as follows:
  - 1103 – The value of the ObjectName parameter is set to the reference taken from the instance of  
1104 the CIM\_RegisteredProfile class obtained in step 3).
  - 1105 – The value of the AssocClass parameter is set to “CIM\_ReferencedProfile”.
  - 1106 – The value of the ResultClass parameter is set to “CIM\_Processor”.

- 1107           – The result is a list of instances of the CIM\_Processor class that are central instances of the  
1108           scoped *CPU Profile*; the list may be empty.

1109 If for any of the results from step 4) at least one instance of the CIM\_Processor class was detected, then  
1110 the central class profile implementation advertisement methodology is applied by the implementation with  
1111 respect to implementations of the *CPU Profile*; this is the case in this example. If no such instances were  
1112 detected, then the scoping class profile implementation advertisement methodology would have been  
1113 applied.

1114 At this point the client has validated that the *CPU Profile* is implemented as a scoped profile of this profile,  
1115 and that the central class profile implementation advertisement methodology is applied by the implemen-  
1116 tation with respect to the *CPU Profile*.

1117 In the second part of this use case it is now the responsibility of the client for any detected scoped in-  
1118 stance of the CIM\_Processor class to validate that the *CPU Profile* is indeed implemented. The use case  
1119 describes how to locate such instances, and perform the validation:

- 1120           5) Client resolves the CIM\_SystemDevice association from the central instance to associated vir-  
1121           tual resources, invoking the intrinsic AssociatorNames( ) CIM operation as follows:
- 1122           – The value of the ObjectName parameter is set referring to the instance of the CIM\_Compu-  
1123           terSystem class.
  - 1124           – The value of the AssocClass parameter is set to “CIM\_SystemDevice”.
  - 1125           – The value of the ResultClass parameter is set to “CIM\_Processor”.
  - 1126           – The result is a list of references referring to scoped instances of the CIM\_Processor class  
1127           representing virtual processors.
- 1128           6) For each reference returned by step 5) the client resolves the CIM\_ElementConformsToProfile  
1129           association to locate associated instances of the CIM\_RegisteredProfile class, invoking the in-  
1130           trinsic Associators( ) CIM operation as follows:
- 1131           – The value of parameter ObjectName is set referring to an instance of the CIM\_Processor  
1132           class.
  - 1133           – The value of the AssocClass parameter is set to “CIM\_ElementConformsToProfile”.
  - 1134           – The value of the ResultClass parameter is set to “CIM\_RegisteredProfile”.
  - 1135           – The result is a list of instances of the CIM\_RegisteredProfile class; if the operation is suc-  
1136           cessful, the size of the list is either 0 or 1. A size of 1 indicates that a version of the *CPU*  
1137           *Profile* is implemented for the particular processor; a size of 0 indicates that the *CPU Pro-*  
1138           *file* is not implemented for the particular processor.

1139 **Result:** The client knows the set of scoped instances of the CIM\_Processor class that represents proces-  
1140 sors of the assumed virtual system, and whether the instances are central instances of the *CPU Profile*,  
1141 that is, whether the *CPU Profile* is implemented in the context of these instances.

#### 1142 **9.1.5.2 Determine Resource Type Support of Scoped Resources (Scoping Class Methodology)**

1143 **Assumption:** The client has a reference referring an instance of the CIM\_ComputerSystem class that  
1144 represents a virtual system that is a central instance of this profile; see section 9.1.1. A situation as  
1145 shown in Figure 7 for the “Memory” resource type is assumed.

1146 The first part of this use case determines the profile implementation advertisement methodology for  
1147 memory.

- 1148           1) The client resolves the CIM\_ElementConformsToProfile association to locate associated in-  
1149           stances of the CIM\_RegisteredProfile class, invoking the intrinsic AssociatorNames( ) CIM op-  
1150           eration as follows:

## Virtual System Profile

- 1151           – The value of the ObjectName parameter is set to the reference referring to the instance of  
1152           the CIM\_ComputerSystem class.
- 1153           – The value of the AssocClass parameter is set to “CIM\_ElementConformsToProfile”.
- 1154           – The value of the ResultClass parameter is set to “CIM\_RegisteredProfile”.
- 1155           – The result is a list of references referring to instances of the CIM\_RegisteredProfile class  
1156           representing implementations of this profile; if the operation is successful, the size of the  
1157           result set is 1.
- 1158           2) The client resolves the CIM\_ReferencedProfile association to locate scoped instances of the  
1159           CIM\_RegisteredProfile class, invoking the intrinsic Associators( ) CIM operation as follows:
- 1160           – The value of parameter ObjectName is set to the reference referring to the instance of the  
1161           CIM\_RegisteredProfile class obtained in step 1).
- 1162           – The value of the AssocClass parameter is set to “CIM\_ReferencedProfile”.
- 1163           – The value of the ResultClass parameter is set to “CIM\_RegisteredProfile”.
- 1164           – The result is a list of instances of the CIM\_RegisteredProfile class that represent imple-  
1165           mentations of scoped profiles.
- 1166           3) The client iterates over the list obtained in step 2), selecting only instances where the Regis-  
1167           teredName property has a value of “System Memory Profile”.
- 1168           – The result is a list of instances of the CIM\_RegisteredProfile class that represents imple-  
1169           mentations of scoped profiles implementing the *System Memory Profile*.
- 1170           4) The client resolves the CIM\_ElementConformsToProfile association for each of the instances of  
1171           the CIM\_RegisteredProfile class from step 3) to locate at least one associated instance of the  
1172           CIM\_Memory class, invoking the intrinsic Associators( ) CIM operation as follows:
- 1173           – The value of the ObjectName parameter is set to the reference taken from the instance of  
1174           the CIM\_RegisteredProfile class obtained in step 3).
- 1175           – The value of the AssocClass parameter is set to “CIM\_ElementConformsToProfile”.
- 1176           – The value of the ResultClass parameter is set to “CIM\_Memory”.
- 1177           – The result is a list of instances of the CIM\_Memory class that are central instances of the  
1178           scoped *System Memory Profile*. The list may be empty.

1179           If for any of the results from step 4) no instance of the CIM\_Memory class was detected, then the scoping  
1180           class profile implementation advertisement methodology is applied by the implementation with respect to  
1181           implementations of the *System Memory Profile*; this is the case in this example. If any such instances  
1182           were detected, then the central class profile implementation advertisement methodology would have been  
1183           applied.

1184           At this point the client has validated that the *System Memory Profile* is implemented as a scoped profile of  
1185           this profile, and that the scoping class profile implementation advertisement methodology is applied by  
1186           the implementation with respect to the *System Memory Profile*.

1187           In the second part of this use case the client now may assume for any detected scoped instance of the  
1188           CIM\_Memory class that the *System Memory Profile* is implemented. The use case describes how to lo-  
1189           cate such instances:

- 1190           5) The client resolves the CIM\_SystemDevice association from the central instance to associated  
1191           virtual resources, invoking the intrinsic AssociatorNames( ) CIM operation as follows:
- 1192           – The value of the ObjectName parameter is set to the reference referring to the instance of  
1193           the CIM\_ComputerSystem class.
- 1194           – The value of the AssocClass parameter is set to “CIM\_SystemDevice”.
- 1195           – The value of the ResultClass parameter is set to “CIM\_Memory”.



- 1196           – The result is a list of references referring to scoped instances of the CIM\_Memory class  
1197           that represents virtual memory.

1198 **Result:** The client knows the set of scoped instances of the CIM\_Memory class that represents memory  
1199 in the assumed virtual system, and that these are central instances of the *System Memory Profile*.

## 1200 9.1.6 Determine the Next Boot Configuration

1201 **Assumption:** The client has a reference referring to an instance of the CIM\_ComputerSystem class that  
1202 represents a virtual system that is a central instance of this profile.

- 1203           1) The client resolves the CIM\_ElementSettingData association to find instances of the CIM\_Boot-  
1204           ConfigSetting class that describe the boot configuration of the virtual system, invoking the intrinsic  
1205           References( ) CIM operation as follows:

- 1206           – the ObjectName parameter referring to the instance of the CIM\_ComputerSystem class  
1207           that represents the virtual system
- 1208           – the ResultClass parameter set to a value of “CIM\_ElementSettingData”
- 1209           – the Role parameter set to a value of “ManagedElement”

1210           The result of this step is a set of instances of the CIM\_ElementSettingData association.

- 1211           2) The client analyzes the result set of the previous step and selects that instance of the CIM\_Ele-  
1212           mentSettingData association that has the IsNext property set to a value of 3 (Is Next For Single  
1213           Use) or, if there is no such instance, that has the IsNext property set to a value of 1 (Is Next).

1214           The result of this step is an instance of the CIM\_ElementSettingData association where the Set-  
1215           tingData property refers to the instance of the CIM\_BootConfigSetting class that is used for the  
1216           next boot process.

- 1217           3) The client obtains the instance of the CIM\_BootConfigSetting class, using the intrinsic GetIn-  
1218           stance( ) CIM operation with the InstanceName parameter referring to that instance.

1219 **Result:** The client knows the boot configuration that is used during the next “Activate” virtual system state  
1220 transition.

## 1221 9.2 Virtual System Operation

1222 This set of use cases describes how a client can perform basic operations on virtual system, like activat-  
1223 ing, deactivating, pausing or resuming a virtual system.

### 1224 9.2.1 Change Virtual System State

1225 This use case is a generic use case that describes the generic procedure to effect a virtual system state  
1226 change. A number of use cases follow that describe the effects on objects and association instances rep-  
1227 resenting virtual systems, their components, and relationships as defined in this profile.

1228 **Assumption:** The client has a reference referring to an instance of the CIM\_ComputerSystem class that  
1229 represents a virtual system that is a central instance of this profile. The client intends to effect a virtual  
1230 system state transition. (For a list of virtual system state transitions defined by this profile, see Table 3.)

- 1231           1) The client applies the rules outlined in section 7.1.2 to determine a value for the Requested-  
1232           State parameter of the CIM\_EnabledLogicalElement.RequestStateChange( ) method that des-  
1233           ignates the intended state transition.
- 1234           2) The client resolves the CIM\_ElementCapabilities association from the instance of the  
1235           CIM\_ComputerSystem class to find the instance of the CIM\_EnabledLogicalElementCapabilities  
1236           class that describes capabilities of the virtual system; if there is no associated instance of  
1237           CIM\_EnabledLogicalElementCapabilities, then client state management is not supported for the  
1238           virtual system.

- 1239 3) The client analyzes the RequestedStatesSupported property to check whether it contains an  
1240 element that designates the intended state transition as determined by step 1). If the Re-  
1241 requestedStatesSupported property does not contain a respective element, then the intended  
1242 state transition is not supported for the virtual system as a client state management activity.  
1243 This may be a temporary situation. Also it might still be possible to effect the state transition us-  
1244 ing other means, such as the native capabilities of the virtualization platform.
- 1245 4) The client invokes the RequestStateChange method on the instance of the CIM\_Computer-  
1246 System class that represents the virtual system, using a value for the RequestedState param-  
1247 eter as determined in step 1).
- 1248 5) The client checks the return code.
- 1249 – If the return code is zero, the virtual system state transition was performed as requested.
  - 1250 – If the return code is 1, the RequestStateChange method is not supported by the implemen-  
1251 tation. This should not occur if the checks above were performed.
  - 1252 – If the return code is 2, an error occurred.
  - 1253 – If the return code is 0x1000, the implementation has decided to perform the state transition  
1254 as an asynchronous task. The client may monitor progress by analyzing the instance of the  
1255 CIM\_ConcreteJob class returned through the Job parameter.

1256 If the operation is performed as an asynchronous task, the client may obtain intermediate instances of the  
1257 CIM\_ComputerSystem class representing the virtual system (see section 9.1.2). These would show val-  
1258 ues for the EnabledState and RequestedState properties that indicate an ongoing state transition. For  
1259 example, during an “Activate” virtual system state transition the EnabledState property might show a  
1260 value of 10 (Starting) and the RequestedState property might have a value of 2 (Enabled).

1261 **Result:** The virtual system performs the intended virtual system state transition. The client may next ob-  
1262 tain the actual virtual system state by, for example, following the procedures outlined the use case in sec-  
1263 tion 9.1.2.

## 1264 9.2.2 Activate Virtual System

1265 **Assumption:** This use case is predicated on the assumptions described in section 9.2.1 and the same  
1266 starting point described in section 9.1.3.

- 1267 1) The client applies the steps in the use case described in section 9.2.1 to perform an “Activate”  
1268 transition, for example using a value of 2 (Enabled) for the RequestedState parameter.
- 1269 2) The client verifies that the operation was executed successfully, making sure that either a return  
1270 code of 0 results or, if the state change is performed as an asynchronous task, by checking that  
1271 the result of the respective instance of the CIM\_ConcreteJob class indicates a successful com-  
1272 pletion.

1273 If the operation is performed as an asynchronous task, a client may obtain intermediate elements of the  
1274 virtual system structure (see section 9.1.4). This structure might be incomplete during the state transition.  
1275 For example, if a client resolves associations to instances of the CIM\_LogicalDevice class that represent  
1276 the virtual resources as shown in Figure 6 (such as, for example, the CIM\_SystemDevice association  
1277 from the instance of the CIM\_ComputerSystem class representing the virtual system, or the CIM\_Ele-  
1278 mentSettingData association from the instance of the CIM\_ResourceAllocationSettingData class repre-  
1279 senting the virtual resource allocation), then the client might observe that some virtual resources are al-  
1280 ready allocated and represented through instances of the CIM\_LogicalDevice class, while other virtual  
1281 resources are not yet allocated to the virtual system and not yet represented through instances of the  
1282 CIM\_LogicalDevice class.

1283 **Result:** The virtual system is in the “Active” state as shown in the use case described in Figure 6 and in  
1284 section 9.1.4.

1285 **10 CIM Elements**

1286 Table 9 lists CIM elements that are defined or specialized for this profile. Each CIM element shall be im-  
 1287 plemented as described in Table 9. The CIM Schema descriptions for any referenced element and its  
 1288 sub-elements apply.

1289 Sections 7 (“Implementation”) and 8 (“Methods”) may impose additional requirements on these elements.

1290 **Table 9 – CIM Elements: Virtual System Profile**

Element	Requirement	Notes
<b>Classes</b>		
CIM_AffectedJobElement	Conditional	See section 10.1
CIM_ComputerSystem	Mandatory	See section 10.2
CIM_ConcreteJob	Conditional	See section 10.3
CIM_ElementCapabilities	Conditional	See the <i>Computer System Profile</i> (section 10)
CIM_ElementConformsToProfile	Mandatory	See section 10.4
CIM_ElementSettingData	Mandatory	See section 10.5
CIM_EnabledLogicalElementCapab ilities	Optional	See section 10.6
CIM_PowerManagementService	Optional	See section 10.7
CIM_ReferencedProfile	Conditional	See section 10.8
CIM_RegisteredProfile	Mandatory	See section 10.9
CIM_SettingsDefineState	Mandatory	See section 10.10
CIM_VirtualSystemSettingData	Mandatory	See section 10.11
CIM_VirtualSystemSettingDataCom ponent	Conditional	See section 10.12
<b>Indications</b>		
None defined in this profile		

1291 **10.1 CIM\_AffectedJobElement (Conditional)**

1292 The CIM\_AffectedJobElement association associates an instance of the CIM\_ComputerSystem class  
 1293 representing a virtual system and an instance of the CIM\_ConcreteJob class representing an ongoing  
 1294 virtual system state transition.

1295 Support of the CIM\_AffectedJobElement association is conditional with respect to the support of the  
 1296 CIM\_ConcreteJob class.

1297 Table 10 lists the requirements for this association.

1298 **Table 10 – Association: CIM\_AffectedJobElement**

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

## 1299 10.2 CIM\_ComputerSystem

1300 The use of the CIM\_ComputerSystem class is specialized in the *Computer System Profile* and refined in  
1301 this profile.

1302 The requirements in Table 11 are in addition to those mandated by the *Computer System Profile*.

1303 **Table 11 – Class: CIM\_ComputerSystem**

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

## 1304 10.3 CIM\_ConcreteJob (Conditional)

1305 An implementation shall use an instance of the CIM\_ConcreteJob class to represent an asynchronous  
1306 task.

1307 Support of the CIM\_ConcreteJob class is conditional with respect to the implementation supporting asyn-  
1308 chronous execution of methods.

1309 Table 12 lists requirements for elements of this class.

1310 **Table 12 – Class: CIM\_ConcreteJob**

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

## 1311 10.4 CIM\_ElementConformsToProfile

1312 The CIM\_ElementConformsToProfile association associates an instance of the CIM\_RegisteredProfile  
1313 class representing an implementation of this profile with each instance of the CIM\_ComputerSystem class  
1314 representing a virtual system that is conformant with this profile.

1315 Table 13 lists the requirements for this association.

1316 **Table 13 – Association: CIM\_ElementConformsToProfile**

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

## 1317 10.5 CIM\_ElementSettingData

1318 The CIM\_ElementSettingData association associates the top-level instance of the CIM\_VirtualSystemSet-  
1319 tingData class in a “State” virtual system configuration and top-level instances of the CIM\_VirtualSystem-  
1320 SettingData class in other virtual system configurations.

1321 Table 14 lists the requirements for this association.

1322 **Table 14 – Association: CIM\_ElementSettingData**

Element	Requirement	Notes
ManagedElement	Mandatory	<b>Key:</b> Reference to an instance of the CIM_VirtualSystemSettingData class that represents the virtualization-specific properties of the virtual system <b>Cardinality:</b> 0..1 See section 7.3.3 for additional restrictions on the cardinality.
SettingData	Mandatory	<b>Key:</b> Reference to an instance of the CIM_VirtualSystemSettingData class that represents a virtual system configuration <b>Cardinality:</b> * See section 7.3.3 for additional restrictions on the cardinality.
IsDefault	Mandatory	See section 7.3.11.
IsCurrent	Unspecified	
IsNext	Mandatory	See section 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.

1323 **10.6 CIM\_EnabledLogicalElementCapabilities**

1324 The use of the CIM\_EnabledLogicalElementCapabilities class is specialized in the *Computer System Profile*.  
1325

1326 The requirements denoted in Table 15 are in addition to those mandated by the *Computer System Profile*.

1327 **Table 15 – Class: CIM\_EnabledLogicalElementCapabilities**

Element	Requirement	Notes
RequestedStatesSupported[]	Mandatory	See section 7.5.1.1.

1328 **10.7 CIM\_PowerManagementService**

1329 The CIM\_PowerManagementService class is defined by the *Power State Management Profile*, section  
1330 10. The *Virtual System Profile* specifies optional elements in section 7.7 and conditional elements in sec-  
1331 tion 8.1.2.

1332 **10.8 CIM\_ReferencedProfile (Conditional)**

1333 The CIM\_ReferencedProfile association associates the instance of the CIM\_RegisteredProfile class rep-  
1334 resenting an implementation of this profile with instances of the CIM\_RegisteredProfile class representing  
1335 DMTF management profiles that describe logical elements.

1336 This profile refines requirements of the *Profile Registration Profile* by establishing conditions for the sup-  
1337 port of the CIM\_ReferencedProfile association.

1338 Support of the CIM\_ReferencedProfile association is conditional with respect to the presence of an in-  
1339 stance of the CIM\_RegisteredProfile class representing a DMTF management profile that is scoped by  
1340 this profile.

1341 Table 16 contains the requirements for this association.

1342 **Table 16 – Association: CIM\_ReferencedProfile**

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

1343 **10.9 CIM\_RegisteredProfile**1344 The use of the CIM\_RegisteredProfile class is specialized by the *Profile Registration Profile*.1345 The requirements denoted in Table 17 are in addition to those mandated by the *Profile Registration Profile*.  
13461347 **Table 17 – Class: CIM\_RegisteredProfile**

Elements	Requirement	Notes
RegisteredOrganization	Mandatory	Shall be set to 2 (DMTF)
RegisteredName	Mandatory	Shall be set to “ <i>Virtual System Profile</i> ”
RegisteredVersion	Mandatory	Shall be set to the version of this profile: “1.0.0a”.

1348 **10.10 CIM\_SettingsDefineState**1349 The CIM\_SettingsDefineState association associates an instance of the CIM\_ComputerSystem class representing a virtual system and an instance of the CIM\_VirtualSystemSettingData class that represents the virtualization-specific properties of a virtual system and is the top-level instance of the “State” virtual system configuration.  
1350  
1351  
1352

1353 Table 18 contains the requirements for this association.

1354 **Table 18 – Association: CIM\_SettingsDefineState**

Elements	Requirement	Notes
ManagedElement	Mandatory	<b>Key:</b> Reference to an instance of the CIM_ComputerSystem class that represents a virtual system <b>Cardinality:</b> 0..1 See section 7.3.2 for additional restrictions on the cardinality.
SettingData	Mandatory	<b>Key:</b> Reference to an instance of the CIM_VirtualSystemSettingData class that represents the virtualization-specific properties of a virtual system. <b>Cardinality:</b> 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.		

1355 **10.11 CIM\_VirtualSystemSettingData**

1356 The CIM\_VirtualSystemSettingData class models virtualization-specific aspects of a virtual system.

1357 Table 19 contains the requirements for this class.

1358 **Table 19 – Class: CIM\_VirtualSystemSettingData**

Element	Requirement	Notes
InstanceID	Mandatory	<b>Key</b>
Caption	Optional	See section 7.3.6

Element	Requirement	Notes
Description	Optional	See section 7.3.7.
ElementName	Optional	See section 7.3.8.
VirtualSystemIdentifier	Optional	See section 7.3.9.
VirtualSystemType	Optional	See section 7.3.10.

1359 **10.12 CIM\_VirtualSystemSettingDataComponent (Conditional)**

1360 The CIM\_VirtualSystemSettingDataComponent association associates an instance of the CIM\_Virtual-  
 1361 SystemSettingData class representing the virtual aspects of a virtual system and instances of the  
 1362 CIM\_ResourceAllocationSettingData class representing virtual aspects of virtual resources.

1363 Support of the CIM\_VirtualSystemSettingDataComponent association is conditional with respect to the  
 1364 implementation of the CIM\_ResourceAllocationSettingData class specified by a resource-type-specific  
 1365 resource allocation profile or by the *Generic Device Resource Virtualization Profile*.

1366 Table 20 contains the requirements for this association.

1367 **Table 20 – Association: CIM\_VirtualSystemSettingDataComponent**

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

1368



## Annex A (Informative)

### Virtual System Modeling — Background Information

1369  
1370  
1371  
1372

#### 1373 **A.1 Concepts: Model, View, Controller**

1374 This profile – the *Virtual System Profile* - (like any DMTF management profile) specifies only an interface  
1375 or view to an otherwise opaque internal model maintained by an implementation. This profile does not  
1376 specify how a virtual system is modeled within an implementation; this profile specifies only a view of that  
1377 internal model and some control elements. The view enables a client to *observe* the internal model; the  
1378 control elements enable a client to *effect* model *changes* that in turn become visible through the view.

1379 The view is specified in terms of CIM classes and CIM associations; the control elements are specified in  
1380 terms of CIM methods. For that reason the term *CIM model* is frequently used instead of view. This is ac-  
1381 ceptable as long as it is understood that a CIM model in fact just represents an interface or view to the  
1382 internal model maintained by the implementation.

1383 The implementation presents instances of CIM classes and associations on request from clients. These  
1384 instances are fed with data that the implementation obtains from the internal model, using implementa-  
1385 tion-specific means. The implementation executes CIM methods on request from clients. CIM methods  
1386 are realized using implementation-specific control mechanisms such as program or command-line inter-  
1387 faces, for example.

1388 This profile does not specify restrictions on the internal model itself. For example, the implementation is  
1389 free to decide which elements of its internal model it exposes through the view defined by this profile, and  
1390 in most cases the CIM view exposes only a very limited subset of the internal model.

#### 1391 **A.2 Aspect-Oriented Modeling Approach**

1392 One possible approach to model system virtualization would be to specify virtualization-specific derived  
1393 classes for virtual systems and components. For example, to model a virtual system one could model a  
1394 CIM\_VirtualComputerSystem class extending the CIM\_ComputerSystem class with virtualization-specific  
1395 properties and methods.

1396 This inheritance-based modeling approach was not applied for various reasons:

- 1397 • A virtual system should appear to a virtualization-unaware client exactly like a non-virtual com-  
1398 puter system.
- 1399 • The single-inheritance modeling approach is not suited for various management domains being  
1400 modeled on top of the same set of base classes. For example, if the CIM\_VirtualComputerSys-  
1401 tem and CIM\_PartitionedComputerSystem classes were both derived from the CIM\_Computer-  
1402 System class, then a particular instance could represent either a virtual system or a partitioned  
1403 system, but not both.
- 1404 • Many virtualization platforms support the concepts of virtual system definition and virtual system  
1405 instance. The definition is a formal description of the virtual system; the instance is the internal  
1406 representation of the virtual system in the “Active” state. Ideally, both definition and instance are  
1407 described using the same set of CIM classes.

1408 Instead, a large part of the model specified by this profile is based on classes derived from CIM\_Setting-  
1409 Data:

- 1410 • Settings allow virtualization-specific information to be modeled separately from the target class.
- 1411 • Settings are ideally suited to model descriptive data, such as virtual resource definitions.

- 1412 • Settings are easily aggregated into larger configurations, such as virtual system configuration
- 1413 covering the virtual system itself and all of its resources.
- 1414 • Settings allow extending the property set of existing classes in an aspect-oriented way. Various
- 1415 aspects, such as “virtualization” and “partitioning,” can exist in parallel for the same managed
- 1416 element.

1417 **A.3 Presence of Model Information**

1418 The *Management Profile Specification Usage Guide* requires an autonomous profile to specify a central

1419 class and a scoping class. The *Computer System Profile* specifies the CIM\_ComputerSystem class for

1420 both the central and scoping class. This profile - the *Virtual System Profile* - specializes the *Computer*

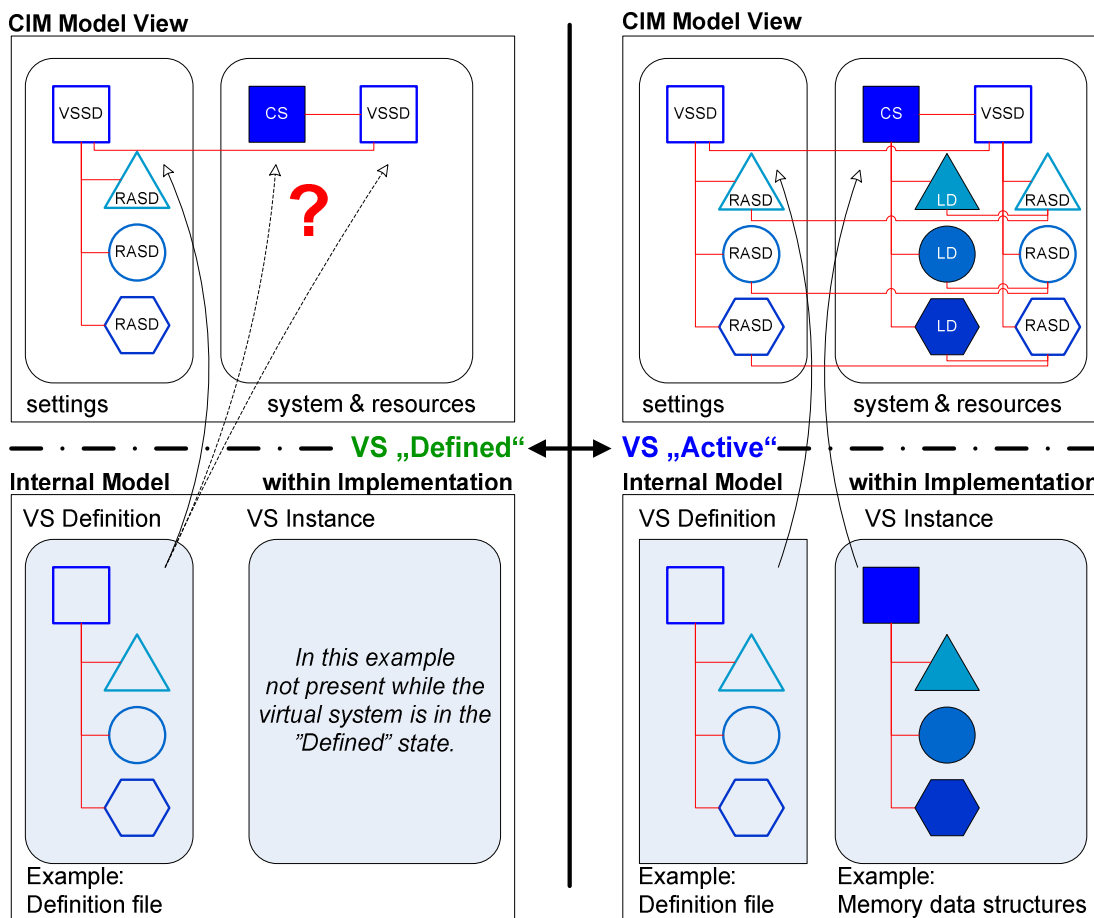
1421 *System Profile*, and thus is required to use the CIM\_ComputerSystem class (or a derived class) for central

1422 and scoping class as well.

1423 The *Management Profile Specification Usage Guide* further requires that an instance of that class must

1424 be present at all times. Figure 8 illustrates that this requirement in some cases causes a potential model

1425 representation problem.



1426

1427

**Figure 8 – State-Dependent Presence of Model Elements**

1428 The left side of Figure 8 shows a virtual system in the “Defined” state. In this example the virtualization

1429 platform distinguishes between virtual system definition and virtual system instance; the virtual system

1430 instance does not exist while the virtual system is in the “Defined” state. Nevertheless, the implementation

1431 is required to represent a (virtual) computer system through an instance of the CIM\_ComputerSystem

1432 class during its complete lifecycle, including periods when the virtual system is only defined but not active

1433 and instantiated at the virtualization platform. This causes a model representation problem: Many proper-  
1434 ties of the CIM\_ComputerSystem class (with instances labeled “CS” in Figure 8) model information about  
1435 a stateful virtual system instance, but not about a stateless virtual system definition.

1436 For that reason the property set of the CIM\_ComputerSystem class can only be completely presented by  
1437 the implementation while the virtual system is instantiated. While the virtual system is in the “Defined”  
1438 state, respective properties of the instance of the CIM\_ComputerSystem class representing the virtual  
1439 system are one of the following:

- 1440 • undefined and have a value of NULL
- 1441 • fed from the virtual system definition instead of from the (in this state, non-existent) virtual sys-  
1442 tem instance (This is indicated by the dashed curved arrows in Figure 8.)

1443 The right side of Figure 8 shows the same virtual system in the “Active” state. Because in this state the  
1444 virtual system instance exists in addition to the virtual system definition, data is directly fed from the virtual  
1445 system instance into the system and resources part of the CIM model.

1446 Note that the situation is different for virtual resources. The *Resource Allocation Profile* does not require  
1447 an instance of the CIM\_LogicalDevice class to be present at all times; consequently, instances of the  
1448 CIM\_LogicalDevice class appear only as long as their scoping virtual system is instantiated.

#### 1449 **A.4 Model Extension through Settings**

1450 The right side of Figure 8 illustrates another modeling approach applied by this profile: The extension of  
1451 the virtual system representation with virtualization-specific properties through settings. The upper right  
1452 part of Figure 8 shows how the virtual system itself is represented by an instance of the CIM\_Computer-  
1453 System class (labeled “CS”) and virtual resources are represented by instances of the CIM\_LogicalDevice  
1454 class (labeled “LD”). On the right side these instances are associated with setting classes that extend the  
1455 property set of computer system and resource representations with virtualization-specific information (la-  
1456 beled VSSD for the virtual system extension and RASD for the set of virtual resource extensions). This  
1457 profile specifies an approach where these extensions are modeled by the same set of classes that are  
1458 used to represent a virtual system definition.

## Annex B (Informative)

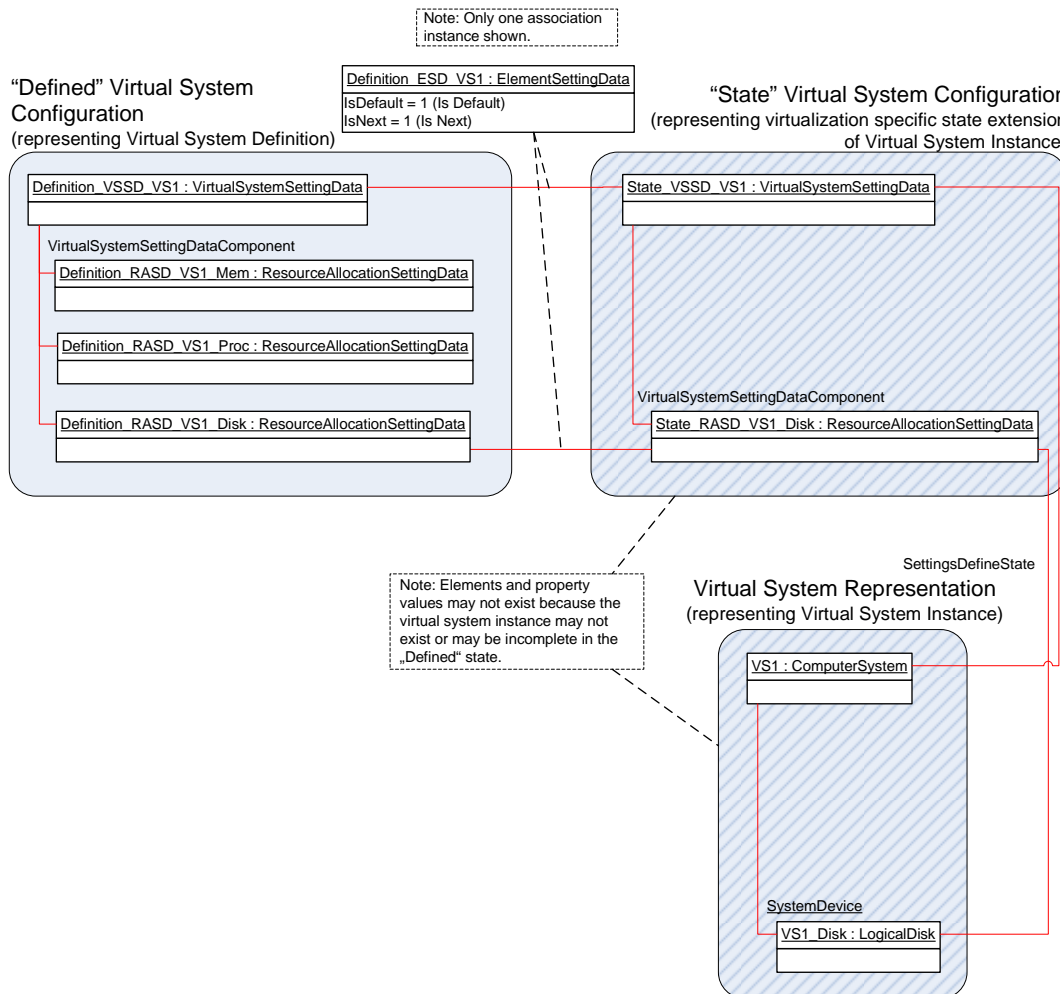
### Implementation Details

1459  
1460  
1461  
1462

#### B.1 Dual-Configuration: Implementation Approach

1464 Figure 9 shows an example of a virtual system in the “Defined” state. There are two virtual system con-  
1465 figurations: The virtual system configuration on the left is the “Defined” virtual system configuration: the  
1466 virtual system configuration on the right is the “State” virtual system configuration.

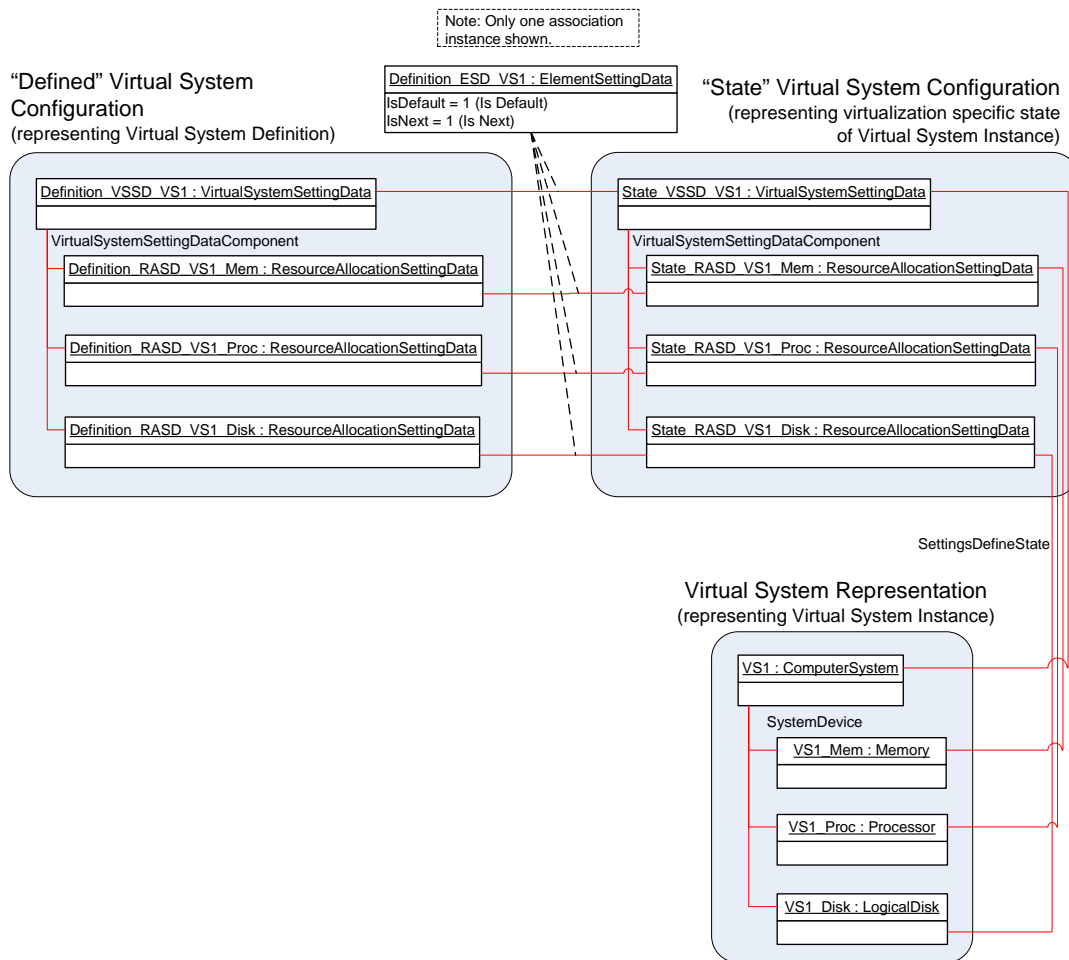
1467 Note that in this example virtual resource VS1\_Disk has a persistently allocated resource that remains  
1468 allocated regardless of the virtual system state. Consequently, an instance of the CIM\_LogicalDisk class  
1469 (tagged VS1\_Disk) represents the disk in the “Defined” state already, and virtualization-specific properties  
1470 are represented by an instance of the CIM\_ResourceAllocationSettingData class (tagged State\_-  
1471 RASD\_VS1\_Disk) in the “State” virtual system configuration that is associated through the CIM\_Set-  
1472 tingsDefineState association.



1473  
1474

**Figure 9 – Sample Virtual System in "Defined" State (Dual-Configuration Approach)**

1475 The same system is shown in Figure 10 in a state other than the “Defined” state.



1476

1477 **Figure 10 – Sample Virtual System in a State Other Than "Defined" (Dual-Configuration Approach)**

1478 Resources for virtual resources were allocated, and virtual resources are represented by instances of the  
 1479 CIM\_LogicalDevice class. Virtualization-specific properties are represented as instances of the CIM\_Re-  
 1480 sourceAllocationSettingData class in the “State” virtual system configuration that are associated through  
 1481 instances of the CIM\_SettingsDefineState association.

1482 NOTE 1 The *Virtual System Profile* specifies a CIM view of virtual systems. The *Virtual System Profile* does not  
 1483 specify restrictions on the internal model maintained by the implementation to ensure that all resources are allocated  
 1484 during system activation; instead, the implementation is free to decide whether activation is successful or fails if some  
 1485 virtual resources are not able to be allocated.

1486 NOTE 2 If the *Resource Allocation Profile* is implemented for a particular resource type, it may require that, as vir-  
 1487 tual resources are allocated or de-allocated, respective instances of the CIM\_LogicalDevice class are created or de-  
 1488 stroyed in the virtual system representation, and that these instances are connected to their counterpart in the “State”  
 1489 virtual system configuration through respective instances of the CIM\_SettingsDefineState association, and that the  
 1490 instances in the “State” virtual system configuration are connected to their counterpart in the “Defined” virtual system  
 1491 configuration through respective instances of the CIM\_ElementSettingData association with the IsDefault property set  
 1492 to 1 (Is Default).

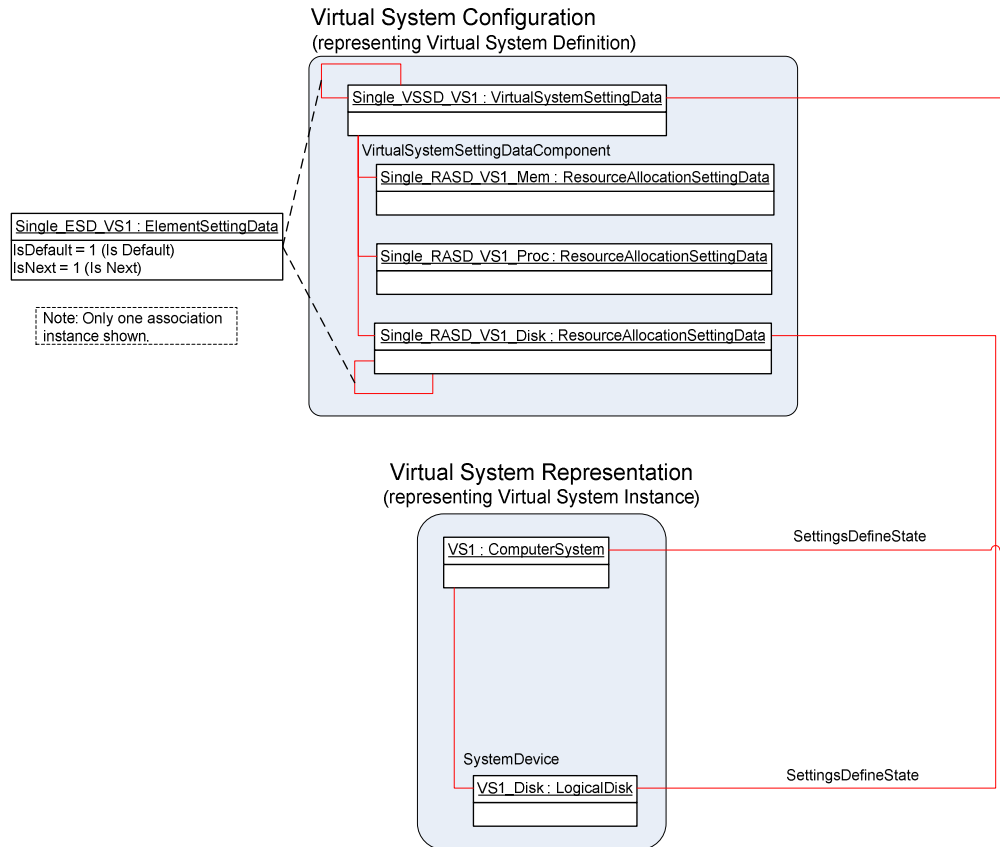
## 1493 B.2 Single-Configuration Implementation Approach

1494 Figure 11 shows an example in which a virtual system is in the “Defined” state. Only one set of instances  
 1495 of the CIM\_VirtualSystemSettingData class and the CIM\_ResourceAllocationSettingData class compose

## Virtual System Profile

1496 a single virtual system configuration instance that acts as the “Defined” and as the “State” virtual system  
 1497 configuration. The single configuration instance is associated to the instance of the CIM\_ComputerSys-  
 1498 tem class representing the virtual system through an instance of the CIM\_SettingsDefineState associa-  
 1499 tion.

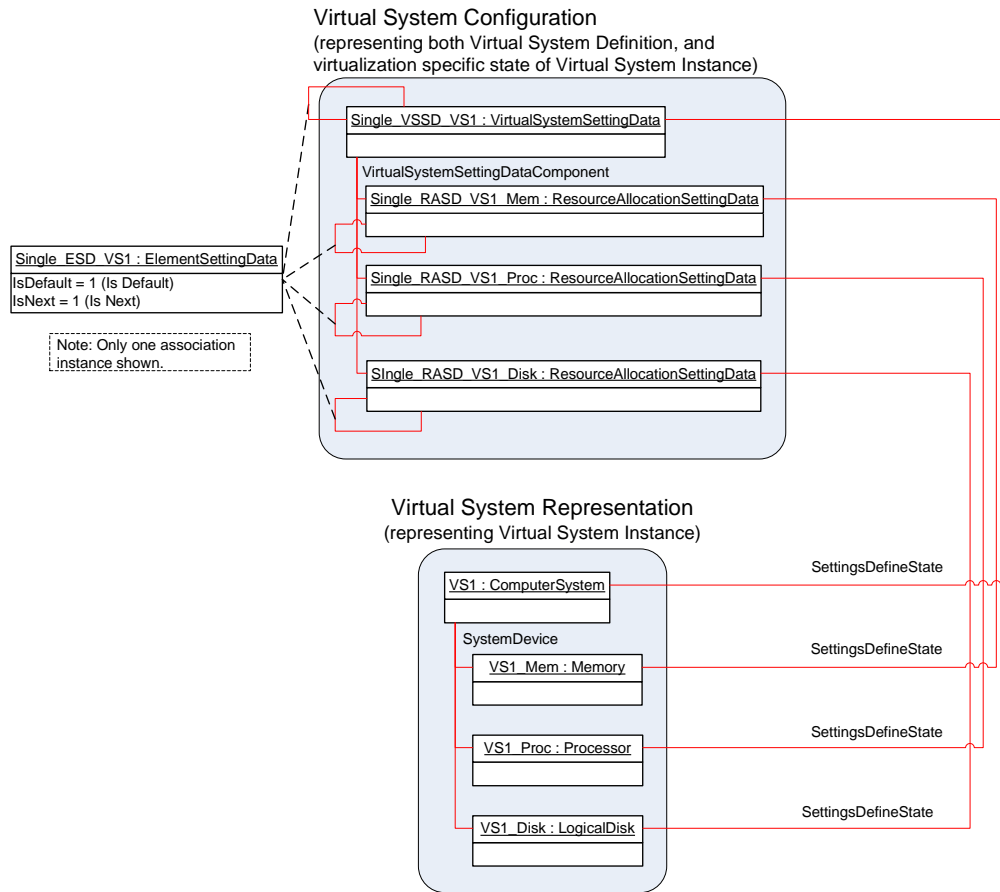
1500 Note that in this example virtual resource VS1\_Disk has a persistently allocated resource that remains  
 1501 allocated regardless of the virtual system state. Consequently, an instance of the CIM\_LogicalDisk class  
 1502 tagged VS1\_Disk represents the disk in the “Defined” state already, and virtualization-specific properties  
 1503 are represented by an instance of the CIM\_ResourceAllocationSettingData class tagged State\_RASD-  
 1504 \_VS1\_Disk in the “State” virtual system configuration that is associated through the CIM\_SettingsDefine-  
 1505 State association.



1506

1507 **Figure 11 – Sample Virtual System in a "Defined" State (Single-Configuration Approach)**

1508 In Figure 12 the same virtual system is shown in a state other than “Defined”.



1509

1510 **Figure 12 – Sample Virtual System in a State Other Than "Defined" (Single-Configuration**  
 1511 **Approach)**

1512 Resources for virtual resources were allocated. Virtual resources are represented by instances of the  
 1513 CIM\_LogicalDevice class, with virtualization-specific properties represented as instances of the CIM\_Re-  
 1514 sourceAllocationSettingData class in the “State” virtual system configuration and associated through in-  
 1515 stances of the CIM\_SettingsDefineState association.

1516 **NOTE** If the *Resource Allocation Profile* is implemented for a particular resource type, it may require that, as virtual  
 1517 resources are allocated or de-allocated and respective instances of the CIM\_LogicalDevice class are created or de-  
 1518 stroyed in the virtual system representation, these instances are connected to their counterpart in the “State” virtual  
 1519 system configuration through respective instances of the CIM\_SettingsDefineState association. The *Resource Alloca-*  
 1520 *tion Profile* may also require that the instances in the “State” virtual system configuration are connected to their coun-  
 1521 terpart in the “Defined” virtual system configuration through respective instances of the CIM\_ElementSettingData  
 1522 association with the `IsDefault` property set to 1 (Is Default); in the single-configuration implementation approach,  
 1523 these association instances connect elements of the single virtual system configuration to themselves.

1524

**Annex C  
(Informative)**

**Change Log**

1525  
1526  
1527  
1528

1529

Version	Date	Description
1.0.0a	05/07/2007	Initial creation.

1530



## Annex D (Informative)

### Acknowledgements

1531  
1532  
1533  
1534

1535 The authors wish to acknowledge the following people.

- 1536 • Editor:
  - 1537 – Michael Johanssen – IBM
- 1538 • Participants from the DMTF System Virtualization, Partitioning and Clustering Working Group:
  - 1539 – Gareth Bestor – IBM
  - 1540 – Chris Brown – HP
  - 1541 – Mike Dutch - Symantec
  - 1542 – Jim Fehlig – Novell
  - 1543 – Kevin Fox – Sun Microsystems, Inc.
  - 1544 – Ron Goering – IBM
  - 1545 – Steve Hand - Symantec
  - 1546 – Daniel Hiltgen – EMC / VMware
  - 1547 – Michael Johanssen – IBM
  - 1548 – Larry Lamers – EMC / VMware
  - 1549 – Andreas Maier - IBM
  - 1550 – Aaron Merkin – IBM
  - 1551 – John Parchem – Microsoft
  - 1552 – Joanne Saathof - CPubs
  - 1553 – Nihar Shah – Microsoft
  - 1554 – David Simpson – IBM
  - 1555 – Carl Waldspurger – EMC / VMware