Document Number: DSP1057

Date: 2010-04-22

Version: 1.0.0

Virtual System Profile

Document Type: Specification

Document Status: DMTF Standard

Document Language: E

CONTENTS

Figures

Tables

Foreword

This profile — the *Virtual System Profile* (DSP1057) — was prepared by the System Virtualization, Partitioning and Clustering Working Group of the DMTF.

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

Acknowledgments

The authors wish to acknowledge the following people.

- Editor:
  - Michael Johanssen – IBM
- Participants from the DMTF System Virtualization, Partitioning and Clustering Working Group:
  - Gareth Bestor – IBM
  - Chris Brown – HP
  - Mike Dutch - Symantec
  - Jim Fehlig – Novell
  - Kevin Fox – Sun Microsystems, Inc.
  - Ron Goering – IBM
  - Steve Hand - Symantec
  - Daniel Hiltgen – EMC / VMware
  - Michael Johanssen – IBM
  - Larry Lamers – EMC / VMware
  - Andreas Maier - IBM
  - Aaron Merkin – IBM
  - John Parchem – Microsoft
  - Nihar Shah – Microsoft
  - David Simpson – IBM
  - Carl Waldspurger – EMC / VMware

Introduction

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

Virtual System Profile

1 Scope

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

2 Normative references

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

DMTF DSP0004, *CIM Infrastructure Specification* 2.5
http://www.dmtf.org/standards/published_documents/DSP0004_2.5.pdf

DMTF DSP0200, *CIM Operations over HTTP* 1.3
http://www.dmtf.org/standards/published_documents/DSP0200_1.3.pdf

DMTF DSP0205, *WBEM Discovery Using the Service Location Protocol (SLP) 1.0*
http://www.dmtf.org/standards/published_documents/DSP0205_1.0.pdf

DMTF DSP1001, *Management Profile Specification Usage Guide* 1.0
http://www.dmtf.org/standards/published_documents/DSP1001_1.0.pdf

DMTF DSP1012, *Boot Control Profile 1.0*
http://www.dmtf.org/standards/published_documents/DSP1012_1.0.pdf

DMTF DSP1022, *CPU Profile 1.0*
http://www.dmtf.org/standards/published_documents/DSP1022_1.0.pdf

DMTF DSP1026, *System Memory Profile 1.0*
http://www.dmtf.org/standards/published_documents/DSP1026_1.0.pdf

DMTF DSP1027, *Power State Management Profile 1.0*
http://www.dmtf.org/standards/published_documents/DSP1027_1.0.pdf

DMTF DSP1033, *Profile Registration Profile* 1.0
http://www.dmtf.org/standards/published_documents/DSP1033_1.0.pdf

DMTF DSP1041, *Resource Allocation Profile 1.1*
http://www.dmtf.org/standards/published_documents/DSP1041_1.1.pdf

DMTF DSP1042, *System Virtualization Profile 1.0*
http://www.dmtf.org/standards/published_documents/DSP1042_1.0.pdf

DMTF DSP1043, *Allocation Capabilities Profile* 1.0
http://www.dmtf.org/standards/published_documents/DSP1043_1.0.pdf

DMTF DSP1044, *Processor Device Resource Virtualization Profile 1.0*
http://www.dmtf.org/standards/published_documents/DSP1044_1.0.pdf

DMTF DSP1045, *Memory Resource Virtualization Profile 1.0*
http://www.dmtf.org/standards/published_documents/DSP1045_1.0.pdf

DMTF DSP1047, *Storage Resource Virtualization Profile 1.0*
http://www.dmtf.org/standards/published_documents/DSP1047_1.0.pdf

DMTF DSP1052, *Computer System Profile 1.0*
http://www.dmtf.org/standards/published_documents/DSP1052_1.0.pdf

DMTF DSP1059, *Generic Device Resource Virtualization Profile 1.0*
http://www.dmtf.org/standards/published_documents/DSP1059_1.0.pdf

ISO/IEC Directives, Part2:2004, *Rules for the structure and drafting of International Standards*, http://isotc.iso.org/livelink/livelink.exe?func=ll&objId=4230456&objAction=browse&sort=subtype

3 Terms and definitions

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

3.1

can

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

3.2

cannot

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

3.3

conditional

indicates requirements strictly to be followed in order to conform to the document and from which no deviation is permitted when the specified conditions are met

3.4

mandatory

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

3.5

may

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

3.6

need not

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

3.7

optional

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

3.8

referencing profile

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

3.9

shall

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

3.10

shall not

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

3.11

should

indicates that among several possibilities, one is recommended as particularly suitable, without mentioning or excluding others, or that a certain course of action is preferred but not necessarily required

3.12

should not

indicates that a certain possibility or course of action is deprecated but not prohibited

3.13

unspecified

indicates that this profile does not define any constraints for the referenced CIM element

3.14

implementation

a set of CIM providers that realize the classes specified by this profile

3.15

client

an application that exploits facilities specified by this profile

3.16

virtualization platform

virtualizing infrastructure provided by a host system enabling the deployment of virtual systems

3.17

WBEM service

an abstract class for WBEM services such as the ObjectManager, providers, CIM repositories, or any other WBEM Server component. It is a type of service that provides associated capabilities and details about the component.

4 Symbols and abbreviated terms

4.1

CIM

Common Information Model

4.2

RASD

CIM_ResourceAllocationSettingData

4.3

SLP

service location protocol

4.4

VS

virtual system

4.5

VSSD

CIM_VirtualSystemSettingData

5 Synopsis

Profile Name: Virtual System

Version: 1.0.0

Organization: DMTF

CIM Schema Version: 2.22

Central Class: CIM_ComputerSystem

Scoping Class: CIM_ComputerSystem

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

The instance of the CIM_ComputerSystem class representing a virtual system shall be the central instance and the scoping instance of this profile.

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

Table 1 – Related profiles

Profile Name	Organization	Version	Relationship	Description
Profile Registration	DMTF	1.0	Mandatory	The profile that specifies registered profiles.
Computer System	DMTF	1.0	Specialization	The abstract autonomous profile that specifies the minimum object model needed to define a basic computer system.
Power State Management	DMTF	1.0	Optional	The component profile that specifies an object model needed to describe and manage the power state of server systems.
Boot Control	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.

6 Description

This profile (DSP1057, Virtual System Profile) specializes DSP1052 (*Computer System Profile)* that defines the minimum top-level object model needed to define a basic computing platform. The primary design objective applied by this profile is that a virtual system and its components appear to a client in the same way as a non-virtual system. Typical management tasks such as enumerating, analyzing, controlling, or configuring a system should be enabled without requiring the client to understand specific aspects of virtual systems.

6.1 Profile relationships

This profile (DSP1057) is complementary to DSP1042 (*System Virtualization Profile)*:

- This profile focuses on virtualization aspects that relate to virtual systems and their virtual resources, such as modeling the *structure* of virtual systems and their resources. The profile introduces the concept of virtual system configurations allowing the inspection of virtual system configuration and state information.
- DSP1042 focuses on virtualization aspects that relate to host systems and their resources, such as modeling the *relationships* between host resources and virtual resources. Further it addresses virtualization-specific tasks such as the creation or modification of virtual systems and their configurations.

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

- This profile (DSP1057)

  This profile enables the inspection of and basic operations on virtual systems.
- DSP1042

  DSP1042 enables the inspection of host systems, their capabilities, and their services for creation and manipulation of virtual systems.
- Resource-type-specific profiles

  Resource-type-specific profiles enable the inspection and operation of resources for one particular resource type. They apply to both virtual and host resources; they do not cover virtualization-specific aspects of resources. A client may exploit resource-type-specific management profiles for the inspection and manipulation of virtual and host resources in a similar manner.
- Resource allocation profiles

  Resource allocation profiles enable the inspection of existing resource allocations and of host and other resources available for allocation. Resource allocation profiles are based on DSP1041 and DSP1043, and they are scoped by DSP1042. A client may exploit resource allocation profiles to inspect all of the following:

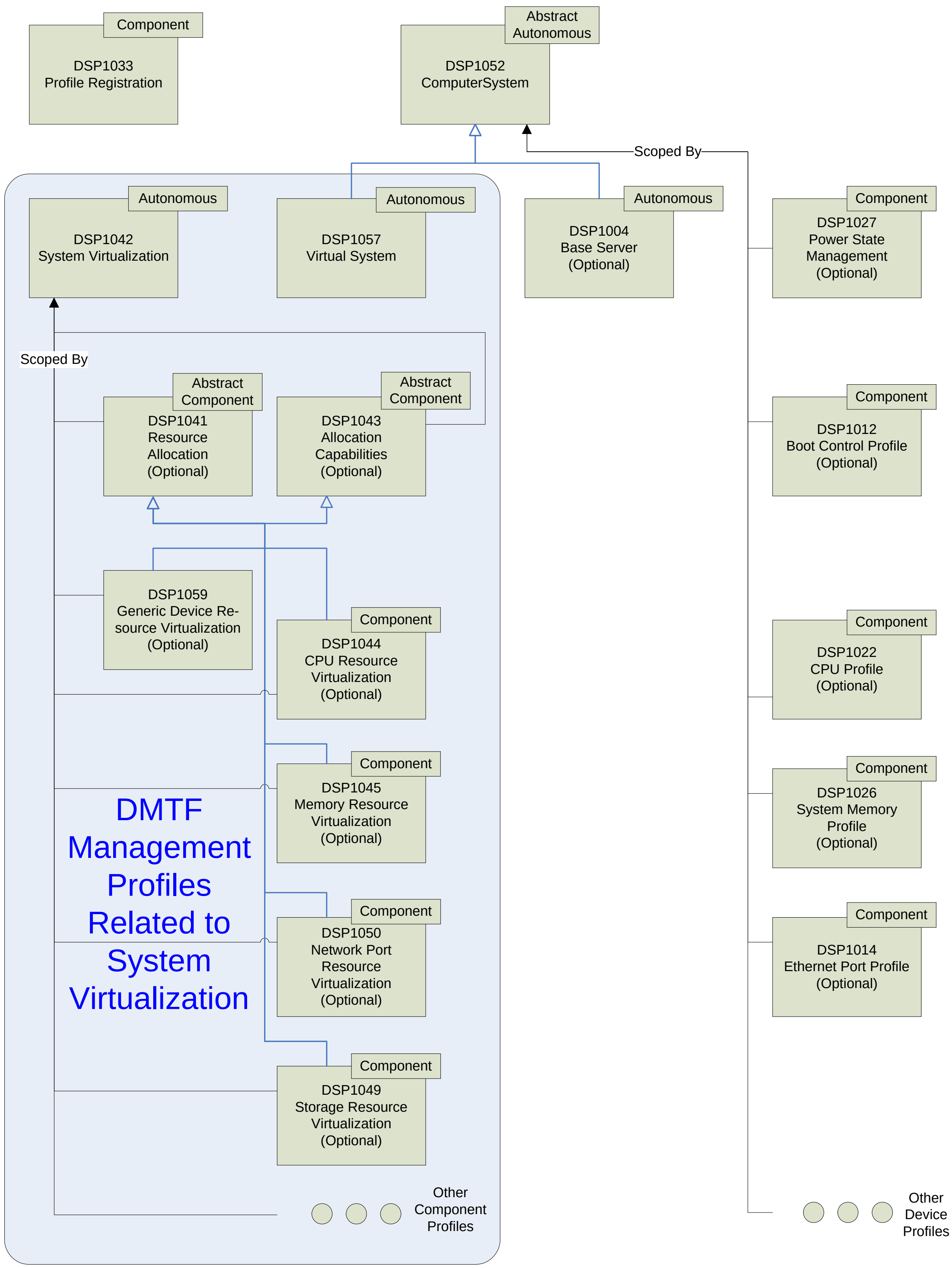
  – the allocation of resources

  – the allocation dependencies that virtual resources have on host resources and resource pools

  – the capabilities describing possible values for resource allocations

  – the capabilities describing the mutability of resource allocations

  DSP1059 (*Generic Device Resource Virtualization Profile*) is a resource-type-independent resource allocation profile that specifies the management of the allocation of basic virtual resources. For some resource types, specific resource allocation profiles are defined that address resource-type-specific allocation aspects and capabilities.

358 

359 **Figure 1 – Profiles related to system virtualization**

6.2 Virtual system class schema

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

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

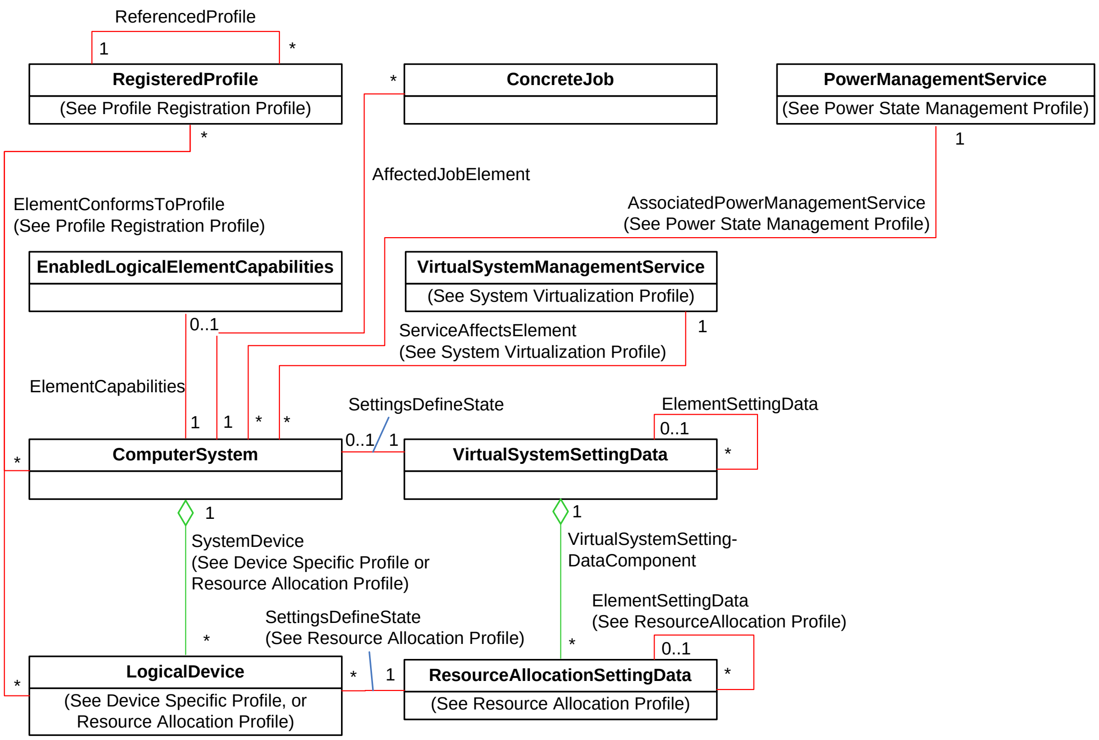


Figure 2 – Virtual System Profile: Class diagram

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

- the CIM_ComputerSystem class to represent virtual systems
- the CIM_RegisteredProfile class and the CIM_ElementConformsToProfile association to model conformance with this profile
- the CIM_ReferencedProfile association to model dependencies between this profile and resource-type-specific resource allocation profiles
- the CIM_EnabledLogicalElementCapabilities class and the CIM_ElementCapabilities association to model capabilities of a virtual system such as characteristics of certain properties or the set of potential state transitions
- the CIM_VirtualSystemSettingData class to model virtualization-specific aspects of a virtual system
- the CIM_VirtualSystemSettingDataComponent association to model the aggregation of instances of the CIM_ResourceAllocationSettingData class to one instance of the CIM_VirtualSystemSettingData class, forming a virtual system configuration

- the CIM_SettingsDefineState association to model the relationship between an instance of the CIM_ComputerSystem class representing a virtual system and an instance of the CIM_VirtualSystemSettingData class representing virtualization specific aspects of that virtual system
- the CIM_ElementSettingData association to model the relationship between an element and configuration data applicable to the element
- the CIM_ConcreteJob class and the CIM_AffectedJobElement association to model a mechanism that allows tracking of asynchronous tasks resulting from operations such as the optional RequestStateChange( ) method applied to instances of the CIM_ComputerSystem class

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

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

6.3 Virtual system concepts: Definition, instance, representation, and configuration

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

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

The term *virtual system representation* refers to the set of CIM class instances that represent the current state of a virtual system instance. A virtual system representation consists of one top-level instance of the CIM_ComputerSystem class and a set of aggregated instances of the CIM_LogicalDevice class. The state of the system and logical devices is thus represented by the set of property values in these instances. Virtualization specific state is not yet represented; for that purpose the next paragraph introduces a virtualization specific state extension to the virtual system representation. The presence of instances of the CIM_LogicalDevice class within the virtual system representation is controlled by specializations of DSP1041. The specializations describe how instances of the CIM_LogicalDevice class are added or removed from the virtual system representation as virtual resources are allocated or deallocated.

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

- Virtual system configurations are used for the representation of configuration information, in particular for the representation of virtual system definitions.
- Virtual system configurations are used for the representation of virtualization specific "State" that extends the virtual system representation. A single "state" virtual system configuration is associated to a virtual system. Elements of the "state" virtual system configuration extend corresponding elements of the virtual system representation with virtualization-specific properties. A variety of virtual system configurations may be associated with the "state" configuration via the CIM_ElementSettingData association. An example is the representation of the virtual system definition by a separate "Defined" virtual system configuration.

Virtualization platforms may support modifications on virtual system definitions or virtual system instances through various means, for example through direct configuration file editing, through a command-line interface, through a program interface, or through a CIM-based interface as modeled in DSP1042. Regardless of the mechanism used to effect a modification on a virtual system definition or a virtual system instance that modification becomes visible to clients through the CIM model view defined in this profile (DSP1057), as expressed by the respective virtual system configuration or the virtual system representation.

6.4 Virtual system states and transitions

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

6.4.1 Virtual system states

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

6.4.1.1 Semantics of the "defined" state

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

6.4.1.2 Semantics of the "active" state

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

6.4.1.3 Semantics of the "paused" state

In the "paused" state the virtual system and its virtual resources remain instantiated and resources remain allocated as in the "active" state, but the virtual system and its virtual resources are not enabled to perform tasks.

6.4.1.4 Semantics of the "suspended" state

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

6.4.1.5 Vendor-defined states

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

6.4.1.6 Semantics of the "unknown" state

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

6.4.2 Virtual system state transitions

This subclause describes various virtual system state transitions and their semantics. Normative requirements for the observation of virtual system state transitions are specified in 7.1.2.

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

6.4.2.1 The "define" state transition

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

6.4.2.2 Semantics of the "activate" state transition

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

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

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

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

6.4.2.3 Semantics of the "deactivate" state transition

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

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

6.4.2.4 Semantics of the "pause" state transition

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

6.4.2.5 Semantics of the "suspend" state transition

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

6.4.2.6 Semantics of the "shut down" state transition

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

6.4.2.7 Semantics of the "reboot" state transition

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

6.4.2.8 Semantics of the "reset" state transition

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

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

6.4.3 Summary of virtual system states and virtual system state transitions

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

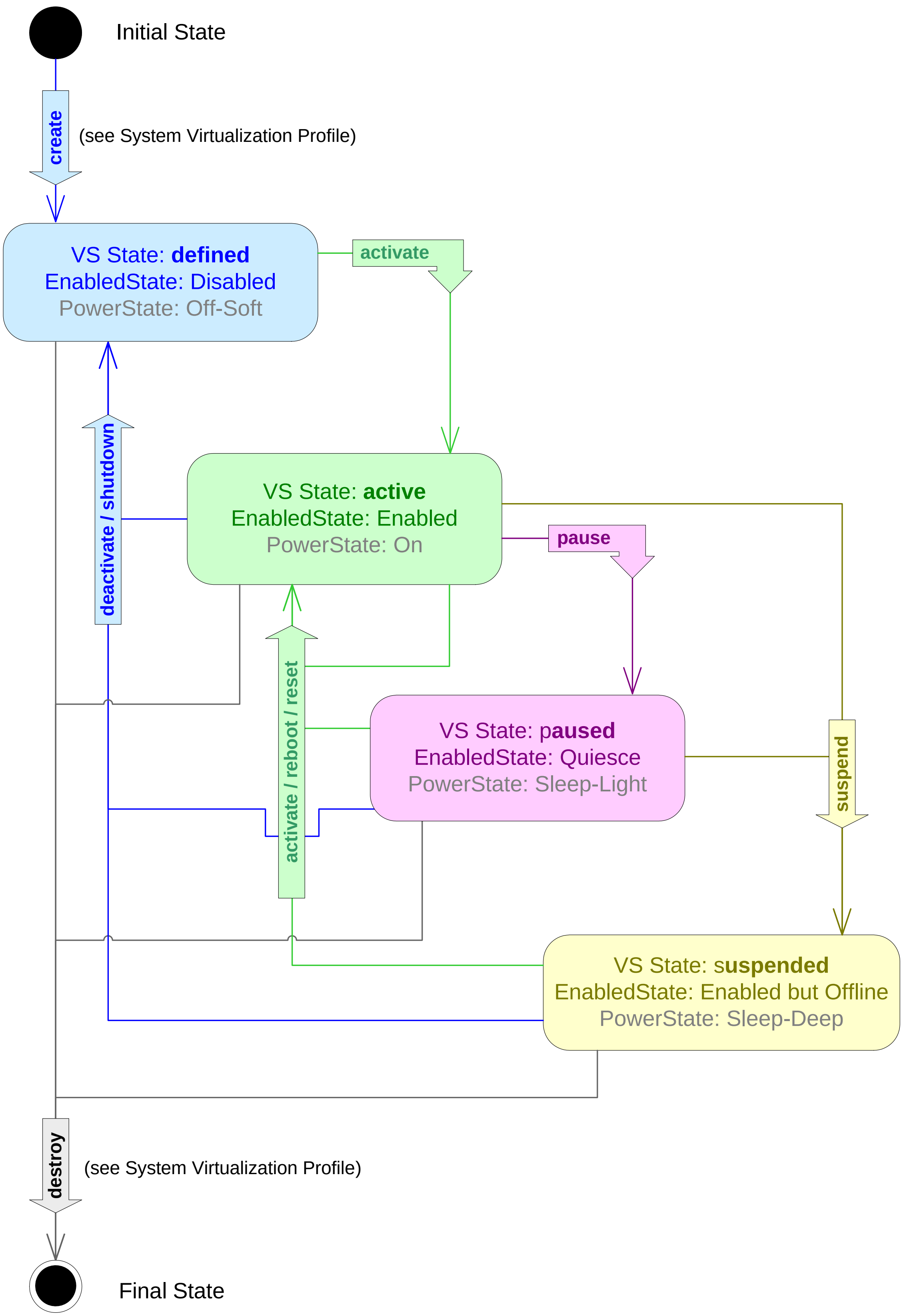


Figure 3 – Virtual system states

7 Implementation

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

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

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

7.1 Virtual system

The CIM_ComputerSystem class shall be used to represent virtual systems. One instance of the CIM_ComputerSystem class shall exist for each virtual system that is conformant to this profile, regardless of its state.

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

7.1.1 CIM_ComputerSystem.EnabledState property

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

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

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

Table 2 – Observation of virtual system states

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

Observation of virtual system state	Requirement	CIM_ComputerSystem EnabledState Property Value	CIM_AssociatedPowerManagementService.PowerState Property Value (Optional)
"**suspended**" (Optional) (See 6.4.1.4)	Optional	**6 (Enabled but Offline)**	**4 (Sleep – Deep)** 7 (Hibernate (Off – Soft))
Vendor Defined (Optional) (See 6.4.1.5)	Optional	**1 (Other)**	**1 (Other)** or (0x7FFF-0xFFFF)
"**unknown**" (Optional) (See 6.4.1.6)	Optional	**0 (Unknown)**	n/a
Unspecified (Values shall not be used by conformant implementations.)	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.			

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

NOTE This profile clearly distinguishes between the observation of virtual system state (as defined in this subclause) and client state management (as defined in 7.6). In particular with respect to the observation of virtual system state no mechanism is specified for determining a supported subset of virtual system states; instead any virtual system state as defined by Table 2 is possible. Opposed to that the set of state transitions that may be effected through client state management is modeled in 7.6 through the CIM_EnabledLogicalElementCapabilities class.

7.1.2 CIM_ComputerSystem.RequestedState property

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

The following provisions apply:

- If the observation of virtual system state transitions is not implemented, the RequestedState property shall be set to a value of 12 (Not Applicable).
- If the observation of one or more virtual system state transitions is implemented, the value of the RequestedState property shall be used to facilitate the observation of virtual system state transitions. The following provisions apply:
  – The RequestedState property shall not have a value of 12 (Not Applicable).
  – The RequestedState property shall have a value designating the most recently requested state transition according to Table 3. For example, if a virtual system is performing an "Activate" state transition, then the RequestedState property shall have a value of 2 (Enabled).
  – If a state transition completes successfully, the value of the EnabledState property shall reflect the "To" virtual system state as defined by Table 3, using values as defined by Table 2. For example, if a virtual system has successfully performed an "activate" state transition, then it shall be in the "active" virtual system state and show a value of 2 (Enabled) for the EnabledState property. The RequestedState property shall maintain the value designating the most recently requested state transition according to Table 3.

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

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

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

Table 3 provides the normative mapping of virtual system state transitions to values of the RequestedState property and the RequestedState parameter.

Table 3 – Observation of virtual system state transitions

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

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

7.2 Virtual resource

Resources in system representations are specified by resource-type-specific profiles such as DSP1052 or DSP1026. These resource-type-specific profiles may be implemented for one or more types of virtual resources, omitting optional elements that model physical aspects.

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

7.3 Virtual system configuration

7.3.1 Structure

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

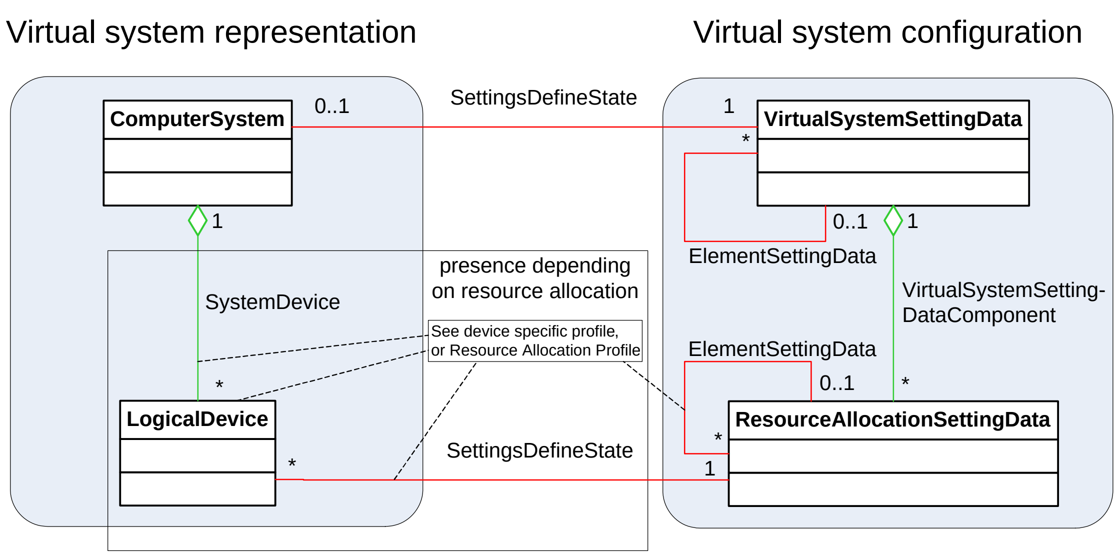


Figure 4 – Virtual system representation and virtual system configuration

7.3.2 The "state" virtual system configuration

There shall be exactly one "state" virtual system configuration representing the virtualization specific state of the virtual system. Elements of the "state" virtual system configuration add virtualization-specific properties to related elements in the virtual system representation. Elements of the "state" virtual system configuration shall have the same lifecycle as their counterparts in the virtual system representation.

The top-level instance of the CIM_VirtualSystemSettingData class in the "state" virtual system configuration shall be associated to the instance of the CIM_ComputerSystem class that represents the virtual system through an instance of the CIM_SettingsDefineState association.

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

NOTE 2 If DSP1041 is implemented for a particular resource type, it may require additional instances of the CIM_SettingsDefineState association connecting instances of the CIM_ResourceAllocationSettingData class in the

"State" virtual system configuration to related instances of the CIM_LogicalDevice class in the virtual system representation.

7.3.3 The "defined" virtual system configuration

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

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

NOTE An implementation may coincide the "defined" virtual system configuration and the "state" virtual system configuration; see 7.3.4 .

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

7.3.4 Implementation approaches for "state" and "defined" virtual system configuration

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

7.3.4.1 Dual-configuration implementation approach

This approach is applicable for implementations that support separate configurations for the representation of the virtual system definition and the virtual system instance. This approach allows the modeling of divergent modifications on definition and instance.

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

For the top-level instance of the CIM_VirtualSystemSettingData class in the "state" virtual system configuration the following provisions apply:

- It shall be associated to the instance of the CIM_ComputerSystem class in the virtual system representation through an instance of the CIM_SettingsDefineState association
- It shall be associated to its counterpart in the "defined" virtual system configuration through an instance of the CIM_ElementSettingData association where
  - the value of the IsDefault property shall be set to according to 7.3.11
  - the value of the IsNext property shall be set to according to 7.3.12
- It shall be associated to any instance of the CIM_ResourceAllocationSettingData class that is part of the "state" virtual system configuration via an instance of the CIM_VirtualSystemSettingDataComponent association

DSP1041 or profiles based on DSP1041 may require compliance to similar conditions with respect to instances of the CIM_ResourceAllocationSettingData class and the CIM_LogicalDevice class. If resources are allocated or de-allocated, respective instances of the CIM_ResourceAllocationSettingData class shall be added to or removed from the "State" virtual system configuration along with the associations referring to them.

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

7.3.4.2 Single-configuration implementation approach

This approach is applicable for implementations that do not support separate configurations for the representation of the virtual system definition and virtual system instance.

With this approach, instances of the CIM_VirtualSystemSettingData class and the CIM_ResourceAllocationSettingData class are shared for both the "defined" virtual system configuration and the "state" virtual system configuration.

For the top-level instance of the CIM_VirtualSystemSettingData class in the single virtual system configuration the following provisions apply:

- It shall be associated to the instance of the CIM_ComputerSystem class in the virtual system representation through an instance of the CIM_SettingsDefineState association
- It shall be associated to itself through an instance of the CIM_ElementSettingData association where
  - the value of the IsDefault property shall be set to according to 7.3.11
  - the value of the IsNext property shall be set to according to 7.3.12
- It shall be associated to any instance of the CIM_ResourceAllocationSettingData class that is part of the single virtual system configuration via an instance of the CIM_VirtualSystemSettingDataComponent association

DSP1041 or profiles based on DSP1041 may require compliance to similar conditions with respect to instances of the CIM_ResourceAllocationSettingData class and the CIM_LogicalDevice class, such that as resources are allocated or de-allocated, respective instances of the CIM_SettingsDefineState association and the CIM_ElementSettingData association are required to be added to or removed from instances of the CIM_ResourceAllocationSettingData class.

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

7.3.5 Other types of virtual system configurations

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

7.3.6 CIM_VirtualSystemSettingData.Caption property

The implementation of the Caption property is optional.

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

If the Caption property is implemented for the CIM_ComputerSystem class, the value of the Caption property in the instance of the CIM_VirtualSystemSettingData class in the "state" virtual system configuration of a virtual system shall be identical to the value of the Caption property in the instance of the CIM_ComputerSystem class representing the virtual system.

7.3.7 CIM_VirtualSystemSettingData.Description property

The implementation of the Description property is optional.

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

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

7.3.8 CIM_VirtualSystemSettingData.ElementName property

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

If the ElementName property is implemented for the CIM_ComputerSystem class, the value of the ElementName property in the instance of the CIM_VirtualSystemSettingData class in the "state" virtual system configuration of a virtual system shall be identical to the value of the ElementName property in the instance of the CIM_ComputerSystem class representing the virtual system.

7.3.9 CIM_VirtualSystemSettingData.VirtualSystemIdentifier property

The implementation of the VirtualSystemIdentifier property is optional.

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

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

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

7.3.10 CIM_VirtualSystemSettingData.VirtualSystemType property

The implementation of the VirtualSystemType property is optional.

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

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

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

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

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

7.3.11 CIM_ElementSettingData.IsDefault property

The IsDefault property shall be implemented. Each top-level CIM_VirtualSystemSettingData instance in a "state" virtual system configuration and the top-level CIM_VirtualSystemSettingData instance in the related "defined" virtual system configuration shall be associated through an instance of the CIM_ElementSettingData association. The value of the IsDefault property shall be used to designate the "defined" virtual system configuration among all configurations associated with the "state" virtual system configuration.

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

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

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

7.3.12 CIM_ElementSettingData.IsNext property

The implementation of the IsNext property is optional.

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

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

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

- The IsNext property shall have one of the following values:
  - a value of 0 (Unknown) if it is not known whether the referenced virtual system configuration will be used for the next activation
  - a value of 1 (Is Next) if the referenced virtual system configuration is established to be used for subsequent activations of the virtual system
  - a value of 3 (Is Next For Single Use) if the referenced virtual system configuration is established to be used for just the next activation of the virtual system in preference of the default and or the persistently established next configuration.
  - In all other cases the IsNext property shall have a value of 2 (Is Not Next). In this case the "default" virtual system configuration is used for the next virtual system activation.
- In the set of all virtual system configurations that are associated with a top-level instance of the CIM_VirtualSystemSettingData class in a "state" virtual system configuration, there shall be
  - at most one configuration that is referenced by an instance of the CIM_ElementSettingData association with a value of 1 (Is Next)
  - at most one configuration that is referenced by an instance of the CIM_ElementSettingData association with a value of 3 (Is Next For Single Use) for the IsNext property. This configuration shall be given preference over one that is designated with a value of 1 (Is Next).

7.4 Profile registration

7.4.1 This profile

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

7.4.2 Scoped profiles

For scoped profiles the following conditions shall be met:

- The instance of the CIM_RegisteredProfile class that represents the implementation of this profile and instances of the CIM_RegisteredProfile class that represent an implementation of the scoped profile shall be associated through instances of the CIM_ReferencedProfile association.
- One of the following conditions shall be met:
  a) Instances of the CIM_ElementConformsToProfile association shall associate any central instance of the scoped profile that is associated to the central instance of this profile through the CIM_SystemDevice association, and the instance of the CIM_RegisteredProfile class that represents an implementation of the scoped profile.
  b) No instances of the CIM_ElementConformsToProfile association shall associate any central instance of the scoped profile that is associated to the central instance of this profile through the CIM_SystemDevice association, and the instance of the CIM_RegisteredProfile class that represents an implementation of the scoped profile.

7.5 Capabilities

7.5.1.1 CIM_EnabledLogicalElementCapabilities.RequestedStatesSupported property

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

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

7.6 Client state management

The implementation of client state management is conditional.

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

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

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

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

- implement the CIM_EnabledLogicalElementCapabilities class according to 7.5.1.1 to indicate the availability of client state management support, and the set of state transitions that are applicable

- implement method RequestStateChange( )
- if it implements DSP1027 for virtual systems, implement the RequestPowerStateChange( ) method

7.7 Power state management

The implementation of power state management is optional.

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

The implementation of power state management requires the implementation of DSP1027. DSP1027 specifies

- how to indicate that DSP1027 is implemented
- how to implement the CIM_PowerManagementService class and the CIM_Associated-PowerManagementService association

If the observation of power states is implemented as specified by DSP1027, then the observation of virtual system states as defined in 7.1.1 and the observation of virtual system state transitions as defined in 7.1.2 shall also be implemented. If power state management is implemented as specified by DSP1027, then client state management as specified in 8.1.1 shall also be implemented.

NOTE The implementation of DSP1027 in the context of virtual systems is intended to support clients that use facilities specified by DSP1027 in preference of facilities specified in this profile (DSP1057). For example, such clients may use the CIM_AssociatedPowerManagementService.PowerState property in favor of the CIM_ComputerSystem.EnabledState property to determine the virtual system state, or may use the CIM_PowerManagementService.RequestPowerStateChange( ) method in favor of the CIM_EnabledLogical-Element.RequestStateChange( ) method to effect virtual system state transitions.

7.7.1 CIM_AssociatedPowerManagementService.PowerState property

The implementation of the PowerState property is conditional.

Condition: All of the following

- Client state management is implemented (see 7.5.1.1)
- DSP1027 is implemented.

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

The CIM_AssociatedPowerManagmentService association shall be used to convey the virtual system state in addition to the CIM_ComputerSystem.EnabledState property. In this case, the PowerState property shall contain a value that corresponds to the virtual system state as defined in Table 2. For example, if the virtual system state is "active", then the PowerState property shall have a value of 2 (On).

A client preferring to use mechanisms defined by DSP1027 may translate the value of the PowerState property of an instance of the CIM_AssociatedPowerManagementService association that is referring to an instance of the CIM_ComputerSystem class representing a virtual system by translating that value according to Table 2. For example, if the PowerState property has a value of 2 (On), then a client shall conclude that the virtual system state is "active".

8 Methods

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

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

8.1 Extrinsic methods

8.1.1 CIM_ComputerSystem.RequestStateChange( ) method

The implementation of the RequestStateChange( ) method is conditional.

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

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

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

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.

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

No standard messages are defined.

8.1.2 CIM_PowerManagementService.RequestPowerStateChange( ) method

The implementation of the RequestPowerStateChange( ) method is conditional.

Condition: All of the following

- Client state management is implemented (see 7.5.1.1)
- DSP1027 is implemented.

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

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

Table 5 – CIM_PowerManagementService.RequestPowerStateChange( ) method: Parameters

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

For return code values, see the CIM Schema description of this method in the CIM_PowerManagementService class.

No standard messages are defined.

8.1.2.1 PowerState parameter

The PowerState parameter encodes the requested new virtual system state.

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

8.1.2.2 ManagedElement parameter

The value of the ManagedElement parameter shall be used to identity the virtual system to which the operation applies.

8.1.2.3 Time parameter

The implementation of the Time parameter is optional.

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

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

8.2 Profile conventions for operations

The default list of operations for all classes is:

- GetInstance( )
- EnumerateInstances( )
- EnumerateInstanceNames( )

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

- Associators( )
- AssociatorNames( )
- References( )
- ReferenceNames( )

8.2.1 CIM_ComputerSystem

All operations in the default list in 8.2 shall be implemented as defined in DSP0200.

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

8.2.2 CIM_ConcreteJob

All operations in the default list in 8.2 shall be implemented as defined in DSP0200.

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

8.2.3 CIM_ElementSettingData

All operations in the default list in 8.2 shall be implemented as defined in DSP0200.

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

8.2.4 CIM_EnabledLogicalElementCapabilities

All operations in the default list in 8.2 shall be implemented as defined in DSP0200.

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

8.2.5 CIM_ReferencedProfile

All operations in the default list in 8.2 shall be implemented as defined in DSP0200.

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

8.2.6 CIM_RegisteredProfile

All operations in the default list in 8.2 shall be implemented as defined in DSP0200.

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

8.2.7 CIM_VirtualSystemSettingData

All operations in the default list in 8.2 shall be implemented as defined in DSP0200.

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

8.2.8 CIM_VirtualSystemSettingDataComponent

All operations in the default list in 8.2 shall be implemented as defined in DSP0200.

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

9 Use-cases

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

9.1 Virtual system detection and inspection

This set of use cases describes how a client can

- discover virtual systems
- determine the state and properties of a virtual system
- determine the "defined" virtual system configuration
- determine the virtual system structure
- determine resource type support
- detect and inspect the boot configuration for the virtual system

9.1.1 Discover conformant virtual systems using SLP

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

1) The service location protocol (SLP) is used to locate WBEM services where this profile is implemented. A WBEM service using SLP facilities provides information about itself to SLP in form of an SLP service template. The service template may contain information about the set of profiles that is implemented at the WBEM service.

2) Normal CIM enumeration and association resolution is used to find instances of the CIM_ComputerSystem class that represent central instances of this profile.

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

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

1) The client invokes the SLPFindSrvs( ) SLP function:

   – The value of the srvtype parameter is set to "service:wbem"

   – The value of the scopelist parameter is set to "default"

   – The value of the filter parameter is set to "(RegisteredProfilesSupported=DMTF:Virtual System)"

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

2) The client contacts each of the WBEM services and enumerates or queries the CIM_RegisteredProfile class.

   • As input, the client needs to use the address information of one server obtained in step 1) and issue the intrinsic EnumerateInstanceNames( ) CIM operation on the CIM_RegisteredProfile class. Alternatively, the client may issue the intrinsic ExecuteQuery CIM operation and specify a where clause that, for example, limits the value ranges for the RegisteredName and RegisteredVersion properties of the CIM_RegisteredProfile class.

   • As a result, the client receives a list of references to instances of the CIM_RegisteredProfile class that represent implementations of this profile at the intended target location. On a query operation this list already is limited according to the input selection criteria.

3) The client selects one reference and resolves the CIM_ElementConformsToProfile association from the instance of the CIM_RegisteredProfile class to instances of the CIM_ComputerSystem class.

   • As input, the client needs to provide the reference to an instance of the CIM_RegisteredProfile class that was selected from the result set obtained in step 2.

   • As a result, the client receives a list of references referencing instances of the CIM_ComputerSystem class that represents virtual systems.

Result: The result is that the client knows a set of references referencing instances of the CIM_ComputerSystem class that represent virtual systems that are central instances of this profile.

9.1.2 Determine a virtual system's state and other properties

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

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

1) The client calls the intrinsic GetInstance( ) CIM operation with the InstanceName parameter referencing the instance of the CIM_ComputerSystem class that represents the virtual system as the input parameter. As a result the client receives an instance of the CIM_ComputerSystem class that describes the virtual system.

2) The client uses the value of the EnabledState property to determine the virtual system state according to the translation rules specified in 7.1.1.

Result: The client knows the property set defined by the CIM_ComputerSystem class describing the affected virtual system, in particular the virtual system state. Many virtual system properties and in particular the virtual system state may change any time; consequently, the result only describes the virtual system at the moment it was provided by the instrumentation.

9.1.3 Determine the "defined" virtual system configuration

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

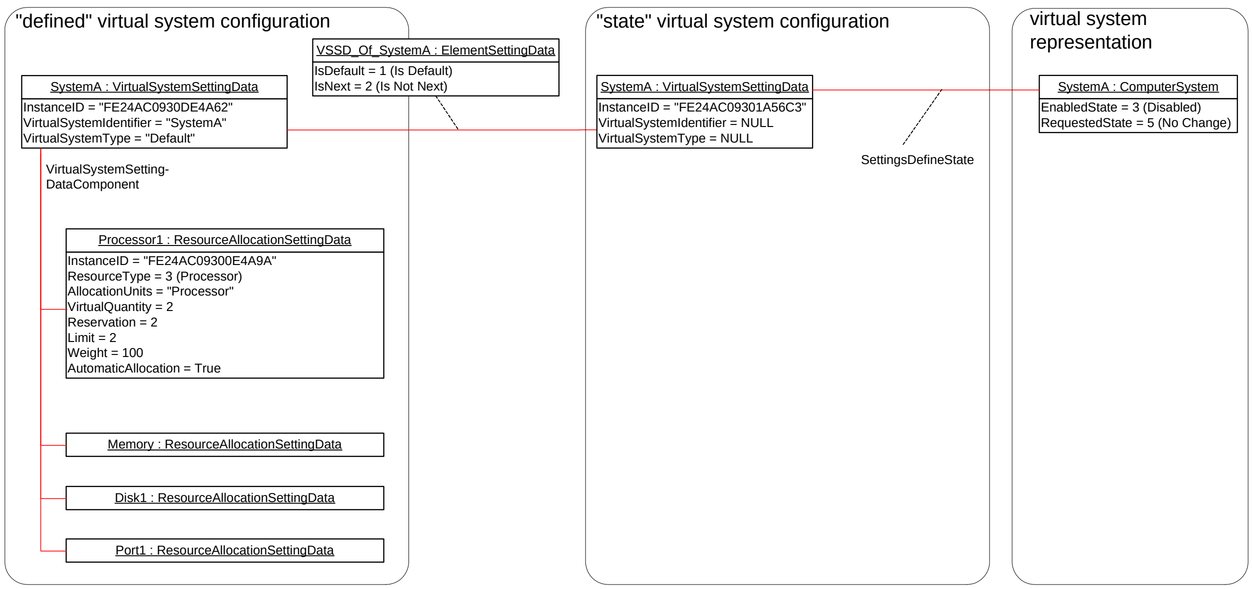


Figure 5 – Sample virtual system configuration

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

1) The client resolves the CIM_SettingsDefineState association from the instance of the CIM_ComputerSystem class representing the virtual system to the top-level instance of the CIM_VirtualSystemSettingData class in the "state" Virtual System configuration.

2) The client resolves the CIM_ElementSettingData association from the "state" instance of the CIM_VirtualSystemSettingData class that represents the virtual aspects of the virtual system to instances of the CIM_VirtualSystemSettingData class with the constraint that the CIM_ElementSettingData.IsDefault property has a value of 2 (IsDefault). The result is a reference referring to an instance of the CIM_VirtualSystemSettingData class that represents the top-level object of the desired virtual system configuration.

3) The client obtains the referenced instance of the CIM_VirtualSystemSettingData class using the intrinsic getInstance( ) CIM operation and analyzes its properties. For example, the client might analyze the VirtualSystemIdentifier property, which reflects the (end-user interpretable) name used for the virtual system ("SystemA" in Figure 5), or the VirtualSystemType property, which reflects a particular virtual system type that the virtualization platform assigned for the respective virtual system ("Default" in Figure 5). Note that the InstanceID property contains an opaque ID for the instance; the structure of InstanceID values is implementation dependent and not known to clients.

4) The client resolves the CIM_VirtualSystemSettingDataComponent association from the instance of the CIM_VirtualSystemSettingData class to instances of the CIM_ResourceAllocationSetting-Data class.

5) The client obtains instances of the CIM_ResourceAllocationSettingData class using the intrinsic getInstance( ) CIM operation and analyzes properties of these instances. For example, the client might analyze the Reservation property. The Reservation property reflects the amount of host resource that is allocated for the virtual resource while the virtual system is instantiated.

Result: The client knows the virtual system configuration in terms of one instance of the CIM_VirtualSystemSettingData class and a set of aggregated instances of the CIM_ResourceAllocationSettingData class.

9.1.4 Determine the virtual system structure

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

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

To avoid cluttering the diagram, an instance of the CIM_ElementSettingData association between the "defined" and the "state" instance of the CIM_ResourceAllocationSettingData class is shown for processor configurations only.

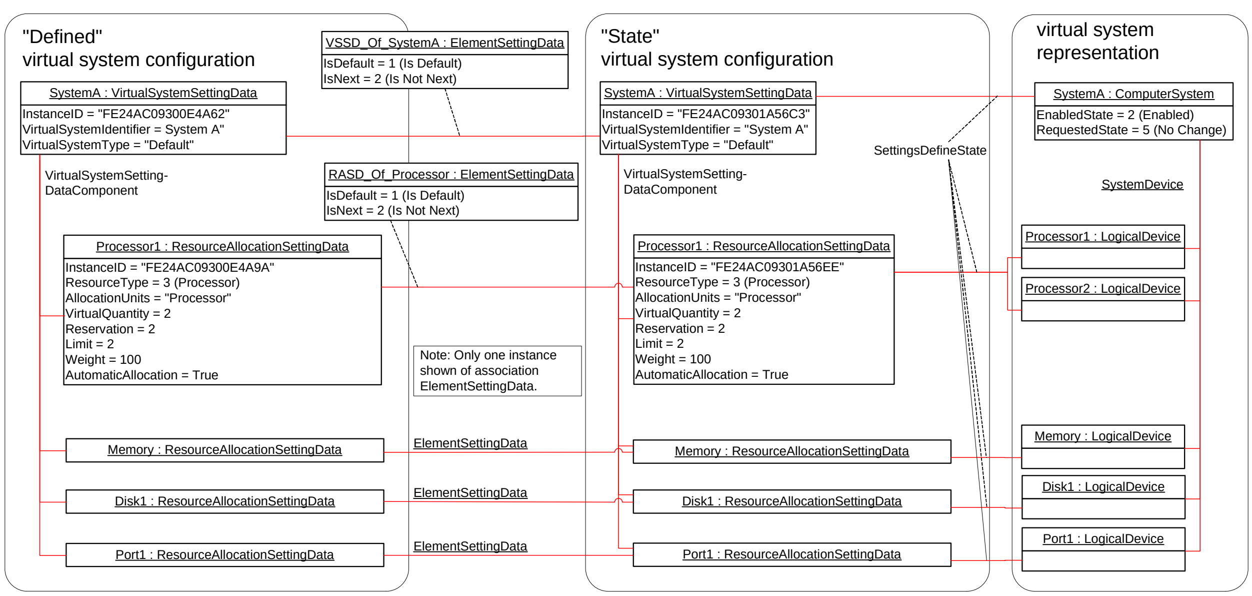


Figure 6 – Sample virtual system in "active" state

A client can determine the virtual system structure as follows:

1) The client may apply the use case described in 9.1.2 to obtain state information and other properties of the CIM_ComputerSystem instance that represents the virtual system.

2) The client may apply the use case described in 9.1.3 to obtain information about the virtual system configuration.

3) The client resolves the CIM_SystemDevice association from the instance of the CIM_ComputerSystem class that represents the virtual system to instances of the CIM_LogicalDevice class.

4) The client obtains instances of the CIM_LogicalDevice class that were returned in step 3) and analyzes properties of interest.

Result: The client knows the virtual system structure as expressed through the virtual system configurations ("defined" and "state") and through the set of objects representing the virtual system and its components.

9.1.5 Determine resource type support

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

DSP1033 defines how an implementation of a profile advertises conformance to the profile. For example, Figure 7 shows an instance of the CIM_ComputerSystem class named VS1 that is associated to an instance of the CIM_RegisteredProfile class named RPVS.

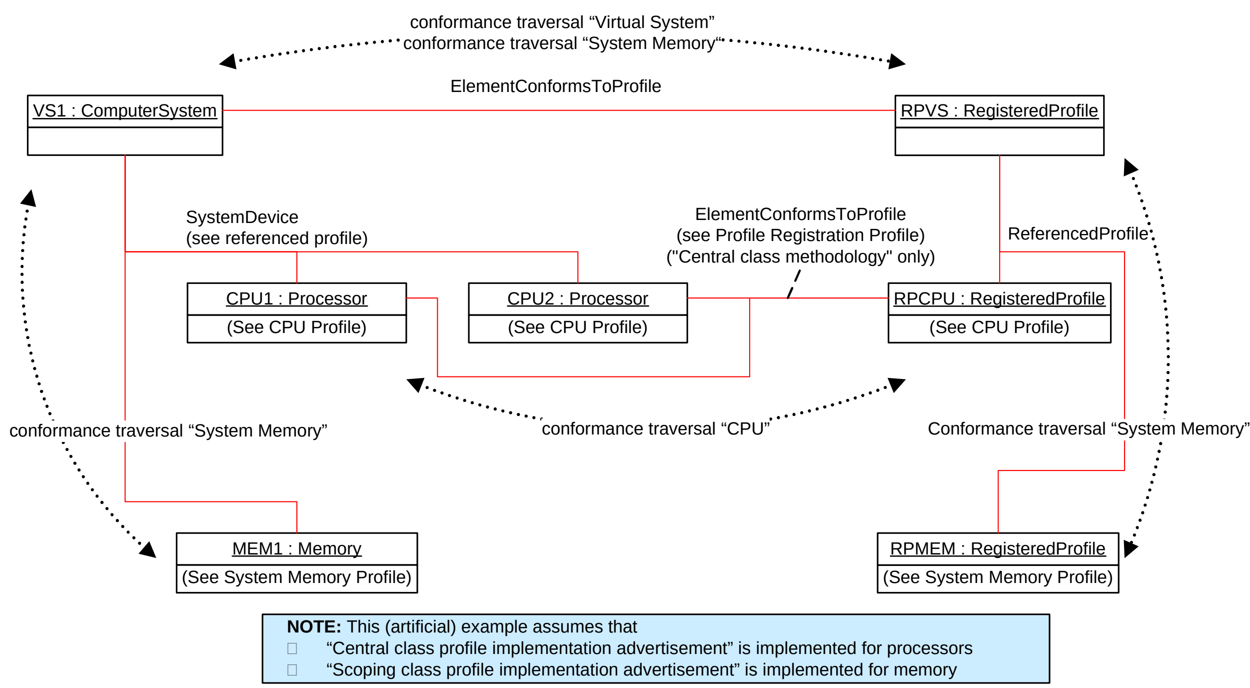


Figure 7 – Instance diagram: Profile conformance of scoped resources

If profiles addressing the management of scoped resources are implemented, then DSP1033 specifies to implement either the "central class profile implementation advertisement methodology" or the "scoped class profile implementation advertisement methodology".

With the "central class profile implementation advertisement methodology" the approach is straight forward: Any central instance of a profile is associated with the respective instance of the CIM_RegisteredProfile class through the CIM_ElementConformsToProfile association.

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

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

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

The first part of this use case determines the profile implementation advertisement methodology for processors.

1) The client resolves the CIM_ElementConformsToProfile association to locate associated instances of the CIM_RegisteredProfile class, invoking the intrinsic AssociatorNames( ) CIM operation as follows:
   - The value of the ObjectName parameter references the instance of the CIM_ComputerSystem class.
   - The value of the AssocClass parameter is set to "CIM_ElementConformsToProfile".
   - The value of the ResultClass parameter is set to "CIM_RegisteredProfile".
   - The result is a list of references referring to instances of the CIM_RegisteredProfile class representing implementations of this profile; if the operation is successful, the size of the result set is 1.
2) The client resolves the CIM_ReferencedProfile association to locate scoped instances of the CIM_RegisteredProfile class, invoking the intrinsic Associators( ) CIM operation as follows:
   - The value of the ObjectName parameter is set to the reference referring to the instance of the CIM_RegisteredProfile class obtained in step 1).
   - The value of the AssocClass parameter is set to "CIM_ReferencedProfile".
   - The value of the ResultClass parameter is set to "CIM_RegisteredProfile".
   - The result is a list of instances of the CIM_RegisteredProfile class representing implementations of scoped profiles.
3) The client iterates over the list obtained in step 2), selecting only instances where the RegisteredName property has a value of "CPU".
   - The result is a list of instances of the CIM_RegisteredProfile class that represents implementations of scoped profiles implementing DSP1022 (CPU Profile).
4) The client resolves the CIM_ElementConformsToProfile association for each of the instances of the CIM_RegisteredProfile class from step 3) to locate at least one associated instance of the CIM_Processor class, invoking the intrinsic Associators( ) CIM operation as follows:
   - The value of the ObjectName parameter is set to the reference taken from the instance of the CIM_RegisteredProfile class obtained in step 3).
   - The value of the AssocClass parameter is set to "CIM_ReferencedProfile".
   - The value of the ResultClass parameter is set to "CIM_Processor".
   - The result is a list of instances of the CIM_Processor class that are central instances of DSP1022; the list may be empty.

If for any of the results from step 4) at least one instance of the CIM_Processor class was detected, then the central class profile implementation advertisement methodology is applied by the implementation with respect to implementations of DSP1022; this is the case in this example. If no such instances were detected, then the scoping class profile implementation advertisement methodology would have been applied.

At this point the client has validated that DSP1022 is implemented as a scoped profile of this profile, and that the central class profile implementation advertisement methodology is applied by the implementation with respect to DSP1022.

In the second part of this use case it is now the responsibility of the client for any detected scoped instance of the CIM_Processor class to validate that DSP1022 is indeed implemented. The use case describes how to locate such instances, and perform the validation:

1) Client resolves the CIM_SystemDevice association from the central instance to associated virtual resources, invoking the intrinsic AssociatorNames( ) CIM operation as follows:
   - The value of the ObjectName parameter is set referring to the instance of the CIM_ComputerSystem class.
   - The value of the AssocClass parameter is set to "CIM_SystemDevice".
   - The value of the ResultClass parameter is set to "CIM_Processor".
   - The result is a list of references referring to scoped instances of the CIM_Processor class representing virtual processors.
2) For each reference returned by step 1) the client resolves the CIM_ElementConformsToProfile association to locate associated instances of the CIM_RegisteredProfile class, invoking the intrinsic Associators( ) CIM operation as follows:
   - The value of parameter ObjectName is set referring to an instance of the CIM_Processor class.
   - The value of the AssocClass parameter is set to "CIM_ElementConformsToProfile".
   - The value of the ResultClass parameter is set to "CIM_RegisteredProfile".
   - The result is a list of instances of the CIM_RegisteredProfile class; if the operation is successful, the size of the list is either 0 or 1. A size of 1 indicates that a version of DSP1022 is implemented for the particular processor; a size of 0 indicates that DSP1022 is not implemented for the particular processor.

Result: The client knows the set of scoped instances of the CIM_Processor class that represents processors of the assumed virtual system, and whether the instances are central instances of DSP1022, that is, whether DSP1022 is implemented in the context of these instances.

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

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

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

3) The client resolves the CIM_ElementConformsToProfile association to locate associated instances of the CIM_RegisteredProfile class, invoking the intrinsic AssociatorNames( ) CIM operation as follows:
   - The value of the ObjectName parameter is set to the reference referring to the instance of the CIM_ComputerSystem class.
   - The value of the AssocClass parameter is set to "CIM_ElementConformsToProfile".

- The value of the ResultClass parameter is set to "CIM_RegisteredProfile".
- The result is a list of references referring to instances of the CIM_RegisteredProfile class representing implementations of this profile; if the operation is successful, the size of the result set is 1.

4) The client resolves the CIM_ReferencedProfile association to locate scoped instances of the CIM_RegisteredProfile class, invoking the intrinsic Associators( ) CIM operation as follows:
   - The value of parameter ObjectName is set to the reference referring to the instance of the CIM_RegisteredProfile class obtained in step 1).
   - The value of the AssocClass parameter is set to "CIM_ReferencedProfile".
   - The value of the ResultClass parameter is set to "CIM_RegisteredProfile".
   - The result is a list of instances of the CIM_RegisteredProfile class that represent implementations of scoped profiles.
5) The client iterates over the list obtained in step 2), selecting only instances where the RegisteredName property has a value of "System Memory".
   - The result is a list of instances of the CIM_RegisteredProfile class that represents implementations of scoped profiles implementing DSP1026 (*System Memory Profile*).
6) The client resolves the CIM_ElementConformsToProfile association for each of the instances of the CIM_RegisteredProfile class from step 3) to locate at least one associated instance of the CIM_Memory class, invoking the intrinsic Associators( ) CIM operation as follows:
   - The value of the ObjectName parameter is set to the reference taken from the instance of the CIM_RegisteredProfile class obtained in step 3).
   - The value of the AssocClass parameter is set to "CIM_ElementConformsToProfile".
   - The value of the ResultClass parameter is set to "CIM_Memory".
   - The result is a list of instances of the CIM_Memory class that are central instances of the scoped DSP1026. The list may be empty.

If for any of the results from step 6) no instance of the CIM_Memory class was detected, then the scoping class profile implementation advertisement methodology is applied by the implementation with respect to implementations of DSP1026; this is the case in this example. If any such instances were detected, then the central class profile implementation advertisement methodology would have been applied.

At this point the client has validated that DSP1026 is implemented as a scoped profile of this profile, and that the scoping class profile implementation advertisement methodology is applied by the implementation with respect to DSP1026.

In the second part of this use case the client now may assume for any detected scoped instance of the CIM_Memory class that DSP1026 is implemented. The use case describes how to locate such instances:

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

Result: The client knows the set of scoped instances of the CIM_Memory class that represents memory in the assumed virtual system, and that these are central instances of DSP1026.

9.1.6 Determine the next boot configuration

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

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

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

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

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

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

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

   The result of this step is an instance of the CIM_ElementSettingData association where the SettingData property refers to the instance of the CIM_BootConfigSetting class that is used for the next boot process.

3) The client obtains the instance of the CIM_BootConfigSetting class, using the intrinsic GetInstance( ) CIM operation with the InstanceName parameter referring to that instance.

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

9.2 Virtual system operation

This set of use cases describes how a client can perform basic operations on virtual system, like activating, deactivating, pausing or resuming a virtual system.

9.2.1 Change virtual system state

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

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

1) The client applies the rules outlined in 7.1.2 to determine a value for the RequestedState parameter of the CIM_EnabledLogicalElement.RequestStateChange( ) method that designates the intended state transition.

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

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

4) The client invokes the RequestStateChange method on the instance of the CIM_ComputerSystem class that represents the virtual system, using a value for the RequestedState parameter as determined in step 1).

5) The client checks the return code.

   – If the return code is zero, the virtual system state transition was performed as requested.

   – If the return code is 1, the RequestStateChange method is not implemented by the implementation. This should not occur if the checks above were performed.

   – If the return code is 2, an error occurred.

   – If the return code is 0x1000, the implementation has decided to perform the state transition as an asynchronous task. The client may monitor progress by analyzing the instance of the CIM_ConcreteJob class returned through the Job parameter.

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

Result: The virtual system performs the intended virtual system state transition. The client may next obtain the actual virtual system state by, for example, following the procedures outlined the use case in 9.1.2.

9.2.2 Activate virtual system

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

1) The client applies the steps in the use case described in 9.2.1 to perform an "activate" transition, for example using a value of 2 (Enabled) for the RequestedState parameter.

2) The client verifies that the operation was executed successfully, making sure that either a return code of 0 results or, if the state change is performed as an asynchronous task, by checking that the result of the respective instance of the CIM_ConcreteJob class indicates a successful completion.

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

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

10 CIM elements

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

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

Table 6 – CIM elements: Virtual System Profile

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

10.1 CIM_AffectedJobElement

The implementation of the CIM_AffectedJobElement association is conditional.

Condition: The CIM_ConcreteJob class is implemented; see 10.3.

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

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

Table 7 lists the requirements for this association.

Table 7 – Association: CIM_AffectedJobElement

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

10.2 CIM_ComputerSystem

The use of the CIM_ComputerSystem class is specialized in DSP1052 and further refined in this profile.

The requirements in Table 8 are in addition to those mandated by DSP1052.

Table 8 – Class: CIM_ComputerSystem

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

10.3 CIM_ConcreteJob

The implementation of the CIM_ConcreteJob class is conditional.

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

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

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

Table 9 lists requirements for elements of this class.

Table 9 – Class: CIM_ConcreteJob

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

10.4 CIM_ElementConformsToProfile

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

Table 10 lists the requirements for this association.

Table 10 – Association: CIM_ElementConformsToProfile

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

10.5 CIM_ElementSettingData

The CIM_ElementSettingData association associates the top-level instance of the CIM_VirtualSystemSettingData class in a "state" virtual system configuration and top-level instances of the CIM_VirtualSystemSettingData class in other virtual system configurations.

Table 11 lists the requirements for this association.

Table 11 – Association: CIM_ElementSettingData

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

10.6 CIM_EnabledLogicalElementCapabilities

The use of the CIM_EnabledLogicalElementCapabilities class is specialized in DSP1052.

The requirements denoted in Table 12 are in addition to those mandated by DSP1052.

Table 12 – Class: CIM_EnabledLogicalElementCapabilities

Element	Requirement	Notes
RequestedStatesSupported[]	Mandatory	See 7.5.1.1.

10.7 CIM_PowerManagementService

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

10.8 CIM_ReferencedProfile

The implementation of the CIM_ReferencedProfile association is conditional.

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

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

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

This profile (DSP1057) refines requirements of DSP1033 by establishing conditions for the support of the CIM_ReferencedProfile association.

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

Table 13 contains the requirements for this association.

Table 13 – Association: CIM_ReferencedProfile

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

10.9 CIM_RegisteredProfile

The use of the CIM_RegisteredProfile class is specialized by DSP1033.

The requirements denoted in Table 14 are in addition to those mandated by DSP1033.

Table 14 – Class: CIM_RegisteredProfile

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

10.10 CIM_SettingsDefineState

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

Table 15 contains the requirements for this association.

Table 15 – Association: CIM_SettingsDefineState

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

10.11 CIM_VirtualSystemSettingData

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

Table 16 contains the requirements for this class.

Table 16 – Class: CIM_VirtualSystemSettingData

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

10.12 CIM_VirtualSystemSettingDataComponent

The implementation of the CIM_VirtualSystemSettingData component association is conditional.

Condition: Component profiles of this profile are implemented, such as DSP1044, DSP1045 or DSP1059.

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

An instance of the CIM_VirtualSystemSettingDataComponent association shall associate each instance of the CIM_VirtualSystemSettingData class representing the virtual aspects of a virtual system with instances of the CIM_ResourceAllocationSettingData class representing virtual aspects of virtual resources of that virtual system.

Table 17 contains the requirements for this association.

Table 17 – Association: CIM_VirtualSystemSettingDataComponent

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

Annex A
(Informative)

Virtual system modeling — background information

A.1 Concepts: Model, view, controller

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

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

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

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

A.2 Aspect-oriented modeling approach

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

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

- A virtual system should appear to a virtualization-unaware client exactly like a non-virtual computer system.
- The single-inheritance modeling approach is not suited for various management domains being modeled on top of the same set of base classes. For example, if the CIM_VirtualComputerSystem and CIM_PartitionedComputerSystem classes were both derived from the CIM_ComputerSystem class, then a particular instance could represent either a virtual system or a partitioned system, but not both.
- Many virtualization platforms support the concepts of virtual system definition and virtual system instance. The definition is a formal description of the virtual system; the instance is the internal representation of the virtual system in the "active" state. Ideally, both definition and instance are described using the same set of CIM classes.

Instead, a large part of the model specified by this profile is based on classes derived from CIM_SettingData:

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

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

A.3 Presence of model information

DSP1001 (the *Management Profile Specification Usage Guide*) requires an autonomous profile to specify a central class and a scoping class. DSP1052 specifies the CIM_ComputerSystem class for both the central and scoping class. This profile (DSP1057) specializes DSP1052, and thus is required to use the CIM_ComputerSystem class (or a derived class) for central and scoping class as well.

DSP1001 further requires that an instance of that class must be present at all times. Figure 8 illustrates that this requirement in some cases causes a potential model representation problem.

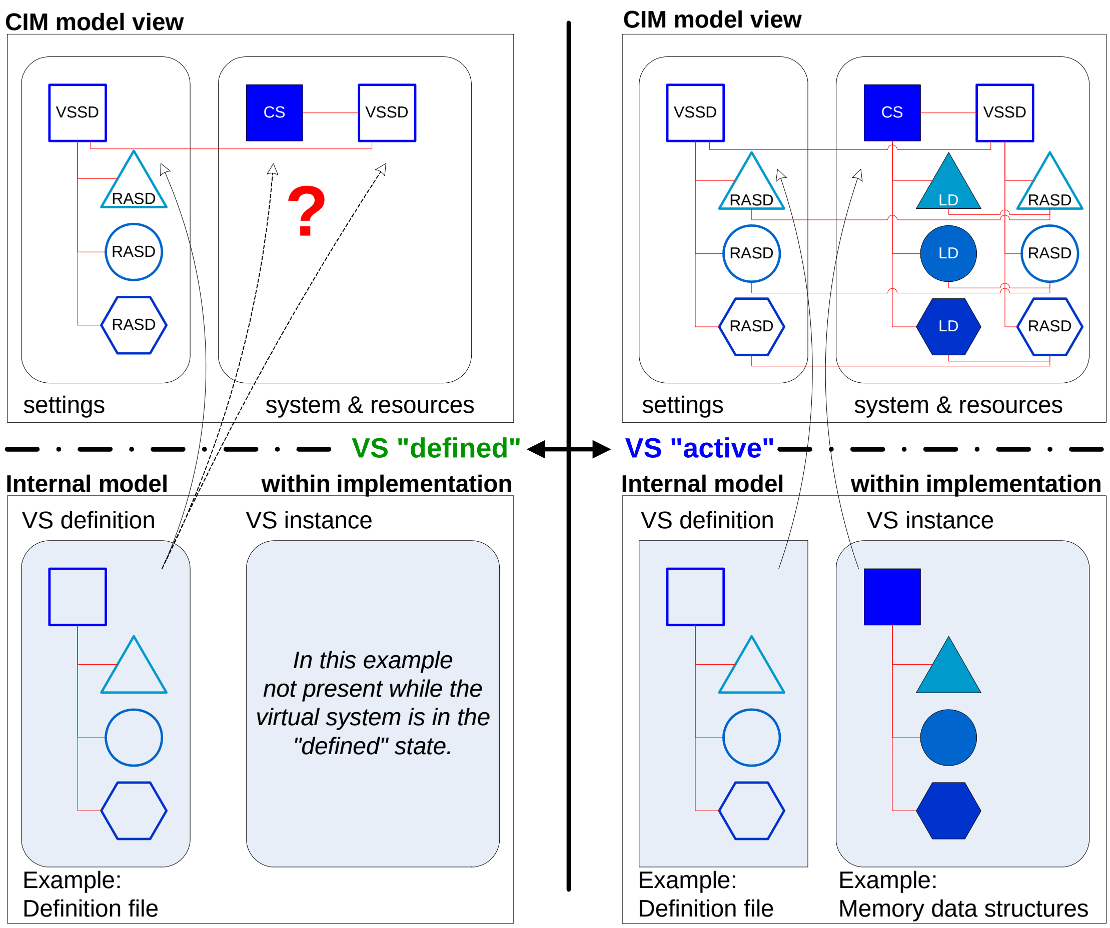


Figure 8 – State-dependent presence of model elements

The left side of Figure 8 shows a virtual system in the "defined" state. In this example the virtualization platform distinguishes between virtual system definition and virtual system instance; the virtual system instance does not exist while the virtual system is in the "defined" state. Nevertheless, the implementation is required to represent a (virtual) computer system through an instance of the CIM_ComputerSystem class during its complete lifecycle, including periods when the virtual system is only defined but not active and instantiated at the virtualization platform. This causes a model representation problem: Many properties of the CIM_ComputerSystem class (with instances labeled "CS" in Figure 8) model information about a stateful virtual system instance, but not about a stateless virtual system definition.

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

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

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

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

A.4 Model extension through settings

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

Annex B
(Informative)

Implementation details

B.1 Dual-configuration implementation approach

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

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

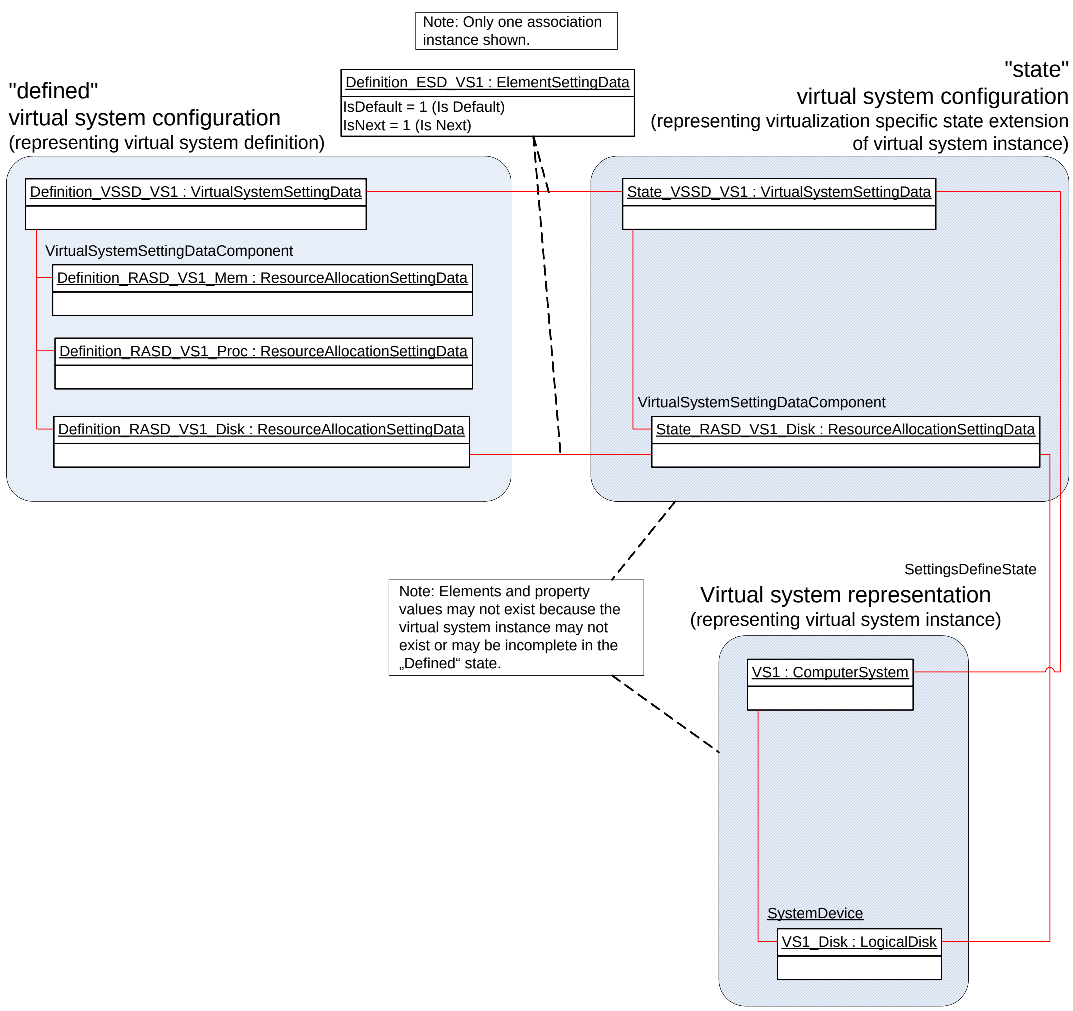


Figure 9 – Sample virtual system in "defined" state (Dual-configuration approach)

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

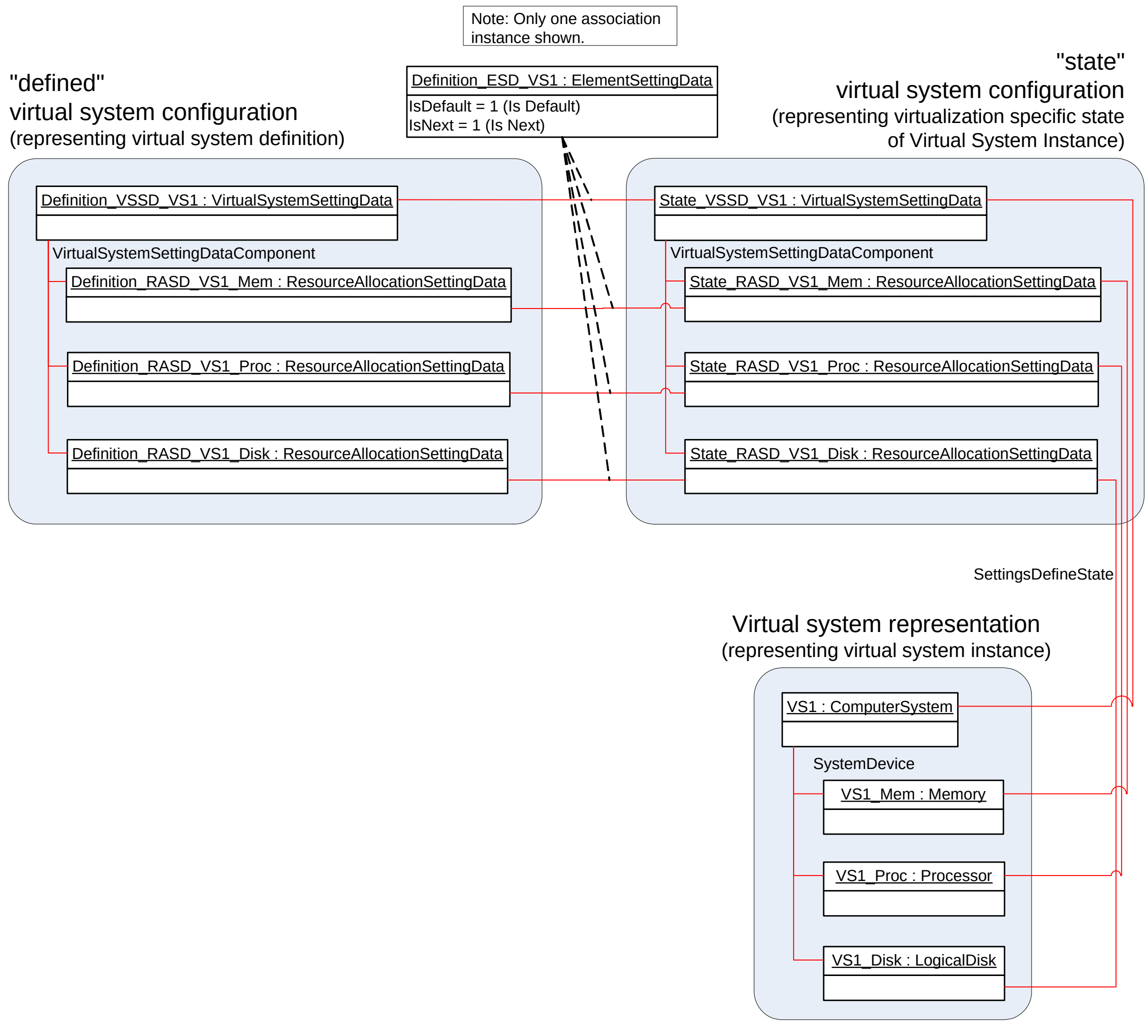


Figure 10 – Sample virtual system in a state other than "defined" (Dual-configuration approach)

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

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

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

B.2 Single-configuration implementation approach

Figure 11 shows an example in which a virtual system is in the "defined" state. Only one set of instances of the CIM_VirtualSystemSettingData class and the CIM_ResourceAllocationSettingData class compose a single virtual system configuration instance that acts as the "defined" and as the "state" virtual system configuration. The single configuration instance is associated to the instance of the CIM_ComputerSystem class representing the virtual system through an instance of the CIM_SettingsDefineState association.

Note that in this example virtual resource VS1_Disk has a persistently allocated resource that remains allocated regardless of the virtual system state. Consequently, an instance of the CIM_LogicalDisk class tagged VS1_Disk represents the disk in the "defined" state already, and virtualization-specific properties are represented by an instance of the CIM_ResourceAllocationSettingData class tagged State_RASD_VS1_Disk in the "state" virtual system configuration that is associated through the CIM_SettingsDefineState association.

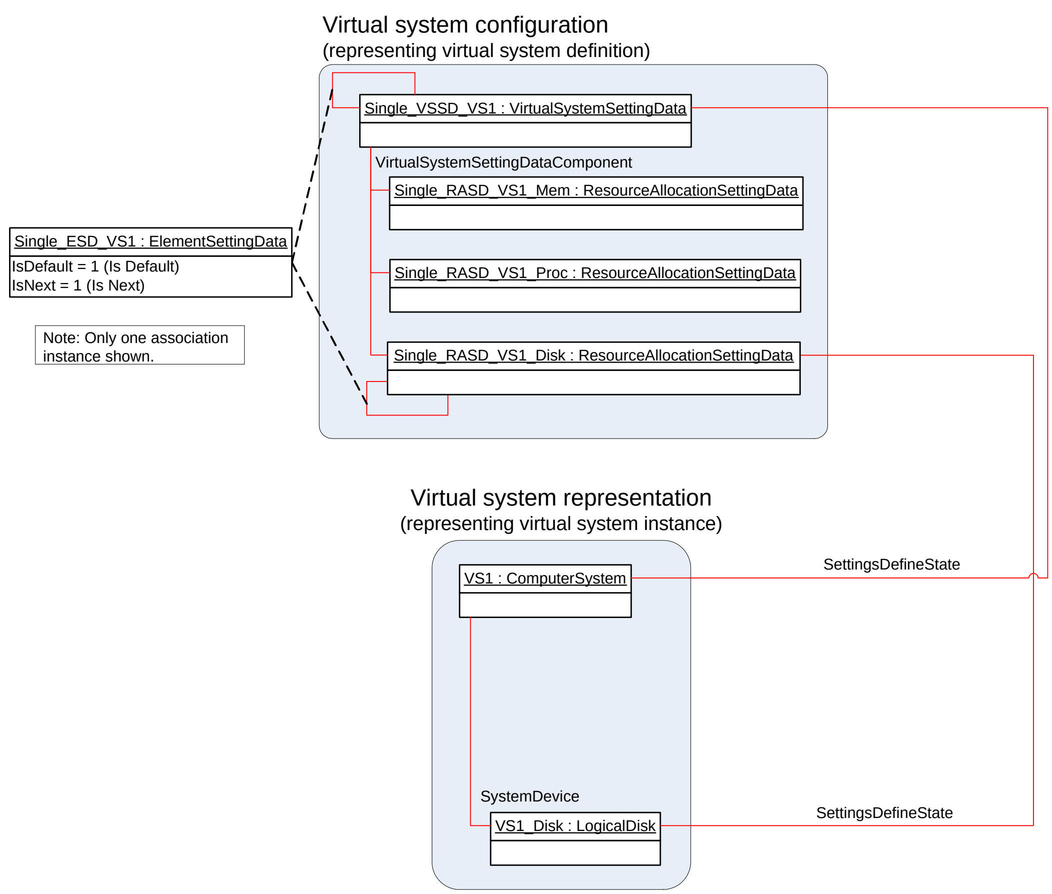


Figure 11 – Sample virtual system in the "defined" state (Single-configuration approach)

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

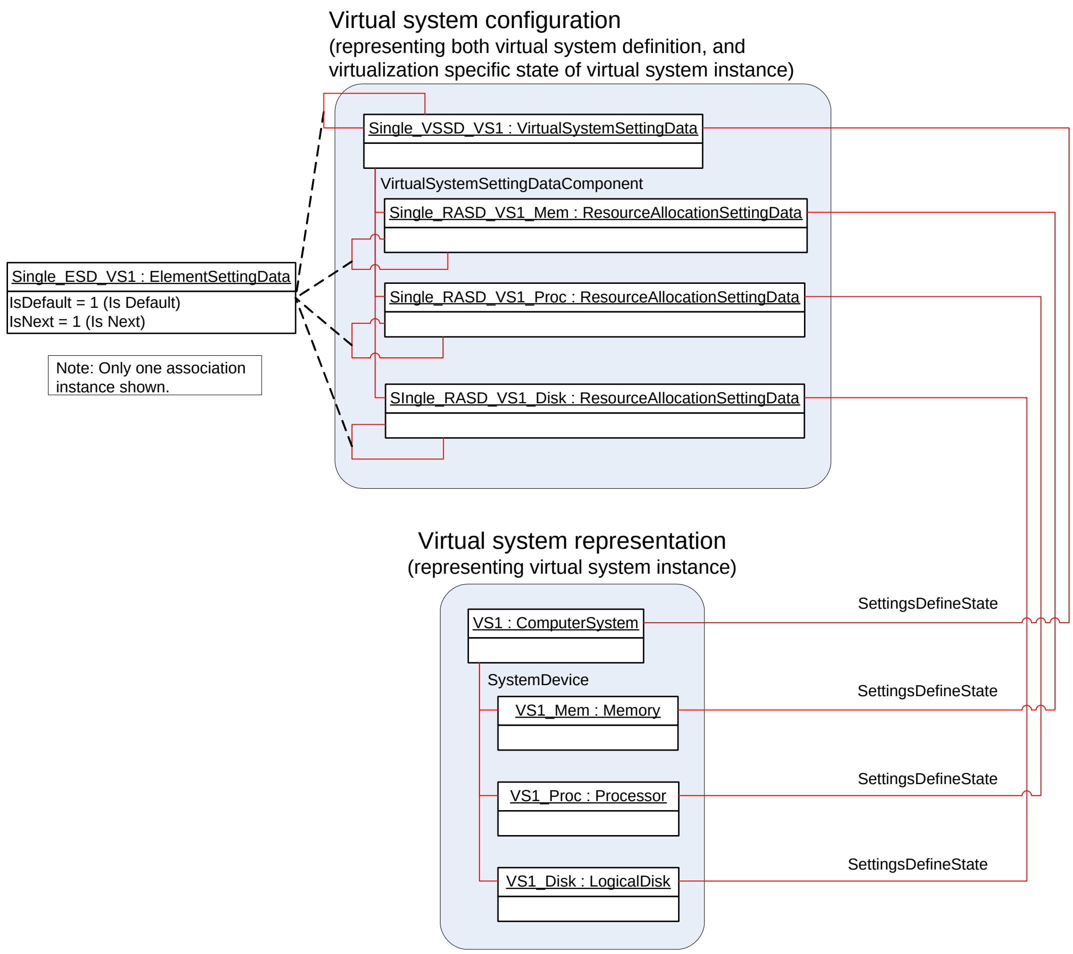


Figure 12 – Sample virtual system in a state other than "defined" (Single-configuration approach)

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

NOTE If DSP1041 is implemented for a particular resource type, it may require that, as virtual resources are allocated or de-allocated and respective instances of the CIM_LogicalDevice class are created or destroyed in the virtual system representation, these instances are connected to their counterpart in the "state" virtual system configuration through respective instances of the CIM_SettingsDefineState association. DSP1041 may also require that the instances in the "state" virtual system configuration are connected to their counterpart in the "defined" virtual system configuration through respective instances of the CIM_ElementSettingData association with the IsDefault property set to 1 (Is Default); in the single-configuration implementation approach, these association instances connect elements of the single virtual system configuration to themselves.

Annex C
(Informative)

Change Log

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