# CIM Physical Model White Paper

**CIM Version 2.7**

**Document Version 1.0 June 9, 2003**

## Abstract

The DMTF Common Information Model (CIM) is a conceptual information model for describing computing and business entities in enterprise and Internet environments. It provides a consistent definition and structure of data, using object-oriented techniques. The CIM Schema establishes a common conceptual framework that describes the managed environment.

The CIM Physical Common Model describes the information related to physical inventory and asset management, describing enclosures, cards and physical components, and cabling information. Physical Elements occupy space and conform to the elementary laws of physics. They represent any element that has a physical identity – i.e., that can be touched or seen. The relationships between Physical Elements are defined as associations in the model, and mainly deal with containment and location.

The goal of this paper is to overview the concepts that are currently modeled in the CIM 2.7 Physical Model. This paper mirrors the organization of the classes as they are presented in the MOF and UML/Visio diagram.

## Notice

***DSP151*** ***Status: Preliminary***

## Table of Contents

# 1 Introduction

## 1.1 Overview

The CIM Physical Common Model describes the information related to physical inventory and asset management, describing enclosures, cards, physical components (such as processor and memory chips), and cabling information. Physical Elements occupy space and conform to the elementary laws of physics. They represent any element that has a physical identity – i.e., that can be touched or seen. The relationships between Physical Elements are defined as associations in the model, and mainly deal with containment and location.

It is important to remember that the abstractions in the Physical Model typically represent the physical make-up of a Computer System. They do NOT represent the functionality that the physical items are capable of providing. This functionality is represented by the abstractions on the logical side of the model – usually as subclasses of CIM_LogicalDevice, or as CIM_Services hosted on the Computer System. For example, there is no *physical* difference between the abstractions of a chassis that functions as a server, storage subsystem, or network printer. All contain cards (for example, network cards) that have mounted components (processor and memory chips), other packages such a power supply, and slots that may be used to house additional cards or packages. However, there is a huge difference in the functionality that these three kinds of systems provide. Logical Devices and Services (defined in the Core Model, as subclasses of Logical Element) realize this functionality.

## 1.2 Background Reference Material

In addition to this white paper, more information can be found in the following documents:

CIM Core and Common Models - Versions 2.0-2.7 - Downloadable from http://www.dmtf.org/standards/standard_cim.php

Common Information Model (CIM) Specification, V2.2, June 14, 1999 - Downloadable from http://www.dmtf.org/spec/cims.html

Unified Modeling Language (UML) from the Open Management Group (OMG) - Downloadable from http://www.omg.org/uml/

Desktop Management Interface (DMI) - Downloadable from http://www.dmtf.org/spec/dmis.html

Internet Engineering Task Force (IETF) - MIBs and Work Group information at http://www.ietf.org

Common Information Model (CIM) Concepts White Paper, DSP0110, June 2003 – Downloadable from http://www.dmtf.org/standards/published_documents.php

Core White Paper, DSP0111, June 2003 – Downloadable from http://www.dmtf.org/standards/published_documents.php

# 2 The Physical Model

## 2.1 Background and Assumptions

It is assumed that the reader is familiar with the concepts and terminology presented in the CIM Specification, CIM Concepts White Paper and the Core White Paper.

The Physical Model extends from the class structures and framework of the CIM Core Model. It is possible that additional objects and properties will be defined in subsequent releases of the Physical Model.

## 2.2 Conceptual Areas Addressed by the Model

There are four major conceptual areas covered by the physical model.

- **Packaging**
  Packaging describes concepts such as cards, frames, chassis, connectors, cabling, chips, tapes (in storage libraries), etc. These concepts derive from the CIM_PhysicalElement abstraction in the Core Model.
- **Location**
  Location describes the position or site of the physical packaging.
- **Replacement Sets**
  These sets group objects that should be removed and/or reinstalled together when repaired or upgraded.
- **Physical Capacity**
  Physical Capacity describes the capability/capacity of related hardware to contain and/or connect to a minimum and maximum number of other objects. The hardware being described is indicated using the Element Capacity association. The objects for which min and max values are reported are indicated by specific enumerations defined in subclasses of Physical Capacity. For example, the Configuration Capacity class is instantiated to describe that a disk array is capable of containing 50 disks, added in increments of one.

## 2.3 Packaging

It is import to keep the following modeling goals in mind when reviewing the packaging definitions:

- The physical environment is concerned with the physical entities themselves, not the functionality that they realize.

- The physical environment should be described starting with the highest-level enclosure and ending at the lowest point of supported repair. For example, it is not relevant to model the chip level if management and repair stops at the card level.

However, the reverse is true. If the supported management and repair levels are at the chip level, then it doesn't make sense to only model the card and not its components.

A Physical Element is defined as one of four subclasses/categorizes. They are **Physical Package, Physical Component, Physical Connector,** and **Physical Link**.

### 2.3.1 Physical Package

The Physical Package class describes general containers and frames, and provides management, maintenance, and repair information. It is further refined by the following subclasses:

- **Physical Frame** – Defines the high level concept of an enclosure. It is further refined by the **Chassis** and **Rack** subclasses.
- **Card** – Defines the basic concept of a card. It includes motherboards, backplanes, adapter cards, daughter cards, etc. It is further refined by **System Bus Card** subclass.
- **Storage Media Location** - Defines the shelf/hole/slot where a storage magazine or tape can be stored. It is further refined by the **Magazine** subclass.

Instances of Physical Packages contain other Physical Elements. This is expressed using the **Container** association or one of its specialized subclasses (**Card On Card, Chassis In Rack, Package In Chassis**).

### 2.3.2 Physical Component

Physical Component describes low-level hardware, such as chips and physical media. Physical Component is further refined by the following subclasses:

- **Physical Media** – The physical storage medium. It includes tape, floppies, and media within a hard disk drive. This is further refined by the **Physical Tape** subclass.
- **Chip** – The physical IC (integrated circuitry), such as processors, flash, ram, etc.. This is further refined by the **Physical Memory** subclass.

The **Packaged Component** association (a specialization of Container) is used to describe a Component mounted on or in a Physical Package. The **Memory On Card** relationship is a further specialization of Packaged Component.

### 2.3.3 Physical Connector

This class describes the connectors used to attach or link Physical Elements together (for example, RJ45 jacks, PCI slots, etc.).

Physical Connector is further refined by the **Slot** subclass. A slot describes the connector used to attach one card to another (for example, a backplane or motherboard).

The **Connector On Package** association (another specialization of Container) is used to describe a Physical Connector that is mounted on a Physical Package.

The **Package In Connector** association (a specialization of Dependency) is used to describe a Physical Package that is inserted into the Physical Connector. The **Package In Slot** relationship is a further specialization of Package In Connector, and **Card In Slot** further specializes Package In Slot.

The **Connected To** relationship (another specialization of Dependency) is used to describe physical connection of two or more connectors. The **Slot In Slot** specialization of Connected To describes that an adapter card is utilized to change the existing slot connector into another type of connector. The adapter card is responsible for correctly mapping the electrical/signal characteristic between the two slots.

The **Adjacent Slots** association is used to describe the relationship between two adjacent slots on a motherboard or backplane. The fact that two slots are adjacent can also be determined by interpreting the Location Within Container property of the Connector On Package association. However, the Adjacent Slots association also provides information on the spacing between the slots, and whether or not the use of one slot eliminates or reduces the functionality of the other.

### 2.3.4 Physical Link

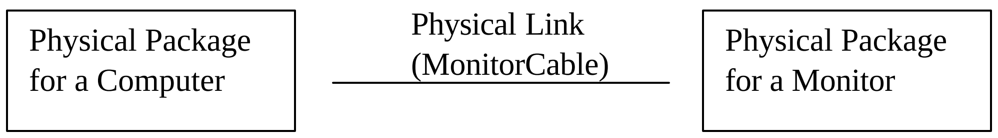
This class describes the cabling used between Physical Elements, such as connectors. This level of detail is not appropriate for all types of cabling or all physical environments. However, there are environments where the type and/or length of physical cabling impacts performance and reliability, or when the cabling is considered a physical asset of the business. (Examples of the latter are physical cabling laid between buildings of a campus.)

The **Elements Linked** association (a specialization of Dependency) indicates the Physical Elements that are connected.

The **Link Has Connector** relationship (a specialization of Component) is used to describe the connectors attached to the physical link/cable.

These two associations define different mechanisms to express the connection of two or more Physical Elements. The major difference between the approaches is whether or not the connector information is available or important.

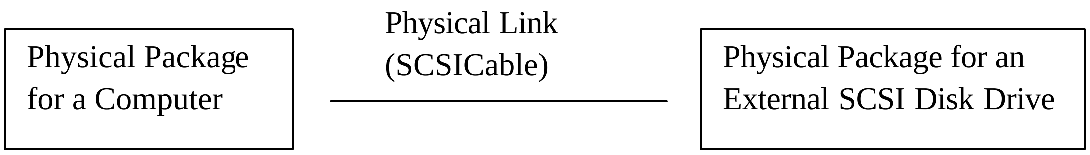
Consider the following example:


Physical Package for a Computer
Physical Link (MonitorCable)
Physical Package for a Monitor

It would be a bit of overkill to instantiate connectors on the Physical Link, since the type and gender of this connector is standardized. For this scenario, it would be sufficient to have the following 5 instances:

- Physical Package(1) – representing the computer
- Physical Package(2) – representing the monitor
- Physical Link(1) – representing the monitor cable
- Elements Linked(1) – representing the relationship between Physical Package(1) and Physical Link(1)
- Elements Linked(2) - representing the relationship between Physical Package(2) and Physical Link(1)

However, in examining the case below, it is not obvious what type and gender the connectors would be on the Physical Link or the Physical Packages.


Physical Package for a Computer
Physical Link (SCSICable)
Physical Package for an External SCSI Disk Drive

Therefore, we end up with two additional implementation scenarios, depending upon whether or not the Connectors are instantiated for the Physical Packages themselves. Let's first examine the situation where they are not. For this scenario, it would be sufficient to have the following nine instances:

- Physical Package(1) – representing the computer
- Physical Package(2) – representing the disk drive
- Physical Link(1) – representing the SCSI cable
- Physical Connector(1) – representing the connector on one end of the SCSI cable
- Physical Connector(2) – representing the connector on the other end of the SCSI cable
- Link Has Connector(1) – representing the relationship between Physical Link(1) and Physical Connector(1)
- Link Has Connector(2) – representing the relationship between Physical Link(1) and Physical Connector(2)
- Elements Linked(1) – representing the relationship between Physical Package(1) and Physical Link(1)
- Elements Linked(2) - representing the relationship between Physical Package(2) and Physical Link(1)

Now examine the case where the Connectors are instantiated for both the Physical Packages and the Physical Link. For this scenario, the following twelve instances are required:

- Physical Package(1) – representing the computer

- Physical Package(2) – representing the disk drive
- Physical Link(1) – representing the SCSI cable
- Physical Connector(1) – representing the connector on one end of the SCSI cable
- Physical Connector(2) – representing the connector on the other end of the SCSI cable
- Physical Connector(3) – representing the SCSI connector on the computer
- Physical Connector(4) – representing the SCSI connector on the disk drive
- Link Has Connector(1) – representing the relationship between Physical Link(1) and Physical Connector(1)
- Link Has Connector(2) – representing the relationship between Physical Link(1) and Physical Connector(2)
- Connector On Package(1) – representing the relationship between Physical Package(1) and Physical Connector(3)
- Connector On Package(2) – representing the relationship between Physical Package(2) and Physical Connector(4)
- Connected To(1) – representing the relationship between Physical Connector(1) and Physical Connector(3)
- Connected To(2) – representing the relationship between Physical Connector(2) and Physical Connector(4)

In the above example, the author chose to use Connected To, versus the Elements Linked relationship. No information is lost by taking this approach.

## 2.4 Location

There are three distinct location concepts:

- The first concept describes the location of a top-level or high-level physical package – such as a Rack (for example, positioning it within a campus, building, floor, and perhaps even wiring closet). This concept is addressed by the **CIM_Location** abstraction in the Core Model. The relationship between a CIM_Location and a Physical Element is defined by the association, **CIM_PhysicalElementLocation**, which is also defined in the Core Model.

- The second concept describes the positioning within the containment hierarchy - for example, Card 3 in Chassis 2 of a specific Rack. This concept is addressed by the Location Within Container property of the **Container** association. The Container association and its specialized associations have been described previously, in the section on Packaging.

- The third concept describes a location that is managed (such as ‘slots’ in a tape library). This concept is a specialization of Physical Package, which was also addressed in the Packaging section.

## 2.5 Replacement Sets

A Replacement Set groups objects that should be removed and/or reinstalled together for the purpose of repair or replacement. The **Participates In Se**t relationship is used to link a particular Physical Element into a Replacement Set.

Subclassing of Replacement Set is from Managed Element versus Collection. This class was defined before Collections were explicitly included in the model. In CIM V2.8, Replacement Set will be moved under Collection, and the Participates In Set association deprecated in lieu of the inherited Member Of Collection relationship.

## 2.6 Physical Capacity

PhysicalCapacity defines minimum and maximum capacities for a variety of physical objects. The Physical Element whose configuration is described is identified using the **CIM_ElementCapacity** association.

Note that capacity planning is not addressed by this class, since a key factor in planning is to understand tradeoffs. Physical Capacity simply reports the minimum and maximum capacities of certain objects (such as disk drives, processors or memory) in a Physical Element. There is no consideration of other entities currently installed, power requirements, etc. For example, suppose a chassis can hold up to five normal-width adapter cards. But, one very wide card is currently installed - taking up two slots. Physical Capacity would indicate that four more adapter cards may be installed – when physical space requirements have reduced that number to three. The combinations and permutations of configuring an enclosure are not described in today's Physical Model.

PhysicalCapacity defines the following subclasses:

- **Memory Capacity** – describing minimum and maximum memory configurations
- **Configuration Capacity** – describing minimums and maximums for a variety of different objects. The ObjectType property identifies the type of entity whose configuration is specified.

The use of the PhysicalCapacity classes is best described by example. Consider a rack holding a "processing" chassis and "storage" chassis (the latter constructed to hold disk drives). The following Capacity instances may be defined:

- An instance of ConfigurationCapacity indicates that the storage chassis can hold "Media Access Devices (Drives)". In this case, the ObjectType property is set to "7", indicating that drives are being described. For discussion purposes, anywhere from 0 to 10 drives can be added to the chassis. The class' properties would be set as follows: Minimum Capacity = 0, Maximum Capacity = 10, Increment = 1. The Element Capacity association is instantiated to define the relationship between this instance of ConfigurationCapacity and the instance of Physical Package representing the storage enclosure.

- An instance of CIM_MemoryCapacity indicates that up to 640Mbytes of DRAM can be placed on the motherboard of the processing chassis. To describe this configuration, the following data would be placed in the class' properties: Memory Type = 2 ("DRAM"), Minimum Configuration = 10Mbytes (for example, required for booting), Maximum Configuration = 640 Mbytes. As above, the Element Capacity association is instantiated to define the relationship between this instance of Memory Capacity and the instance of CIM_Card (representing the Motherboard) in the processing chassis.

## 3 Relationships to Other Standards and Specifications

The major areas of overlap with other standards are with DMI [1] and the IETF's Entity MIB, defined in RFC 2737 [2]. Components in the DMI Master.MIF were taken as a starting point for the Physical Model. Throughout the model, references to the DMI corollaries can be found by searching on the MappingStrings qualifier.

For the Entity MIB, the following table describes that mapping in detail:

| SNMP property name | SNMP property description | DMTF class.property |
|---|---|---|
| entPhysicalIndex | The index for this entry | N/A – An instance is recognized by a specific set of keys |
| entPhysicalDescr | A textual descriptio n of the physical entity. This object should contain a string which identifies the manufacturer's name for the physical entity, and should be set to a distinct value for each version or model of the physical entity. | PhysicalElement.Model |
| entPhysicalVendorType | An indication of the vendor-specific hardware type of the physical entity. Note that this is different from the definition of MIB-II's sysObjectID.<br>An agent should set this object to an enterprise -specific registration identifier value indicating the specific equipment type in detail. The associated instance of entPhysicalClass is used to indicate the general type of hardware device.<br>If no vendor-specific registration identifier exists for this physical entity, or the value is unknown by this agent, then the value{0 0} is returned. | PhysicalElement. VendorEquipmentType |

| SNMP property name | SNMP property description | DMTF class.property |
|---|---|---|
| entPhysicalContainedIn | The value of entPhysicalIndex for the physical entity which 'contains' this physical entity. A value of zero indicates this physical entity is not contained in any other physical entity. Note that the set of 'containment' relationships define a strict hierarchy; that is, recursion is not allowed. In the event a physical entity is contained by more than one physical entity (e.g., double-wide modules), this object should identify the containing entity with the lowest value of entPhysicalIndex. | Container. GroupComponent |
| entPhysicalClass | An indication of the general hardware type of the physical entity. An agent should set this object to the standard enumeration value which most accurately indicates the general class of the physical entity, or the primary class if there is more than one. If no appropriate standard registration identifier exists for this physical entity, then the value 'other(1)' is returned. If the value is unknown by this agent, then the value 'unknown(2)' is returned. The enum values are: other(1), unknown(2), chassis(3), backplane(4), container(5) -- e.g., chassis slot or daughter-card holder, powerSupply(6), fan(7), sensor(8), module(9) -- e.g., plug-in card or daughter-card, port(10), stack(11) | N/A – Determined by the instance's class type. The correspondence is as follows: chassis(3) - Chassis, backplane(4) - Backplane, container(5) – Slot, powerSupply(6) and fan(7) – PhysicalPackage with Realizes relationship to PowerSupply and Fan LogicalDevices, sensor(8) – PhysicalPackage with Realizes relationship to Sensor, module(9) – Card, port(10) – PhysicalConnector with Realizes relationship to LogicalPort, stack(11) – Rack |

| SNMP property name | SNMP property description | DMTF class.property |
|---|---|---|
| entPhysicalParentRelPos | An indication of the relative position of this 'child' component among all its 'sibling' components. Sibling components are defined as entPhysicalEntries which share the same instance values of each of the entPhysicalContainedIn and entPhysicalClass objects. An NMS can use this object to identify the relative ordering for all sibling components of a particular parent (identified by the entPhysicalContainedIn instance in each sibling entry). This value should match any external labeling of the physical component if possible. For example, for a container (e.g., card slot) labeled as 'slot #3', entPhysicalParentRelPos should have the value '3'. Note that the entPhysicalEntry for the module plugged in slot 3 should have an entPhysicalParentRelPos value of '1'. | Container.LocationWithin Container |

| SNMP property name | SNMP property description | DMTF class.property |
|---|---|---|
| entPhysicalName | The textual name of the physical entity. The value of this object should be the name of the component as assigned by the local device and should be suitable for use in commands entered at the device's `console'. This might be a text name, such as `console' or a simple component number (e.g., port or module number), such as `1', depending on the physical component naming syntax of the device.<br>If there is no local name, or this object is otherwise not applicable, then this object contains a zero-length string.<br>Note that the value of entPhysicalName for two physical entities will be the same in the event that the console interface does not distinguish between them, e.g., slot-1 and the card in slot-1. | PhysicalElement. ElementName |
| entPhysicalHardware Rev | The vendor-specific hardware revision string for the physical entity. The preferred value is the hardware revision identifier actually printed on the component itself (if present).<br>Note that if revision information is stored internally in a non-printable (e.g., binary) format, then the agent must convert such information to a printable format, in an implementation-specific manner. If no specific hardware revision string is associated with the physical component, or this information is unknown to the agent, then this object will contain a zero-length string. | PhysicalElement.Version |

| SNMP property name | SNMP property description | DMTF class.property |
|---|---|---|
| entPhysicalFirmware Rev | The vendor-specific firmware revision string for the physical entity. Note that if revision information is stored internally in a non-printable (e.g., binary) format, then the agent must convert such information to a printable format, in an implementation-specific manner. If no specific firmware programs are associated with the physical component, or this information is unknown to the agent, then this object will contain a zero-length string. | N/A – SoftwareIdentity where Classifications property has value 6 (firmware), associated with the PhysicalElement in a Dependency relationship (Allows multiple firmwares to be defined) |
| entPhysicalSoftwareRev | The vendor-specific software revision string for the physical entity. Note that if revision information is stored internally in a non-printable (e.g., binary) format, then the agent must convert such information to a printable format, in an implementation-specific manner. If no specific software programs are associated with the physical component, or this information is unknown to the agent, then this object will contain a zero-length string. | N/A – SoftwareIdentity where Classifications property is set appropriately (`"Unknown", "Other", "Driver", "Configuration Software", "Application Software", "Instrumentation", "Firmware/BIOS", "Diagnostic Software", "Operating System", "Middleware"`), associated with the PhysicalElement in a Dependency relationship (Allows multiple software products to be defined) |
| entPhysicalSerialNum | The vendor-specific serial number string for the physical entity. The preferred value is the serial number string actually printed on the component itself (if present). | PhysicalElement. SerialNumber |

| SNMP property name | SNMP property description | DMTF class.property |
|---|---|---|
| entPhysicalMfgName | The name of the manufacturer of this physical component. The preferred value is the manufacturer name string actually printed on the component itself (if present). Note that comparisons between instances of the entPhysicalModelName, entPhysicalFirmwareRev, entPhysicalSoftwareRev, and the entPhysicalSerialNum objects, are only meaningful amongst entPhysicalEntries with the same value of entPhysicalMfgName. If the manufacturer name string associated with the physical component is unknown to the agent, then this object will contain a zero-length string. | PhysicalElement. Manufacturer |
| entPhysicalModelName | The vendor-specific model name identifier string associated with this physical component. The preferred value is the customer-visible part number, which may be printed on the component itself. If the model name string associated with the physical component is unknown to the agent, then this object will contain a zero-length string. | PhysicalElement. PartNumber |

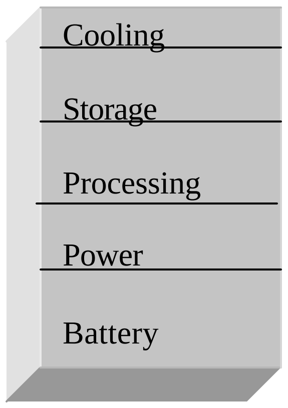
| SNMP property name | SNMP property description | DMTF class.property |
|---|---|---|
| entPhysicalAlias | This object is an 'alias' name for the physical entity as specified by a network manager, and provides a non-volatile 'handle' for the physical entity. On the first instantiation of a physical entity, the value of entPhysicalAlias associated with that entity is set to the zero-length string. However, agent may set the value to a locally unique default value, instead of a zero-length string. | PhysicalElement. OtherIdentifyingInfo |
| entPhysicalAssetID | This object is a user-assigned asset tracking identifier for the physical entity as specified by a network manager, and provides non-volatile storage of this information. On the first instantiation of a physical entity, the value of entPhysicalAssetID associated with that entity is set to the zero-length string. Not every physical component will have a asset tracking identifier, or even need one. Physical entities for which the associated value of the entPhysicalIsFRU object is equal to 'false(2)' (e.g., the repeater ports within a repeater module), do not need their own unique asset tracking identifier. An agent does not have to provide write access for such entities, and may instead return a zero-length string. | PhysicalElement. UserTracking |

| SNMP property name | SNMP property description | DMTF class.property |
|---|---|---|
| entPhysicalIsFRU | This object indicates whether or not this physical entity is considered a 'field replaceable unit' by the vendor. If this object contains the value 'true(1)' then this entPhysicalEntry identifies a field replaceable unit. For all entPhysicalEntries which represent components that are permanently contained within a field replaceable unit, the value 'false(2)' should be returned for this object. | PhysicalElement. CanBeFRUed |

# 4 Physical Model Use Case

## 4.1 Rack example

Pictured is an instance of a Rack that contains 5 Chassis:



Using the Physical Model, this would be described by one instance of CIM_Rack and five instances of CIM_Chassis.

The Location class (defined in the Core Model) can be instantiated to specify the location of the Rack. The Physical Element Location association is used to link the Rack to its Location instance.

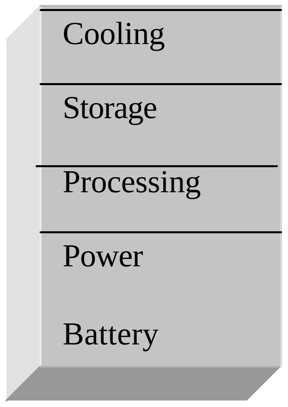
Five instances of the Chasssis In Rack association are defined to link each of the five Chassis to the Rack. The Location Within Container property (inherited by the Chassis In Rack association) is used to specify the location of the Chassis within the Rack.

Now that the containers are identified (i.e., the Racks and Chassis), the physical entities within them can be detailed. This is done be creating the appropriate instances of Connectors, Links, Cards, and Components.

An interesting question is how to recognize the functionality of each chassis? Each chassis represents the functionality of one or more computer systems. The relationship between a chassis and a system is defined by the System Packaging association. In turn, the Dedicated property in Computer System indicates the purpose of each system instance.

## 4.2 Chassis example

Pictured is an instance of a Chassis that contains four Physical Packages (for example, for cooling or disk space) and 1 processing card/motherboard. The chassis is designed as a "tower" personal computer, to be located under a desk.



Using the Physical Model, this example would be described by one instance of CIM_Chassis, four instances of CIM_PhysicalPackage and one instance of CIM_Card. The Chassis Types property of Chassis would be set to "7" to indicate a tower PC.

The Location class (defined in the Core Model) is instantiated to specify the location of the Chassis (perhaps the particular office where the tower is installed). The association, Physical Element Location, links the Chassis to its Location instance.

Four instances of the Package In Chassis (or Container) association link each of the four Physical Packages to the Chassis. The Location Within Container property (inherited by Package In Chassis) specifies the location of the packages within the Chassis.

Another instance of the Package In Chassis (or Container) association is used to link the Card to the Chassis. Again, the Location Within Container property specifies the relative location of the Card, within the Chassis.

Now that the containers are identified (i.e., the Chassis and Packages), the physical entities contained within them can be described. This would be done be creating the appropriate instances of Connectors, Links, Cards, and Components.

As in the Rack example, an interesting question is how to recognize the functionality of the packages in the Chassis? The functionality of the packages and components of the motherboard are described by instantiating the CIM_Realizes association. This relationship binds the physical and logical worlds together – tying a CIM_PhysicalPackage to its appropriate CIM_LogicalDevice counterpart(s). In addition, each instance of LogicalDevice is tied to its "owning" System by the System Device association.

Concluding the example, the “Cooling” Physcial Package would have one or more CIM_Realizes relationships to one or more CIM_Fans. The “Storage” Physical Package would have one or more Realizes relationships to one or more CIM_MediaAccessDevices, and so on.

## 5 Future Work

Two items suggested by the TeleManagement Forum, to better align CIM and their information model (SID), are to:

- Simplify the Connector Type enumeration in Physical Connector, separating the various aspects of the connector (gender, electrical capabilities, number of physical pins and connector layout). This allows a more convenient means of describing connectors.
- Generalize the Storage Media Location class to describe general locations, spaces, shelves, or "slots" where something may be placed or mounted. Unfortunately, the word "slot" is overloaded since it can mean a physical connector (for example, a PCI slot) or a location in an enclosure where something may be placed. In CIM V2.8, a Package Location class will be defined to convey the latter semantics.

## Appendix A – Change History

| | | |
|---|---|---|
| Version 1.0 | <Initial release> | Initial Draft |

## Appendix B – References

[1] DMI Master.MIF - Downloadable from http://www.dmtf.org/standards/standard_dmi.php

[2] IETF RFC 2737 – Downloadable from http://www.ietf.org/rfc/rfc2737.txt?number=2737

## Appendix C – Extending the Model

Major extensions to the Physical Model are not anticipated. The class hierarchy has been in place for several years and is quite stable. However, extensions may be needed to manage new types of hardware or specific vendor information – for example, subclassing CIM_Card to define vendor-specific hardware information.

Take care when extending or subclassing from the Physical Model that the added semantics are truly "physical" – dealing with the things that you can see or touch. Most extensions to CIM occur on the logical side of the models.