


CIM Tutorial > Introduction

# Introduction


Welcome to the Distributed Management Task Force (DMTF) CIM Tutorial. This tutorial will familiarize you with the DMTF, its organization, standards and initiatives.

The tutorial is designed for both beginner and intermediate users. It is designed for management application developers, instrumentation developers, information technology (IT) managers and system administrators To better understand the terminology and concepts used in this tutorial, a basic understanding of object oriented concepts and the Unified Modeling Language (UML) is required.

## Tutorial Requirements

To view this tutorial you will need the following:

- **Netscape Navigator** or **Internet Explorer** version 4.0 or higher
- **800 x 600** or higher screen resolution

## Tutorial Navigation

There are two ways to navigate through this tutorial:

- Index - Select any item in the index to the left to go directly to that page. For example, if you are already familiar with the Common Information Model and are only interested in learning about the CIM Schema, select the Common Information Model item in the index and then click CIM Schema from the navigation tool bar to go through all of the Common Information Models.
- Next/Back - To go through the tutorial in order, just click on the next image in the lower right hand corner.

**Note:** If you click on a link or access the glossary you will need to use the navigation menu on the left to return to the tutorial.

## Downloading the Tutorial

To download and run the tutorial, follow these steps.

- Download the cimtutorial.pdf file. Approximate size: 4 MB.
- Once downloaded, you can run the tutorial locally.




CIM Tutorial > End to End Management


# End to End Management


Are you ready? End-to-End management, across multiple components, in a distributed heterogeneous environment is fast becoming realty and quickly becoming a requirement of management and customers alike. It is no longer sufficient to manage personal computers, subnets, the network core and individual systems in isolation. These components all interoperable to provide connectivity and services. Information passes between these boundaries. Management must pass across these boundaries as well.

The Distributed Management Task Force (DMTF) is leading the development, adoption and unification of management standards and initiatives for desktop, enterprise and internet environments. Working with key technology vendors and affiliated standards groups, the DMTF is enabling a more integrated, cost-effective, and less crisis-driven approach to management through interoperable management solutions.

One such standard, is the Common Information Model (CIM). CIM is a model for describing overall management information in a network / enterprise environment. CIM is comprised of a Specification and a Schema. The Specification defines the details for integration with other management models, while the Schema provides the actual model descriptions.

The CIM Specification is the language and methodology for describing management data. The CIM Schema includes models for Systems, Applications, Networks (LAN) and Devices. The CIM Schema will enable applications from different developers on different platforms to describe management data in a standard format so that it can be shared among a variety of management applications. The xmlCIM Encoding Specification defines XML elements, written in Document Type Definition (DTD), which can be used to represent CIM classes and instances. The CIM Operations over HTTP specification defines a mapping of CIM operations onto HTTP that allows implementations of CIM to interoperate in an open, standardized manner and completes the technologies that support the Web Based Enterprise Management (WBEM) initiative.

Finally, companies implementing the WBEM initiative should be able to realize and recognize these value adds:

1. **Reduced Total Cost of Ownership** by actually managing systems and devices with less man-hours as a result of interoperability
2. **Improved Time to Market** (and possibly gain a competitive advantage) using standards based models
3. **Reduced Development Cost** using and re-using existing standards models, no need to "re-invent the wheel" every time








# Common Information Model


Overview | CIM Schema | Extension Schema | CIM Certification | CIM Query Language

The goal of this section is to introduce the Common Information Model (CIM). If you are familiar with the "basics" of CIM and want more detailed information in one or more of the components of CIM please use the navigation tool bar above to jump to the topic of your choice.

## Introduction to CIM

The Common Information Model (CIM) is conceptual information model for describing computing and business entities in internet, enterprise and service provider environments. It provides a consistent definition and structure of data , using object oriented techniques. The CIM includes expressions for common elements that must be clearly presented to management applications like object classes, properties, methods and associations to name a few. CIM uses a set of terminology specific to the model and the principles of object oriented programming. The standard language used to define elements of CIM is Managed Object Format (MOF).








# CIM Overview


Overview | CIM Schema | Extension Schema | CIM Certification | CIM Query Language

Object Orientated Overview | Specification | Meta Schema | MOF | UML

The CIM is a hierarchical, object oriented architecture that makes it comparatively straightforward to track and depict the often complex interdependencies and associations among different managed objects. Such interdependencies may include those between logical network connections and underlying physical devices, or those of an e-commerce transaction and the web and database servers on which it depends.

The CIM is an information model, a conceptual view of the managed environment, that attempts to unify and extend the existing instrumentation and management standards (SNMP, DMI, CMIP, etc.) using object-oriented constructs and design. The CIM does not require any particular instrumentation or repository format. It is only an information model – unifying the data, using an object-oriented format, made available from any number of sources.

The CIM is comprised of a specification and a schema. The CIM Specification defines the details for integration with other management models, while the CIM Schema provides the actual model descriptions. The CIM Schema captures notions that are applicable to all common areas of management, independent of implementations.

This section will describe the CIM Specification, including the meta schema and the meta schema elements, the Managed Object Format (MOF) and how the Unified Modeling Language (UML) is used to diagram CIM models.

The *CIM Schema* section will describe the schema, including a description of the core and common models.


# Object Orientated Overview

Overview | CIM Schema | Extension Schema | CIM Certification | CIM Query Language

Object Orientated Overview | Specification | Meta Schema | MOF | UML

A prerequisite of understanding and working with CIM is understanding object-oriented modeling. The goal of this section is to deliver a "high-level" overview of the object-oriented model and the various objects of the CIM Schema relate to each other.

CIM is based on an object-oriented model. It is important to recognize that object-oriented *modeling* is different from object-oriented *programming*.

Object-oriented modeling is a formal way of representing something in the real world. It draws from traditional set theory and classification theory. Some basics to keep in mind in object-oriented modeling are that:

- **Instances** are *things*.
- **Properties** are *attributes*.
- **Relationships** are pairs of *attributes*.
- **Classes** are *types* of *things*.
- **Subclasses** are subtypes of *things*.

Note the concept of object-oriented modeling is not limited to computer-related elements. One may use object-oriented modeling to represent many different types of things, from organizational structures, to organic materials, to physical buildings. In the context of CIM and WBEM, object-oriented modeling is used to model hardware and software elements.

For illustrative purposes of this tutorial we are going to use the "Cheeseburger Example" to explain the key concepts of object oriented modeling.

**Abstraction:** DENotes the essential characteristics of an object that distinguish it from all other kinds of objects and thus provide crisply defined conceptual boundaries.

> Example: A Cheesburger - is good to eat and fun to cook.

**Modularity:** Decomposition of abstractions into discrete units.
Example: The various "layers" of a cheesburger - the bun, the lettuce, the ketchup, the mayonnaise, the burger, the cheese, onions, pickels, etc.

**Encapsulation:** Process of compartmentalizing the elements of an abstraction that constitute its structure and behavior; encapsulation serves to separate the interface of an abstraction and its implementation.

> Example:
>
> - To cook the cheeseburger: - Is the stove available? Are the burners working? Are the ingredients available?

- To eat the cheeseburger: - Is it made correctly? Is my plate clean or disgusting?

**Hierarchy:** A ranking or ordering of abstractions.
Example: A cheeseburger is really a subclass of a hamburger with cheese added which is a sub class of sandwich which is a subclass of the Hierarchal superclass food.

**Key Elements: Classes** – A collection of definitions of state, behavior, and/or identity

- Properties
- Methods

**Objects – Instances of a class**
**Associations - Relationships**

- Dependency
- Identity
- Aggregation
- Composition
- And others








# CIM Specification


Overview | CIM Schema | Extension Schema | CIM Certification | CIM Query Language

Object Orientated Overview | Specification | Meta Schema | MOF | UML

The Common Information Model (CIM) is an approach to the management of systems and networks that applies the basic structuring and conceptualization techniques of the object-oriented paradigm. The approach uses a uniform modeling formalism that supports the cooperative development of an object-oriented schema.

The Common Information Model (CIM) specification describes an object-oriented meta model based on the Unified Modeling Language (UML). This model includes expressions for common elements that must be clearly presented to management applications (for example, object classes, properties, methods and associations).

The specification defines the syntax and rules. The specification defines the CIM meta schema, each of the meta schema elements, and the rules for each element. The specification also defines a CIM syntax language based on Interface Definition Language (IDL) called Managed Object Format (MOF). The specification also defines the CIM Naming mechanism.

The CIM Specification does not describe specific CIM implementations, APIs, or communication protocols – those topics are outside the scope of the specification. The CIM Specification also does not include the core and common models. These models are separate from the CIM Specification and are produced independently of the specification.

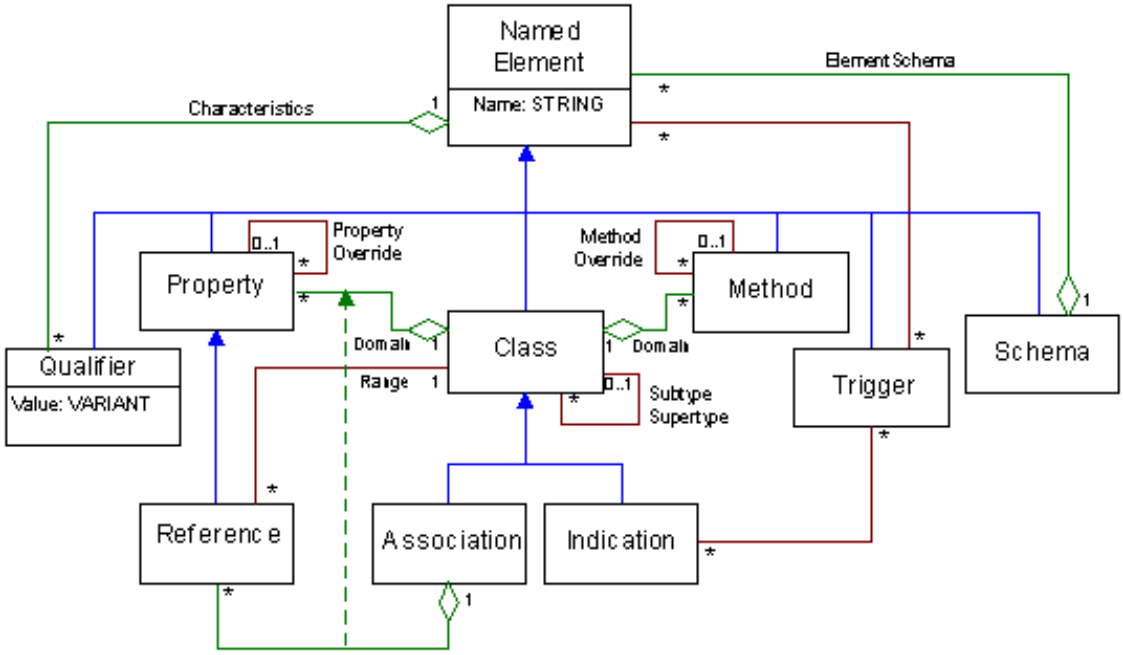

# CIM Meta Schema

Overview | CIM Schema | Extension Schema | CIM Certification | CIM Query Language

Object Orientated Overview | Specification | Meta Schema | MOF | UML

Schema | Class | Property | Method | Qualifier | Reference | Association | Indication

The Meta Schema is a formal definition of the model. It defines the terms used to express the model and its usage and semantics.




The elements of the model are Schemas, Classes, Properties and Methods. The model also supports Indications and Associations as types of Classes and References as types of Properties. The rest of this section will describe each of the elements.







# CIM Meta Schema - Schema


Overview | CIM Schema | Extension Schema | CIM Certification | CIM Query Language

Object Orientated Overview | Specification | Meta Schema | MOF | UML

Schema | Class | Property | Method | Qualifier | Reference | Association | Indication

A *Schema* is a group of classes with a single owner. Schemas are used for administration and class naming. Class names must be unique within their owning schemas. Each class name includes the schema name and follows this format:

*Schemaname_classname*

The rules for schema names are as follows:

1. "CIM" and "PRS" are reserved schema names and MUST NOT be used by any company/organization other than the DMTF.

2. The schema name MUST be unique and MUST begin with an alphabetic character. We recommend the use one of the following methods to assure uniqueness:

   a. Use a trademark registered by your company eliminating any "dot" characters from the trademark, and use the result as your schema name.
   **Examples**: mycompanyname_ or myproductname_

   b. Make use of your company or organization's registered DNS entry:

      - Use the rightmost elements of your company or organization registered DNS address up to and including your company or organization name
      - move any elements to the right of your company or organization name to the left of that name
      - eliminate all "dot" characters
      - use the result as a prefix to your schema name

      **Examples**:
      companyname.com comcompanyname
      company.com.de comdecompany
      pickastate.state.gov stategovpickastate

   3. Use another unique name that you can assure is unique.








# CIM Meta Schema - Class


Overview | CIM Schema | Extension Schema | CIM Certification | CIM Query Language

Object Orientated Overview | Specification | Meta Schema | MOF | UML

Schema | Class | Property | Method | Qualifier | Reference | Association | Indication

A *class* is a blueprint, or prototype, that defines the properties and the methods common to a particular kind of object. Each CIM class is a blueprint for a type of managed element. Classes contain *properties*, which describe the data of the class, and *methods* , which describe the behavior of the class. A class is scoped by the schema in which it belongs. A class must belong to only one schema and the class name must be unique for that schema. A fully qualified class name includes the schema name using the following format: *schemaname_classname*.

## Example:

```
[Abstract, Version ("2.7.0"), Description (
   "ManagedElement is an abstract class that provides a common superclass"
   "(or top of the inheritance tree) for the non-association classes in the CIM Schema.") ]
class CIM_ManagedElement {

   [MaxLen (64), Description (
      "The Caption property is a short textual description (one-"
      "line string) of the object.") ]
   string Caption;

   [Description (
      "The Description property provides a textual description of "
      "the object.") ]
   string Description;

   [Description (
      "  A user-friendly name for the object. This property allows "
      "each instance to define a user-friendly name IN ADDITION TO its "
      "key properties/identity data, and description information. \n"
      "  Note that ManagedSystemElement's Name property is also defined "
      "as a user-friendly name. But, it is often subclassed to be a "
      "Key. It is not reasonable that the same property can convey "
      "both identity and a user friendly name, without inconsistencies. "
      "Where Name exists and is not a Key (such as for instances of "
      "LogicalDevice), the same information MAY be present in both "
      "the Name and ElementName properties.") ]
   string ElementName;
};
```



# CIM Meta Schema - Property


Overview | CIM Schema | Extension Schema | CIM Certification | CIM Query Language

Object Orientated Overview | Specification | Meta Schema | MOF | UML

Schema | Class | Property | Method | Qualifier | Reference | Association | Indication

A *property* is a value used to denote a characteristic of a class. Properties are scoped by the class that owns the property and must be unique within the class. A property has a name, data type, value and an optional default value. A property that does not have a default value is initialized to null.

Property data types are limited to the following intrinsic data types or arrays of such.

| INTRINSIC DATA TYPE | INTERPRETATION |
|---|---|
| uint8 | Unsigned 8-bit integer |
| sint8 | Signed 8-bit integer |
| uint16 | Unsigned 16-bit integer |
| sint16 | Signed 16-bit integer |
| uint32 | Unsigned 32-bit integer |
| sint32 | Signed 32-bit integer |
| uint64 | Unsigned 64-bit integer |
| sint64 | Signed 64-bit integer |
| string | UCS-2 string |
| boolean | Boolean |
| real32 | IEEE 4-byte floating-point |
| real64 | IEEE 8-byte floating-point |
| datetime | A string containing a date-time |
| <classname> ref | Strongly typed reference |
| char16 | 16-bit UCS-2 character |

## Examples

```
[MaxLen (64), Description (
      "The Caption property is a short textual
description (one-"
      "line string) of the object.") ]
string Caption;
```

```
[Write, Description (
      "An enumerated value indicating an
administrator's "
      "default/startup configuration for an element's
Enabled"
      "Status. By default, the element is \"Enabled\"
(value=2)."),
      ValueMap {"2", "3", "5", "6", "7"},
      Values {"Enabled", "Disabled", "Not
Applicable",
      "Enabled but Offline", "No Default"} ]
uint16 EnabledDefault = 2;
```








# CIM Meta Schema - Method


Overview | CIM Schema | Extension Schema | CIM Certification | CIM Query Language

Object Orientated Overview | Specification | Meta Schema | MOF | UML

Schema | Class | Property | Method | Qualifier | Reference | Association | Indication

A *method* is an operation that can be invoked . Methods are scoped by the class that owns the method and must be unique within the class. A class can have zero or more methods.

A method signature includes a name, return type, optional input parameters and optional output parameters.

The method return type must be one of the CIM supported data types. Return types must not be arrays.

A method parameter must be one of the CIM supported data types, fixed or variable length array of one of those types, or an object reference. A parameter can be an input parameter, an output parameter or both. The type of parameter is specified using the IN or OUT qualifier

## Examples

```
[Description (
    "The StartService method places the Service in the started "
    "state. Note that this method's function overlaps with the "
    "RequestedStatus property. RequestedStatus was added to "
    "the model to maintain a record (i.e., a persisted value) "
    "of the last status request. Invoking the StartService "
    "method should set the RequestedStatus property appropriately."
    " The method returns an integer value of 0 if the Service was "
    "successfully started, 1 if the request is not supported and "
    "any other number to indicate an error. In a subclass, the "
    "set of possible return codes could be specified, using a "
    "ValueMap qualifier on the method. The strings to which the "
    "ValueMap contents are 'translated' may also be specified in "
    "the subclass as a Values array qualifier.") ]
```

```
[Description (
    "AddNode brings a new ComputerSystem into a Cluster. "
    "The node to be added is specified as a parameter to the "
    "method. The return value should be 0 if the Computer"
    "System is successfully added, 1 if the method is not "
    "supported and any other number if an error occurred. "
    "In a subclass, the set of possible return codes could be "
    "specified, using a ValueMap qualifier on the method. The "
    "strings to which the ValueMap contents are 'translated' "
    "may also be specified in the subclass as a Values array "
    "qualifier.") ]
uint32 AddNode ([IN] CIM_ComputerSystem ref CS);
```

```
uint32 StartService();
```


# CIM Meta Schema - Qualifier

Overview | CIM Schema | Extension Schema | CIM Certification | CIM Query Language

Object Orientated Overview | Specification | Meta Schema | MOF | UML

Schema | Class | Property | Method | Qualifier | Reference | Association | Indication

*Qualifiers* are values that provide additional information about classes, associations, indications, methods, method parameters, properties or references. The *Qualifier Type* is the definition of a qualifier. A qualifier can not be used without a qualifier type definition and the qualifier must agree with its qualifier type, that is the data type and value must match that of the qualifier type. Qualifiers are scoped by the namespace in which they are present and the qualifier type definition must be unique within that namespace.

All qualifiers have a name, type, value, scope, flavor and an optional default value. The type can be any of the types that are available for a property (except References).

The flavor defines additional behavior for qualifiers. For example, qualifiers can be transmitted automatically from classes to derived classes or restricted to the class for which it was defined. Qualifiers can also be defined to allow whether or not derived classes can override the qualifier value, or whether it must be fixed for an entire class hierarchy. The following table describes the qualifier flavors.

| Parameter | Definition | Default |
|---|---|---|
| EnableOverride | The qualifier can be overridden | yes |
| DisableOverride | The qualifier can not be overridden | no |
| ToSubClass | The qualifier is inherited by any subclass | yes |
| Restricted | The qualifier applies to the class in which it is declared | no |
| Translatable | Indicates the value of the qualifier can be specified in multiple locales | no |

The scope defines the meta elements that the qualifier can be applied to. The scope must contain at least one meta element, but can contain a combination of meta elements or can contain the keyword *any* to imply that the qualifier can be applied to all meta elements. The scope can include the following meta elements: Class, Association, Indication, Property, Reference, Method, Parameter

**Example Qualifier Type Definitions** **Example Qualifier Usage**

```
Qualifier Abstract : boolean = false,
    Scope(class, association, indication),
    Flavor(Restricted);

Qualifier Description : string = null,
    Scope(any),
    Flavor(Translatable);

Qualifier Version : string = null,
    Scope(class, association, indication),
    Flavor(Translatable);
```

```
[Abstract, Version ("2.7.0"), Description (
    "ManagedElement is an abstract class
that provides a common "
    "superclass (or top of the inheritance tree)
for the "
    "non-association classes in the CIM
Schema.") ]
class CIM_ManagedElement {
```








# CIM Meta Schema - Reference


Overview | CIM Schema | Extension Schema | CIM Certification | CIM Query Language

Object Orientated Overview | Specification | Meta Schema | MOF | UML

Schema | Class | Property | Method | Qualifier | Reference | Association | Indication

A *reference* is a special property data type that is declared with the REF key word, indicating that it is a pointer to other instances.

A reference defines the role each object plays in an association. The reference represents the role name of a class in the context of an association. Associations support the provision of multiple relationship instances for a given object. In other words, a system can be related to many system components in many different ways.  A reference may also be used as a method parameter.

### Example:

```
[Association, Version ("2.6.0"), Description (
   "RunningOS indicates the currently executing OperatingSystem. "
   "At most one OperatingSystem can execute at any time on a "
   "ComputerSystem. 'At most one' is specified, since the Computer"
   "System may not be currently booted, or its OperatingSystem "
   "may be unknown.") ]
class CIM_RunningOS : CIM_Dependency {

[Override ("Antecedent"), Max (1), Description (
   "The OperatingSystem currently running on the "
   "ComputerSystem.") ]
CIM_OperatingSystem REF Antecedent;

[Override ("Dependent"), Max (1), Description (
   "The ComputerSystem.") ]
CIM_ComputerSystem REF Dependent;
};
```








# CIM Meta Schema - Association


Overview | CIM Schema | Extension Schema | CIM Certification | CIM Query Language

Object Orientated Overview | Specification | Meta Schema | MOF | UML

Schema | Class | Property | Method | Qualifier | Reference | Association | Indication

An *association* is a type of class that contains two or more references. Associations represent relationships between two or more classes.

Associations are classes that have the association qualifier. Since associations are classes, they establish a relationship between classes without affecting any of the related classes. In other words, the addition of an association has no effect on any of the related classes. An association cannot be a subclass of a non-association class.

In CIM, it is very important to note that associations are *not handled as inverses* with references attached to each class. An association is a separate object with references attached to it.

### Example:

```
    [Association, Aggregation, Version("2.6.0"), Description (
        "The ProductSoftwareFeatures association identifies the "
        "SoftwareFeatures for a particular Product.") ]
class CIM_ProductSoftwareFeatures {

      [Key, Min (1), Max (1), Aggregate, Description (
         "The Product that scopes the SoftwareFeatures.") ]
   CIM_Product REF Product;

      [Key, Weak, Description (
          "The SoftwareFeature in a Product.") ]
   CIM_SoftwareFeature REF Component;
};
```








# CIM Meta Schema - Indiciation


Overview | CIM Schema | Extension Schema | CIM Certification | CIM Query Language

Object Orientated Overview | Specification | Meta Schema | MOF | UML

Schema | Class | Property | Method | Qualifier | Reference | Association | Indication

An *indication* is the active representation of the occurrence of an event. Indications are classes that have the indication qualifier applied. Since indications are types of classes, they can have properties and methods, and can be arranged in a hierarchy. Instances of an indication are transient and can not be obtained by using CIM Operations, such as getInstance() or enumerateInstances(). Indications can only be received by subscribing to them.

### There are two types of indications:

- Life Cycle Indications - CIM class and instance life cycle events
    - Classes - class creation, deletion and modification
    - Instances - instance creation, deletion, modification, method invocation and read access
- Process Indications - alert notifications associated with objects that may or may not be completely modeled in CIM or do not correspond to a simple life cycle event; like low-level instrumentation alerts, DMI alerts, SNMP traps and TMN events

*A Subscription* is expressed by the creation of an IndicationSubscription association instance that references an IndicationFilter (a Filter) instance, and an IndicationHandler (a Handler) instance. A Filter contains the query that selects an Indication class or classes. The size and complexity of the result delivered to the subscriber is dictated by the query.

For more information on indications and the CIM Event Model, see the CIM Event White Paper and UML Diagram.


# CIM Managed Object Format (MOF)

Overview | CIM Schema | Extension Schema | CIM Certification | CIM Query Language

Object Orientated Overview | Specification | Meta Schema | MOF | UML

MOF BNF

There are potentially many ways in which CIM management information could be represented to exchange information. The CIM Specification defines a language based on the Interface Definition Language (IDL) called Managed Object Format (MOF).

The grammar for MOF syntax is described in the notation defined in Augmented BNF for Syntax Specifications. The MOF syntax is a way to describe object definitions in textual form. It establishes the syntax for writing definitions. The main components of a MOF specification are textual descriptions of classes, associations, properties, references, methods and instance declarations and their associated qualifiers. Comments are permitted. A MOF file can be encoded in either Unicode or UTF-8.

The MOF file is basically made up of a series of class and instance declarations.

### Class Example

```
[Version ("2.7.0"), Experimental, Description (
   "A CIM is a type of CIM_WBEMService "
   "that instruments one or more aspects of the
CIM Schema. "
   "A CIM_Provider operates at the request of the
"
   "CIM_ObjectManager to perform operations on
CIM objects. "
   "The properties CreationClasName,
SystemCreationClassName "
   "and SystemName can be set to empty strings.
In this case, "
   "the CIM Object Manager must interpret the
properties with "
   "the local system information.") ]
class CIM_Provider : CIM_WBEMService {

   [Override ("Name"), Description (
      "A human-readable name that uniquely "
      "identifies the provider within a system.") ]
   string Name;

   [Required, Description (
      "An implementation specific string that
identifies the "
      "handle to the provider.") ]
   string Handle;
};
```

### Instance Example

```
instance of CIM_Provider {
   Name = "ACME_OperatingSystemProvider";
   Handle = "ACME_OperatingSystemProvider";
};


instance of CIM_ProviderCapabilities {
   ClassName = "CIM_OperatingSystem";
   ProviderType = { 2 };
   SupportedProperties = NULL;
   SupportedMethods = NULL;
};
```








# CIM MOF BNF "Backus Naur Form"


Overview | CIM Schema | Extension Schema | CIM Certification | CIM Query Language

Object Orientated Overview | Specification | Meta Schema | MOF | UML

MOF BNF

John Backus and Peter Naur introduced for the first time a formal notation to describe the syntax of a given language. Story has it that most of BNF was introduced by Backus in a report, but when Peter Naur read the report he was surprised at some of the differences he found between his and Backus's interpretation. Naur made a few modificiations that are almost universally used and drew up on his own the BNF. Depending on how you attribute presenting it to the world, it was either by Backus in 59 or Naur in 60.

BNF is simply one way of expressing a grammar. It also, turns out to be the most common way of expressing languages in computer science.

The grammar for MOF syntax is described in the notation defined in Augmented BNF for Syntax Specifications, with this deviation: each token may be separated by an arbitrary number of white space characters, except where stated otherwise (at least one tab, carriage return, line feed, form feed or space).

However, while this notation is convenient for describing the MOF syntax clearly, it should be noted that the MOF syntax has been defined to be expressible in an LL(1)-parseable grammar. This has been done to allow low-footprint implementations of MOF compilers.

In addition, note these points:

1. An empty property list is equivalent to "*".
2. All keywords are case-insensitive.
3. The IDENTIFIER type is used for names of classes, properties, qualifiers, methods and namespaces; the rules governing the naming of classes and properties are to be found in section 1 of Appendix F.
4. A string Value may contain quote (") characters, provided that each is immediately preceded by a backslash (\) character.

```
mofSpecification     = *mofProduction


mofProduction        = compilerDirective |
                       classDeclaration |
                       assocDeclaration |
                       indicDeclaration |
                       qualifierDeclaration |
                       instanceDeclaration
```

```
compilerDirective    = PRAGMA pragmaName "(" pragmaParameter ")"

pragmaName           = IDENTIFIER

pragmaParameter      = stringValue

classDeclaration     = [ qualifierList ]
                       CLASS className [ alias ] [ superClass ]
                       "{" *classFeature "}" ";"

assocDeclaration     = "[" ASSOCIATION *( "," qualifier ) "]"
                       CLASS className [ alias ] [ superClass ]
                       "{" *associationFeature "}" ";"

                       // Context:
                       // The remaining qualifier list must not include
                       // the ASSOCIATION qualifier again. If the
                       // association has no super association, then at
                       // least two references must be specified! The
                       // ASSOCIATION qualifier may be omitted in
                       // sub associations.

indicDeclaration     = "[" INDICATION *( "," qualifier ) "]"
                       CLASS className [ alias ] [ superClass ]
                       "{" *classFeature "}" ";"

className            = schemaName "_" IDENTIFIER // NO whitespace !

                       // Context:
                       // Schema name must not include "_" !

alias                = AS aliasIdentifer

aliasIdentifer       = "$" IDENTIFIER // NO whitespace !

superClass           = ":" className


classFeature         = propertyDeclaration | methodDeclaration

associationFeature   = classFeature | referenceDeclaration

qualifierList        = "[" qualifier *( "," qualifier ) "]"

qualifier            = qualifierName [ qualifierParameter ] [ ":" 1*flavor
                       ]

qualifierParameter   = "(" constantValue ")" | arrayInitializer

flavor               = ENABLEOVERRIDE | DISABLEOVERRIDE | RESTRICTED |
```

```
                          TOSUBCLASS | TRANSLATABLE

propertyDeclaration  = [ qualifierList ] dataType propertyName
                       [ array ] [ defaultValue ] ";"

referenceDeclaration = [ qualifierList ] objectRef referenceName
                       [ defaultValue ] ";"

methodDeclaration    = [ qualifierList ] dataType methodName
                       "(" [ parameterList ] ")" ";"

propertyName         = IDENTIFIER

referenceName        = IDENTIFIER

methodName           = IDENTIFIER

dataType             = DT_UINT8 | DT_SINT8 | DT_UINT16 | DT_SINT16 |
                       DT_UINT32 | DT_SINT32 | DT_UINT64 | DT_SINT64 |
                       DT_REAL32 | DT_REAL64 | DT_CHAR16 |
                       DT_STR | DT_BOOL | DT_DATETIME

objectRef            = className REF

parameterList        = parameter *( "," parameter )

parameter            = [ qualifierList ] (dataType|objectRef) parameterName
                       [ array ]
parameterName        = IDENTIFIER

array                = "[" [positiveDecimalValue] "]"

positiveDecimalValue = positiveDecimalDigit *decimalDigit

defaultValue         = "=" initializer

initializer          = ConstantValue | arrayInitializer |
                       referenceInitializer

arrayInitializer     = "{" constantValue*( "," constantValue)"}"

constantValue        = integerValue | realValue | charValue | stringValue |
                       booleanValue | nullValue

integerValue         = binaryValue | octalValue | decimalValue | hexValue

referenceInitializer = objectHandle | aliasIdentifier

objectHandle         = """ [ namespaceHandle ":" ] modelPath """
```

```
namespaceHandle      = *ucs2Character
                       // Note: structure depends on type of namespace

modelPath            = className "." keyValuePairList

keyValuePairList     = keyValuePair *( "," keyValuePair )

keyValuePair         = ( propertyname | referenceName ) "=" initializer

qualifierDeclaration = QUALIFIER qualifierName qualifierType scope
                       [ defaultFlavor ] ";"

qualifierName        = IDENTIFIER

qualifierType        = ":" dataType [ array ] [ defaultValue ]

scope                = "," SCOPE "(" metaElement *( "," metaElement ) ")"

metaElement          = SCHEMA | CLASS | ASSOCIATION | INDICATION |
                       QUALIFIER
                       PROPERTY | REFERENCE | METHOD | PARAMETER | ANY

defaultFlavor        = "," FLAVOR "(" flavor *( "," flavor ) ")"

instanceDeclaration  = [ qualifierList ] INSTANCE OF className [ alias ]
                       "{" 1*valueInitializer "}" ";"

valueInitializer     = [ qualifierList ]
                       ( propertyName | referenceName ) "=" initializer ";"
```

These productions do not allow whitespace between the terms:

```
schemaName           = IDENTIFIER
                       // Context:
                       // Schema name must not include "_" !
fileName             = stringValue

binaryValue          = [ "+" | "-" ] 1*binaryDigit ( "b" | "B" )

binaryDigit          = "0" | "1"

octalValue           = [ "+" | "-" ] "0" 1*octalDigit

octalDigit           = "0" | "1" | "2" | "3" | "4" | "5" | "6" | "7"

decimalValue         = [ "+" | "-" ] ( positiveDecimalDigit *decimalDigit |
                       "0" )

decimalDigit         = "0" | positiveDecimalDigit

positiveDecimalDigit = "1" | "2" | "3" | "4" | "5" | "6" | "7" | "8" | "9"
```

```
hexValue            = [ "+" | "-" ] ( "0x" | "0X" ) 1*hexDigit

hexDigit            = decimalDigit | "a" | "A" | "b" | "B" | "c" | "C" |
                      "d" | "D" | "e" | "E" | "f" | "F"

realValue           = [ "+" | "-" ] *decimalDigit "." 1*decimalDigit
                      [ ( "e" | "E" ) [ "+" | "-" ] 1*decimalDigit ]

charValue           = // any single-quoted Unicode-character, except
                      // single quotes

stringValue         = 1*( """ *ucs2Character """ )

ucs2Character       = // any valid UCS-2-character

booleanValue        = TRUE | FALSE

nullValue           = NULL
```

The remaining productions are case-insensitive keywords:

```
ANY                 = "any"
AS                  = "as"
ASSOCIATION         = "association"
CLASS               = "class"
DISABLEOVERRIDE     = "disableOverride"
DT_BOOL             = "boolean"
DT_CHAR16           = "char16"
DT_DATETIME         = "datetime"
DT_REAL32           = "real32"
DT_REAL64           = "real64"
DT_SINT16           = "sint16"
DT_SINT32           = "sint32"
DT_SINT64           = "sint64"
DT_SINT8            = "sint8"
DT_STR              = "string"
DT_UINT16           = "uint16"
DT_UINT32           = "uint32"
DT_UINT64           = "uint64"
DT_UINT8            = "uint8"
ENABLEOVERRIDE      = "enableoverride"
FALSE               = "false"
FLAVOR              = "flavor"
INDICATION          = "indication"
```

```
INSTANCE            = "instance"
METHOD              = "method"
NULL                = "null"
OF                  = "of"
PARAMETER           = "parameter"
PRAGMA              = "#pragma"
PROPERTY            = "property"
QUALIFIER           = "qualifier"
REF                 = "ref"
REFERENCE           = "reference"
RESTRICTED          = "restricted"
SCHEMA              = "schema"
SCOPE               = "scope"
TOSUBCLASS          = "tosubclass"
TRANSLATABLE        = "translatable"
TRUE                = "true"
```


# CIM Unified Modelling Language (UML)

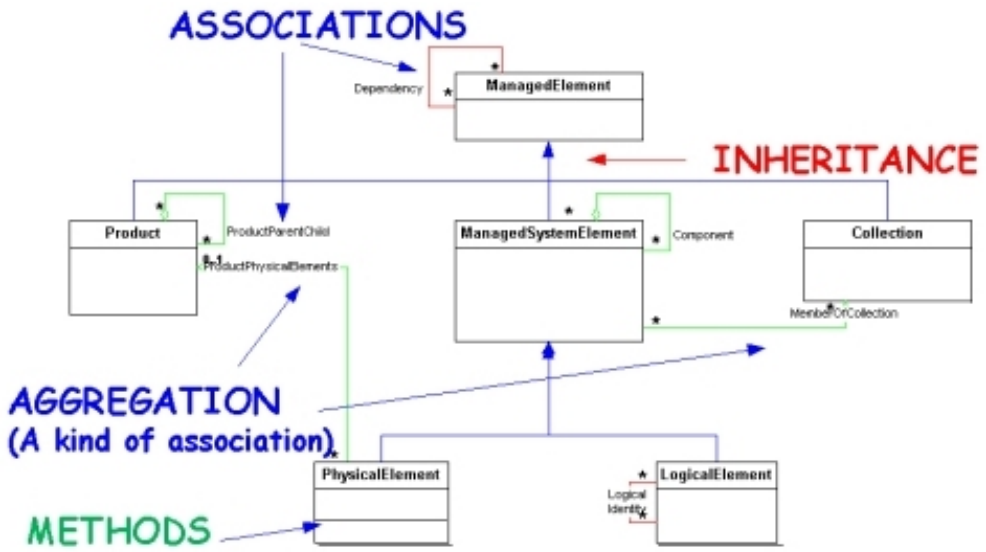
Overview | CIM Schema | Extension Schema | CIM Certification | CIM Query Language

Object Orientated Overview | Specification | Meta Schema | MOF | UML

The DMTF uses the diagramming convention Unified Modeling Language™ (UML) for modeling the CIM core and common models. The Unified Modeling Language™ is a standard specification from the Object Management Group (OMG). UML is the industry-standard language for specifying, visualizing, constructing, and documenting models, including their structure and design.

In UML, a class is represented by a rectangle containing the name of the class. A class with properties is represented by a rectangle divided into two regions, one containing the name of the class and the other a list of properties. Methods are represented by a third region containing the list of methods. Inheritance, or a subclass/superclass relationship, is represented by a line drawn between the subclass and the superclass with an arrow indicating the superclass. Associations are represented by lines with the name of the association usually placed near the center of the line.

CIM model documents generally follow the convention of using **blue** lines for **inheritance**, **red** lines for **associations** and **green** lines for **aggregation**. The color coding makes large diagrams much easier to read but is not a part of the UML standard. There are distinct symbols for all of the major constructs in the schema, with the exception of qualifiers (as opposed to properties, which are directly represented in the diagrams).






CIM Tutorial > CIM > CIM Schema


# CIM Schema


Overview | CIM Schema | Extension Schema | CIM Certification | CIM Query Language

Core Model | Common Models

Management schemas are the building blocks for management platforms and management applications, such as device configuration, performance management, and change management. The CIM is structured in such a way that the managed environment can be seen as a collection of interrelated systems, each of which is composed of a number of discrete elements. The CIM Schema supplies a set of classes with properties and associations that provide a well-understood conceptual framework within which it is possible to organize the available information about the managed environment. The CIM Schema is the combination of the Core and Common Models.

## Core Model

The core model captures notions that are applicable to all areas of management. The core model is a set of classes, associations and properties that provide a basic vocabulary for describing managed systems. The Core model represents a starting point for determining how to extend the common schema.

## Common Models


The Common Models are information models that capture notions that are common to particular management areas, but independent of any particular technology or implementation. Examples of common models include systems, applications, networks and devices. The classes, properties, associations and methods in the common models are intended to provide a view of the area that is detailed enough to use as a basis for program design and, in some cases, implementation.

## Extension Schema

Extension schemas represent technology-specific extensions of the common models. These schemas are specific to environments, such as operating systems. It is expected that the common models will evolve as a result of the promotion of objects and properties defined in the extension schemas.





CIM Tutorial > CIM > CIM Schema > Core Model


# CIM Schema Core Model


Overview | CIM Schema | Extension Schema | CIM Certification | CIM Query Language

Core Model | Common Models

The core model establishes a basic classification of the elements and associations of the managed environment. The class hierarchy begins with the abstract Managed Element class which is in turn subclassed to Managed System Element, the Product related classes, Setting and Configuration, Collection and the Statistical Data classes, among others. From the classes in the Core Model, the model expands in many directions, addressing many problem domains and relationships between managed entities.

## The Core Model is Broken Down into the Following Sections:

- Qualifiers - The meta and standard qualifier type definitions.
- Core Elements - The concepts for the base CIM classes (e.g. ManagedElement, ManagedSystemElement, LogicalElement, System, Service, ...) and base association classes (e.g. Dependency, Component, LogicalIdentity, ...)
- PhysicalElements & Location
- SoftwareIdentity
- Devices - The "logical" function,configuration and state of hardware
- StorageExtents (subclass of LogicalDevice)
- Collections
- Redundancy Information (a subclass of LogicalDevice generalizing the management of storage with more specific classes and services in the Device Model)
- Collections
- Redundancy Information - Both sparing and load balancing
- Product and FRUs
- Statistics
- Capabilities - The superclass to describe the various capabilities of specific ManagedElements.
- Settings and Profiles
- Method Parameters
- Power Management

## Significant Classes and Associations of the Core Model:

The Managed Element class roots the CIM object hierarchy and acts as a reference for associations that apply to all entities in the hierarchy.

Managed System Elements represent Systems, components of Systems, any kinds of services (functionality), software and networks. The definition of "System" in the CIM context is quite broad, ranging from computer systems and dedicated devices, to application systems and network domains.

Both Logical and Physical Elements are subclasses of Managed System Element. Further definition and specification of these subclasses are provided in the Core and Common

Models. For example, System and Logical Device objects are subclasses of Logical Element, defined in the Core Model.

Products represent contracts between vendors and consumers, and capture information about how the Product was acquired, how it is supported, and where it is installed.

Settings define specific, pre-configured parameter data to be "applied" (loosely transitionally) to one or more Managed System Elements. Their definition is very much tied to the properties of existing objects through the Element Setting association. Configurations aggregate Settings and Dependencies, representing a certain behavior or desired functional state for Managed System Elements.

The Statistical Information class is the abstract super class for any kind of statistical data for a Managed Element. The Element to which the Statistical Information applies is indicated via the Statistics association.

Collections represent arbitrary "bags" that group Managed Elements together. Membership can be described by the class definition and/or indicated by explicit instantiation of the Member Of Collection association.

Component associations establish 'part of' relationships between Managed Elements.

Dependency associations describe functional dependencies (one object cannot function without the other) or existence dependencies (the object cannot exist without the other) between Managed Elements.








# CIM Schema Common Models


Overview | CIM Schema | Extension Schema | CIM Certification | CIM Query Language

Core Model | Common Models

The Common Models are information models that capture notions that are common to particular management areas, but independent of a particular technology or implementation. The classes, properties, associations and methods in the Common Models are intended to provide a view of the area that is detailed enough to use as a basis for program design and, in some cases, implementation. Extensions are added below the Common Models in platform-specific additions that supply concrete classes and implementations of the Common Models' classes. As the Common Models are extended, they will offer a broader range of information. The Core and Common models together are referred to as the CIM Schema.

The DMTF is constantly working on new common models that, over time, will help streamline the development process and add to the value statement that is present in CIM and WBEM. The common models for CIM Schema 2.7 are:

| | | | |
|---|---|---|---|
| Applications | Event | Network | Support |
| Database | Interop | Physical | Systems |
| Devices | Metrics | Policy | User |

Each of the common models are discussed in the following pages. It is important that developers when they are working with CIM understand the data they want to manage/provide for as well as an understanding of the various Common Information Models with similar concepts.

Developers are not required to take entire common models, they can use the hierarchies appropriate to their development needs. It is important to note that subclasses have specific semantics (such as ManagedSystemElement) and inherit all properties, methods and associations from classes above. It is also good for developers to understand "rendering" and "infrastructure" too.


# CIM Schema - Application Model

Overview | CIM Schema | Extension Schema | CIM Certification | CIM Query Language

Core Model | Common Models

The CIM Application Management Model describes the information commonly required to deploy and manage software products and applications. This model is based on the need to manage the lifecycle and execution of applications. It can describe applications with structures ranging from standalone desktop applications to a sophisticated, multi-platform distributed, Internet-based application. Both a single software product and a group of interdependent software products that form a business system can be modeled.

The schema today incorporates three major concepts:

1. Structure of an application.
2. Lifecycle of an application.
3. The transition between states in the lifecycle of an application.

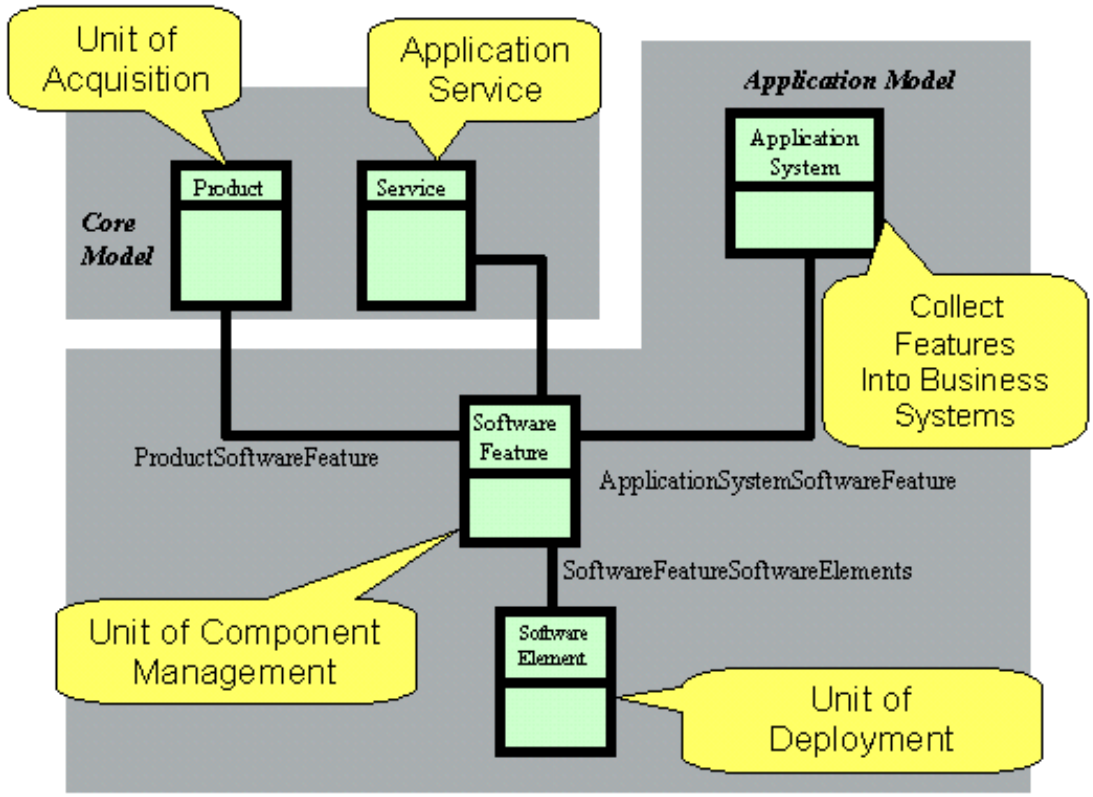
The structure of an application is defined in the following components:

- A ***Software Product*** is a collection of software features that can be acquired as a unit. Acquisition implies an agreement between the consumer and supplier, which may have implications in terms of licensing, support, or warrantee.

- A ***Software Feature*** is a collection of software elements that performs a particular function or role of a software product. This level of granularity is intended to be meaningful to a consumer or user of the application. This concept allows software products or application systems to be decomposed into units that have a meaning to users rather than units that reflect how the product or application was built (i.e., software elements).

- A ***Software Element*** is a collection of one or more files and associated details that are individually deployed and managed on a particular platform. It represents the next level of granularity after software features.

- An ***Application System*** is a collection of software features that can be managed as an independent unit that supports a particular business function.




The most basic aspect of managing applications is managing their transitions through their life cycle. The life cycle can be segmented into four activities:

1. Deployment
2. Installation and configuration
3. Startup
4. Operation including monitoring

The model captures the following states in the lifecycle of an application (note that state information is maintained at the level of the software elements):

- The **deployable state** describes the element in its distribute-able form (for example, in a software repository), as well as the details and operations required to move the element to the installable state (i.e, the next state).

- The **installable state** describes the element as ready for installation (for example, as a zip file that can be decompressed and installed). Also, the details and operations required to move the element to the executable state (i.e., the next state) or back to the deployable state can be defined.

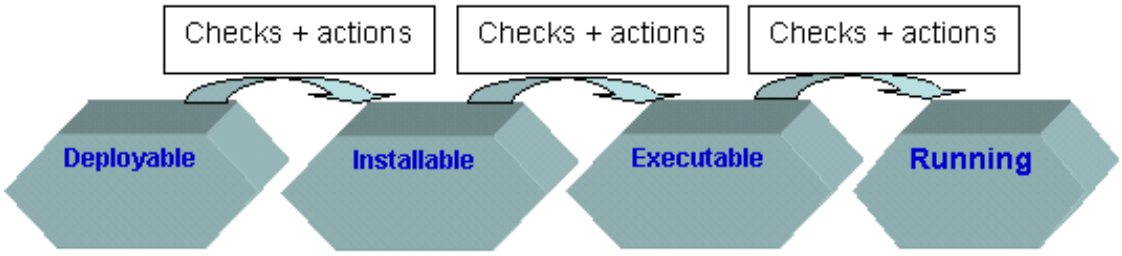
- The **executable state** describes the element as ready to start/run, as well as the details and operations required to move the element to the running state (i.e., the next state) or back to the installable state.

- The **running state** describes the element as it is configured and running.

Managing an application through these states requires an understanding of the conditions that must be true in order to change state (modeled as 'next-state' Conditions), the conditions to verify that an element is in a certain state (modeled as 'in-state' Conditions), and the individual operations to change the state (modeled as Actions). Conditions describe situations in the computer system environment (e.g. file existence or sufficient disk space). Actions are operations that either create a new software element or remove

an existing software element. Actions are organized into two categories: next-state actions and uninstall actions.

This can be visualized as follows:




Taken together, these concepts (and the Application Model that defines them) allow the lifecycle of an application to be fully described, allowing the management of the deployment and operation of the software.

There is additional development, ongoing today within the Application Working Group, to manage the execution of applications. This includes models for describing the structure of complex application runtime environments, and the monitoring and operational aspects of applications. In addition, it is necessary to relate the Unit-of-Work metrics (from the Metrics Model) to the structure of applications, so that software performance can be related to the flow of work through the application.


| | | | |
|---|---|---|---|
| Applications | Event | Network | Support |
| Database | Interop | Physical | Systems |
| Devices | Metrics | Policy | User |

BACK NEXT

# CIM Schema - Database Model

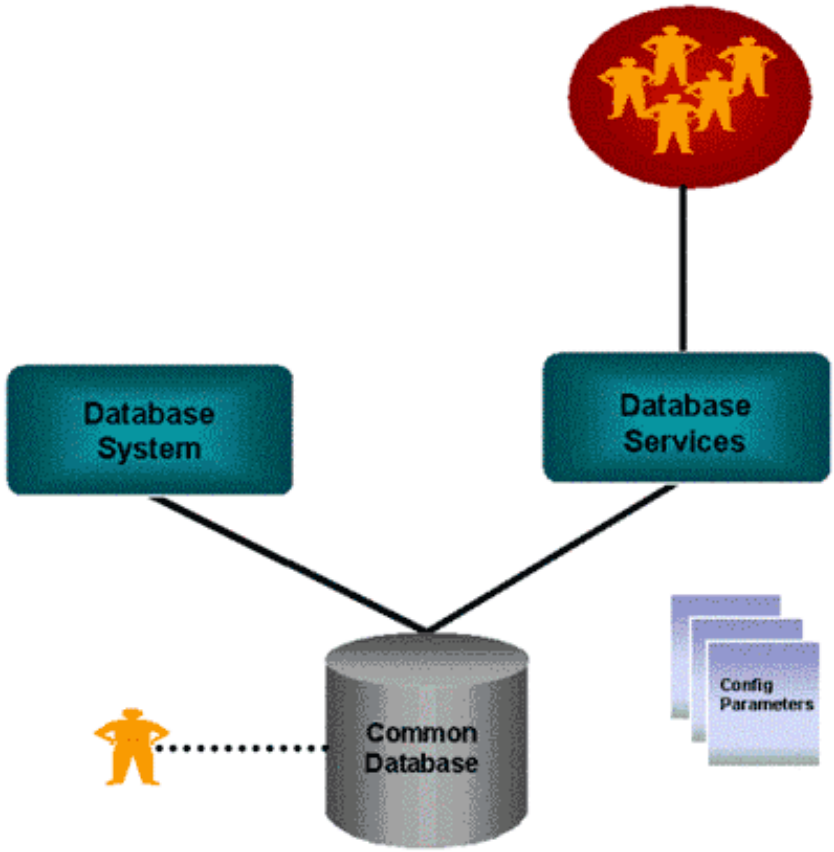
Overview | CIM Schema | Extension Schema | CIM Certification | CIM Query Language

Core Model | Common Models

The CIM database model defines management components for a database environment. In CIM V2.7, the model focuses on the management content that was defined in the IETF's RFC1697 specification. Conceptually, there are three major entities that are modeled.

- The database system, which represents the software application aspects of the database environment
- The common database, which is a logical entity that represents the unit of inter-related, organized data
- The database service, which represents the process or processes that perform tasks for the database, such as providing user access.

In addition, there are a number of supportive classes that represent configuration parameters, resources, and statistics. The following figure shows the conceptual representation of a database environment.



The database system represents the software application aspects of the database environment. Database system software controls the organization, retrieval, storage, security, and maintenance of a database. It includes the software inventory information for the database environment, the software features that are meaningful from a user's perspective, and the software elements that are part of the database software. Background material on the concepts and schema details for application modeling can be found in the Understanding the Application Management Model white paper.

The common database describes the vendor and database organization agnostic properties of a database. It is a logical entity that names a specific, manageable organized body of related information. The working group had a very difficulty time coming up with a definition for a database that was suitable across database domains. The IETF's RFC 1697 specification defines a database as an inter-related unit of data that is organized into a schema. The working group did not come up with a more specific definition that mapped across vendor implementations.

The database service describes the process or set of processes that perform tasks for the database. The RFC 1697 specification calls these database servers. By either name, this class defines the process or processes that co-ordinate user access to the database. Some database services perform other tasks, such as user authentication, authorization, concurrency control, data manipulation, integrity verification and data recovery.


| | | | |
|---|---|---|---|
| Applications | Event | Network | Support |
| Database | Interop | Physical | Systems |
| Devices | Metrics | Policy | User |


# CIM Schema - Devices Model

Overview | CIM Schema | Extension Schema | CIM Certification | CIM Query Language

Core Model | Common Models

The CIM Device Common Models describes the functionality provided by hardware, as well as providing configuration and state data. The model covers a wide breadth of hardware. It addresses low-level concepts such as sensors, batteries and fans, and high-level abstractions such as Storage Volumes.

There are several important concepts related to a CIM_LogicalDevice:

- Devices represent the abstract concepts of the functionality, configuration and state of hardware. They have a "Realized" relationship referencing the hardware that they describe.
- Typically a single hardware component provides multiple functionalities that are realized as multiple different LogicalDevices.
- The configuration of the underlying hardware and software is critical to managing the device.
- The interaction between the various devices (i.e. their interconnections) can play a crucial role in managing the device itself.
- Devices are described as components of a CIM_System that contain them. This relationship is described by the mandatory SystemDevice relationship. It has been identified that this single level of containment makes it difficult to describe a device that is contained by another element, other than a system.

## Disk Drive Example:

The Device Model will not be reviewed in its entirety – because its scope is too large, addressing all the various aspects of hardware functionality, configuration and state. In fact, the Device Model can be broken down to individual components (cooling and power, processors, storage, etc.) that are managed individually. So, to understand the model, a specific, rather common example is chosen – that of a disk drive.


The functionality that we typically associate with a disk drive includes the:

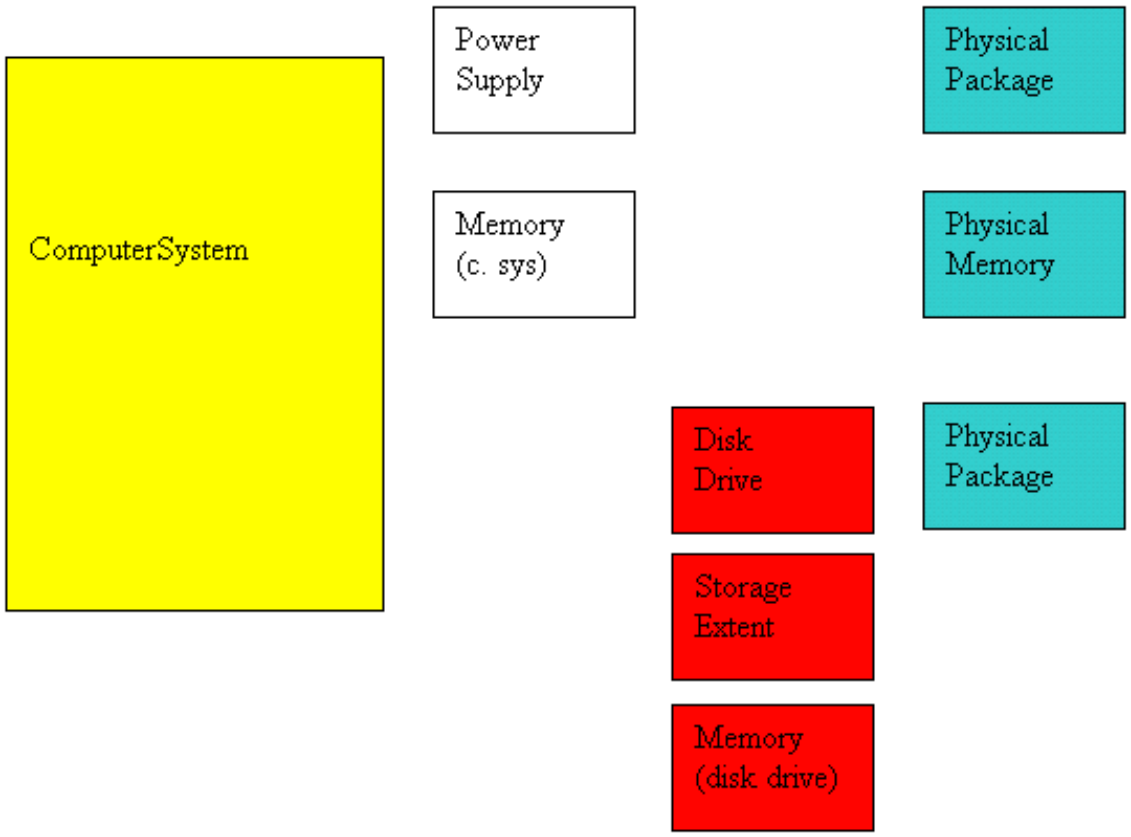
- PhysicalPackage, which represents the drive mechanism that you can see and touch – containing storage, the read/write hardware, on-board flash or EPROMs, etc.
- DiskDrive, which represents the functionality to read/write data from the medium – realized as a type of MediaAccessDevice.
- StorageExtent, which represents the functionality of the medium used to storage the

data – may or may not be removable.
- Memory, which represents the internal cache buffers.
- SoftwareIdentity, which represents the firmware and device driver code that is available for the drive.

Then, there are various associations that tie these classes and concepts together:

- The MediaPresent association is used to tie the StorageExtent to its DiskDrive.
- The AssociatedMemory association is used to tie the Memory to its DiskDrive.
- The DeviceSoftwareIdentity association (defined in CIM V2.8) is used to tie the SoftwareIdentity to its DiskDrive
- The Realizes association is used to tie DiskDrive, StorageExtent, and Memory to the PhysicalPackage.

It is practical to manage a LogicalDevice in the context of the System in which it is functioning. Therefore, the next step in the example is to place the disk drive in the context of a ComputerSystem.




In this example:

The ComputerSystem has a SystemDevice relationship to:

- PowerSupply
- Memory (for the computer system)
- DiskDrive
- StorageExtent
- Memory (for the disk drive)

As noted above, it is cumbersome that the Memory for the disk drive is a component of the ComputerSystem versus a component of the disk drive. However, the Memory is associated to the DiskDrive using the AssociatedMemory relationship. This indicates that

the Memory is indeed "dedicated" to the drive.

The DiskDrive has the following associations:

- SystemDevice to describe its component relationship to the ComputerSystem
- MediaPresent to describe the dependent relationship with StorageExtent (its medium).
- AssociatedMemory to describe its usage of Memory.
- Realizes to tie to the PhysicalPackage (hardware).


| | | | |
|---|---|---|---|
| Applications | Event | Network | Support |
| Database | Interop | Physical | Systems |
| Devices | Metrics | Policy | User |








# CIM Schema - Event Model


Overview | CIM Schema | Extension Schema | CIM Certification | CIM Query Language

Core Model | Common Models

## Overview

The subject of events is complex as it covers a wide range of topics and scenarios. An event is typically assumed to be a change in the state of the environment or a record of the behavior of some component of the environment. For example, the state property for a service may go from Stopped to Started, indicating that the service is now started. Or a device may be added to a machine resulting in a plug and play Indication ultimately notifying the operating system that the device is present and should be configured with settings and drivers in order for it to be usable. An event may be a pervasive incident that occurs infrequently such as a system re-boot or it may reflect very small scale, frequently occurring incidents, for example mouse-clicks. Many things can be affected. The consequences of an event can last a long time. The way that events are dealt with may also vary enormously. Some events may require immediate action on the part of the observer. For example an ‘out of disk space’ event on a web server may require immediate action to make disk space available. Some events may only be of interest at a later time. An example of the interest at a later time is a ‘bandwidth utilization on an interface’ event that a billing application deals with only during nightly billing reconciliation.

## Representation

In the context of this specification, the concrete Indication of the occurrence of an event is represented by an instance of the CIM_Indication class. To improve readability, the redundant use of the 'CIM_' preface will often be omitted in the text when discussing the members of the CIM_Indication class hierarchy.

Types of indications (representing different types of events) are denoted by Indication subclasses. These include:

- InstIndication for modeling CIM life cycle events; instance creation, deletion, modification, method invocation and read access
- ClassIndication for CIM Schema life cycle events; class creation, deletion and modification.
- ProcessIndication for alert notifications associated with objects that may or may not be completely modeled in CIM or do not correspond to a simple life cycle event; like low-level instrumentation alerts, DMI alerts, SNMP traps and TMN events

Instances of indications cannot be enumerated because they are transient objects (not guaranteed to have persistence). Indications are only received after subscribing to them. They cannot be retrieved through enumeration or ordinary query processing in a CIM Object Manager. This was considered to be a necessary design constraint to ensure lightweight Indication processing. It is intended that specialized applications will be devised which will add more sophisticated post processing computations to an Indication

stream on behalf of its clients. Such post processing may include creating unique keys, persistence, aggregation and correlation. Aggregation and correlation will be modeled in a future CIM Indication specification version. Thus indications in this model are not guaranteed to be uniquely identifiable. For example, if the time and date stamp provided is insufficient to distinguish between instances of, say, InstModification, then those instances are not identifiable unless the provider chooses to add an identifier or a logging entity exists which adds this as a post processing function.

### Publication and Subscription

A fundamental idea underlying the CIM approach to the representation of indications is the separation of Indication publication and Indication subscription. The publication of an Indication is accomplished using the same mechanism used for the publication of any other data in CIM; that is, through the declaration of classes and properties. Publication of events also implies the creation of IndicationFilter instances. A Subscription is expressed by the creation of an IndicationSubscription association instance that references an IndicationFilter (a Filter) instance, and an IndicationHandler (a Handler) instance. A Filter contains the query that selects an Indication class or classes. The size and complexity of the result delivered to the subscriber is dictated by the query.

The CIM Object Manager is designed to process queries on behalf of managed object providers. However, it is intended that CIM managed object providers may be designed (although not required) to handle ad hoc Filter queries directly.

Notifications of Filtered events are delivered as instances of the Indication class. A Handler subclass instance is used to specify the destination that is to receive the associated Indication stream. This version of the CIM Indication specification defines the IndicationHandlerCIMXML subclass that is used to deliver indications to clients over HTTP and encoded as cim/XML. Other protocols may be defined in the future to support point to point protocols, multi-cast delivery, email, paging, as well as associated actions like launching a process. Thus the intent in naming this class IndicationHandler (rather than IndicationDelivery) is meant to convey that handling an Indication can require more than delivery.

If there is no provider capable of generating the requested Indication the instantiation of the IndicationSubscription SHOULD fail. Likewise, if there is no instance of the requested IndicationHandler the instantia tion of the IndicationSubscription SHOULD fail. The Modeling Events Section describes the properties and semantics of the CIM_Indication and Subscription class hierarchies.

### Namespace and Subscription Management

Indications and their properties are to be interpreted in the context of a single namespace. The IndicationFilter.SourceNameSpace parameter is used to denote the namespace in which the event that triggered the Indication occurred. This allows creation of all subscriptions in a single CIM namespace even if the events of interest originate from a different namespace. This schema allows creating and examining all Filters in a single namespace regardless of the origin of the events. In addition, since Filter and Handler are subclasses of CIM_ManagedElement, Filters and Handlers can be managed by higher=level services.

### Modeling Events

The event model has to meet two conflicting requirements. First, it has to be extensible allowing schema designers to add new types of indications (events) in arbitrary ways reflecting unforeseen Indication structure and usage. Second, it has to provide a basis for

event analysis and applications that interpret the event flow for aggregation, correlation and throttling purposes without the application having to be aware of the full range of event types implied by the first requirement. As mentioned, while it is intended that this specification support aggregation, correlation and throttling, the exact mechanism for doing this is deferred to a future version. In this proposal, the occurrence of an event is represented as an instance of the Indication class.


| | | | |
|---|---|---|---|
| Applications | Event | Network | Support |
| Database | Interop | Physical | Systems |
| Devices | Metrics | Policy | User |


BACK

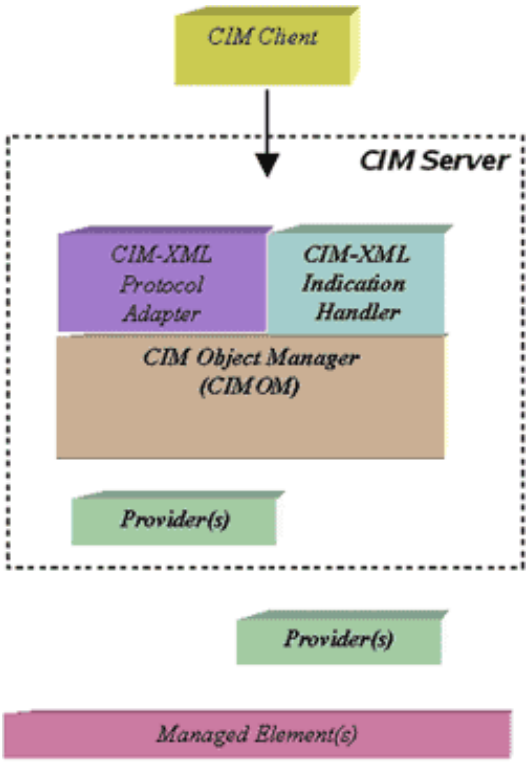
# CIM Schema - Interop Model

NEXT

Overview | CIM Schema | Extension Schema | CIM Certification | CIM Query Language

Core Model | Common Models

The CIM Interop Model defines the management components that describe the WBEM infrastructure and how other WBEM components, such as providers and protocol adapters, interact with the infrastructure. The WBEM infrastructure can be described as follows.




### CIM Client

interacts with a CIM Server by issuing CIM Operation Message Requests and receives and processes CIM Operation Message Responses

### CIM Server

a server that receives and processes CIM Operation Message Requests and issues CIM Operation Message Responses

### CIM Object Manager (CIMOM)

The central component of the CIM Server responsible for the communication between the CIM Server components

### Provider

instruments one or more aspects of the CIM Schema

The CIM Interop Model is broken down into the following sub models. Each sub model describes a particular area of interest.

### CIM Object Manager Model

The CIM Object Manager Model describes the WBEM infrastructure and its relationships. It describes the access mechanisms that the CIM Object Manager supports, the capabilities of the CIM Object Manager and even provides basic statistics data based on CIM Operations.

### Namespace Model

The Namespace Model defines the namespaces that are supported by a CIM Object Manager as well as the type information that is contained in each namespace.

### Provider Model

The Provider Model describes a provider and its capabilities. The capabilities include the class that the provider is supporting as well as the properties and/or methods that are supported. The model also defines the mechanism in which a provider is required to register with the CIM Object Manager.

### Protocol Adapter Model

A protocol adapter is something that accepts information using a particular protocol and converts that information so that it can be used natively, for example CIM-XML or CIM-SOAP. The Protocol Adapter Model describes the protocol adapter information and allows the administrator to query the protocol adapters, configure protocol adapters (e.g. port information), start the protocol adapter and stop the protocol adapter.


| | | | |
|---|---|---|---|
| Applications | Event | Network | Support |
| Database | Interop | Physical | Systems |
| Devices | Metrics | Policy | User |


# CIM Schema - Metrics Model

Overview | CIM Schema | Extension Schema | CIM Certification | CIM Query Language

Core Model | Common Models

The CIM Metrics Model defines the management components that allow the dynamic definition and retrieval of metric information.

The Metrics Model uses a pattern (similar to a "decorator" pattern) based on a metric-definition CIM class, that specifies the semantics and usage of a metric (its meta-data) and another class (a CIM Metric "value" class), containing data values, captured for a particular instance of the metric-definition class.

Originally, this pattern of metric definition / value was defined to manage transaction response time information (e.g., response times at the client, and as a transaction flows through a system), and provide additional metric information about the transaction (for example, identification, processing or resource utilization information). The concept of transaction response time was generalized to the concept of *unit of work*. The unit of work classes measure time for some action to be performed and can attach other metrics to provide additional information.

The goals of the UnitOfWork portion of the Metrics Model are to:

1. Define a model for representing UnitOfWork metric values and their definitions; an instance of a metric should exist only when a definition of its characteristics is present.

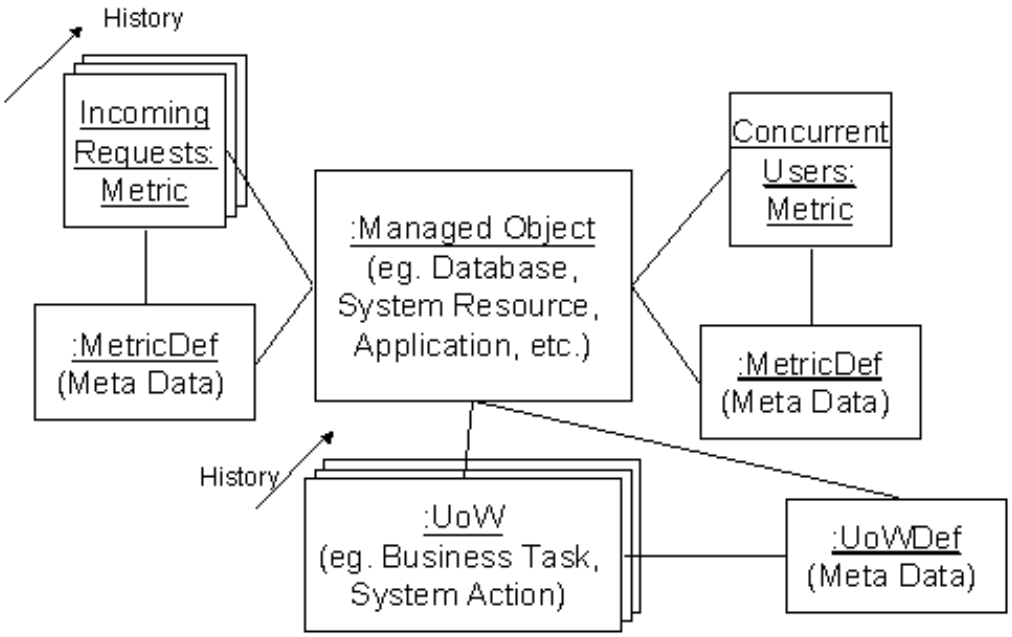
2. Provide a mechanism for dynamically (i.e., at runtime) associating both metrics and their definitions with a LogicalElement.

While it was originally defined for transaction response time measurement, the *unit of work* concept is general enough to address a variety of runtime entities requiring the measurement of time between start and end of an activity (for example, batch processing times). This model matches an instrumentation API for capture of processing time information - the ARM (Application Response Measurement) Specification is defined by the Open Group.

Because the pattern of definition value classes proved to be useful to define dynamic metrics information for unit of work, it was extended to more general metrics (they are named Base Metrics in the model). CIM users often desire metric objects that the models have not yet standardized - for example, time series analysis data on a particular "standard" statistic. Rather than fill more and more CIM Schema with various statistical analysis options, the Metrics Model supports externally defined metrics which add dynamic properties to existing classes.

BaseMetrics provide flexible and dynamically extensible meta-data that is associated with existing ManagedElements. Again, there is a definition class (CIM_BaseMetricDefinition)

and a values class (CIM_BaseMetricValue). An instance of a CIM_BaseMetricDefintion defines the semantics, type, and usage of a metric (e.g., a data type for the metric); instances of the CIM_BaseMetricValue class capture values for a particular definition instance.




One of the core assets of the CIM 2.7 Metrics Model is its capability of introducing late-binding for arbitrary, user/administrator-defined metrics. More specifically, a user is able to introduce new Metric Definitions at runtime into CIM and then instantiate one or more Metric Values that follow the semantics of these Metric Definitions - all as instances of existing classes, without needing to define new classes.

Work continues in the Metrics Sub-Team of the Application Working Group on 1) the development of usage scenarios for metrics, 2) the development of further components of the model to work with metric information (for example, aggregation, time series, and correlation classes), and 3) connection of the Unit-Of-Work metrics to the general runtime applications model that is being developed to allow direct relationships between the modeling of performance and status of application systems and the flow of traffic through those systems.


| | | | |
|---|---|---|---|
| Applications | Event | Network | Support |
| Database | Interop | Physical | Systems |
| Devices | Metrics | Policy | User |



# CIM Schema - Network Model

Overview | CIM Schema | Extension Schema | CIM Certification | CIM Query Language

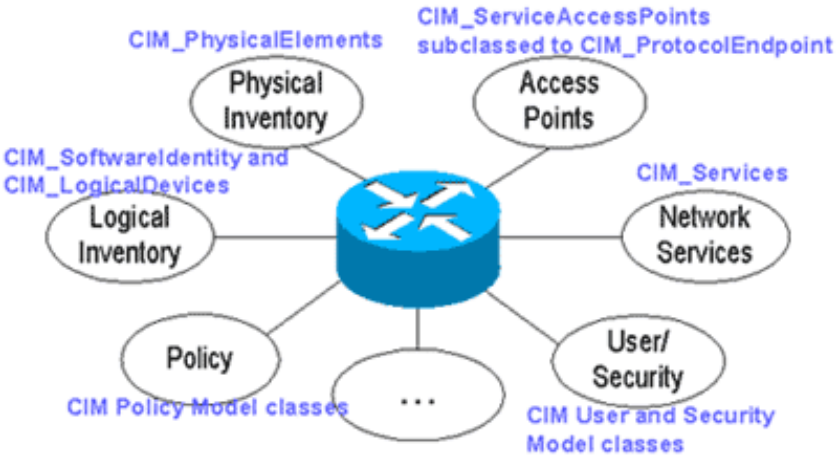
Core Model | Common Models

The Network Model describes and manages communications connectivity and the network "cloud", as well as the individual services and protocols in the network. The managed entities in the model may be grouped into broad categories describing:

1. Network systems (with associated inventory, user and security information, etc.)
2. Network services (for example, routing)
3. Logical interconnection and access (e.g., protocol endpoints, routes and network pipes) – Applicable to both the network and client systems
4. Network protocols (such as OSPF and BGP)
5. Networking technologies (e.g., Switching/Bridging and VLANs)
6. Quality of Service (QoS) technologies (such as meters, markers and queues)
7. Other supporting definitions (for example, various network packet filtering criteria)

The information model characterizes a network as a type of administrative domain, which in itself may contain other networks, sub-networks or domains. These may be defined according to criteria such as a business' management policy domains or geography. In order to operate the network infrastructure, networking services are required and are thus defined in the model. Given the wide use of the word "service", it is important to point out that within the context of the current information model, "service" refers to the functionality provided by infrastructure services, or required by the individual network elements to operate and exchange information. Examples of these services are Routing (for example, OSPF and BGP), Forwarding, and quality of service (QoS).

Within the administrative domains/networks, there are network elements (also known as network systems or network devices). These sit at the core of the network or at its edges. There is much confusion over how to model a network system, since many companies believe that these are different than CIM_ComputerSystems. However, after detailed analysis, it was found that network systems fit the same pattern (i.e., have the same basic properties, methods and associations) as Computer Systems. This can be seen in the figure below, which depicts the various aspects of a router.




Admittedly, network elements are dedicated systems – with hardware and software tuned for network performance and functionality. (Note that Dedicated is a property defined on the ComputerSystem class!) The distinction is that the network element's hardware and software are the components OF the system, not the system itself. Therefore, a network system is modeled as an instance of ComputerSystem with associations to the LogicalDevices, Services and ServiceAccessPoints that are hosted on it, or components of it.

Services are made available or accessed throughout a network via ProtocolEndpoints (a subclass of Service Access Point). Endpoints describe and manage the protocol-specific configuration, state and addressing information that is needed to transmit and receive messages on a network. Protocol Endpoint's properties provide details on total bandwidth, available bandwidth, keepalive timers, retry intervals, etc.

Two Protocol Endpoints may be associated within a system or across a network. Usually, the former describes the protocol stack on a platform – for example, a TCP port running over an IP address on an Ethernet network. This is modeled via a many-to-many relationship, BindsTo. The reason that the association is many-to-many is because it is possible to combine the communication capabilities of lower level ports into a single higher level one, or take a large bandwidth interface and run several higher level interfaces over it. This requirement to describe fan-in and fan-out bindings guided the design of the BindsTo association.

As regards connecting across a network (or even within a system), the ActiveConnection association represents this semantic (i.e., the exchange of information between two Protocol Endpoints). Typically, this association occurs between endpoints at the same protocol level within a communication stack or application. ActiveConnection is used when the potential for communication should be represented, but the connection itself is not managed. That is, the connection exists but it does not have a state nor configuration information associated with it. In situations where there is a need for a managed connection, independent of the managed endpoints, the Network Pipe class is used instead. Network Pipe is a subclass of Enabled Logical Element, and is not an association. It does have its own associations to the Protocol Endpoints that are the ends of the pipe.

All of these logical entities must be associated to the PhysicalPackages and components that are being managed. Although this is not specified as part of the networks model, it is an integral part of the overall model needed to manage the network. The reader is encouraged to consult the Core, Physical and Device Common Models for further details.

In addition to general aspects of the Network Model, specific technology and protocol areas are also addressed. Two commonly used routing protocols are defined in the model,

OSPF and BGP. In both cases, the respective sub-models focus on the configuration of the routing protocol. The protocols are characterized in terms of the services that they offer, the endpoints through which these services are made available, and protocol specific configuration parameters.

In the switching arena, the model covers Spanning Tree Protocol (STP), bridging functions, VLANs and at a slightly higher protocol layer, MPLS. The VLAN model represents a VLAN as a logical network (basically, a collection of protocol endpoints for connectivity), composed of switch and user station endpoints. The VLAN to STP relationship is defined as part of the Switching and Bridging sub-model. The MPLS sub-model focuses on the configuration aspects of Label Switched Paths (LSPs) and Traffic Engineering (TE) Tunnels, which may ride on defined LSPs.

In conclusion, the Network Model broadly describes and manages general connectivity between systems, as well as network technology and protocol specifics. It covers not only the configuration and state aspects of management, but also defines statistics that may be collected from the network elements in support of performance management applications.


| | | | |
|---|---|---|---|
| Applications | Event | Network | Support |
| Database | Interop | Physical | Systems |
| Devices | Metrics | Policy | User |


# CIM Schema - Physical Model

Overview | CIM Schema | Extension Schema | CIM Certification | CIM Query Language

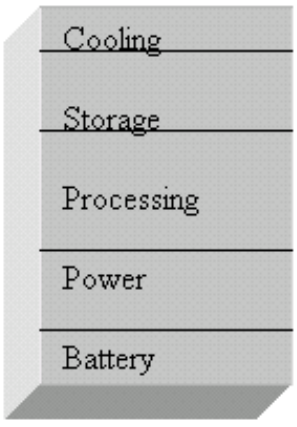
Core Model | Common Models

The CIM Physical Common Models 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.

It is important to remember that the abstractions in the Physical Model typically represent the physical make up of a ComputerSystem. 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 services hosted on the ComputerSystem. For example, there is almost no PHYSICAL difference between 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.

**Example:**

Pictured is an instance of a Rack that contains 5 Chassis. Using the Physical Model:




This would be described by 1 instance of CIM_Rack and 5 instances of CIM_Chassis.

The Location class (defined in the Core Model) can be instantiated to specify the location of the Rack.

The PhysicalElementLocation association is used to link the Rack and Location instances.

The ChasssisInRack association is used to link the five Chassis within the Rack.

The LocationWithinContainer property (in the ChassisInRack association) is used to specify the location of the Chassis within the Rack.

Now that we know about the containers (i.e., the Racks and Chassis), the physical entities contained within them can be described. A Physical Element is defined as one of four

subclasses/categorizes. They are:

- Physical Package – This class describes general containers and frames, and provides management, maintenance, and repair information.Instances of Physical Packages contain other Physical Elements. This is expressed using the Container association. Physical Package is further refined by Frame, Chassis, Rack, Card, and StorageMediaLocation subclasses. The concept of a Card includes motherboards, backplanes, adapter cards, daughter cards, etc. The concept of a StorageMediaLocation defines the shelf/hole/slot where a storage magazine or tape can be stored.
- Physical Component – Physical Component describes low-level hardware, such as chips and physical media. The Container association is used to describe a Component mounted on or in a Physical Package. Physical Component is further refined by the subclasses, Physical Media (for example, a tape) and Chip (i.e., a processor or memory chip).
- 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 Container association is used to describe a Physical Component that is mounted on a Physical Package. The PackageInConnector association is used to describe a Physical Package that is inserted into the Physical Connector.
- Physical Link – This class describes the cabling used between Physical Elements, such as connectors. The Elements Linked association indicates the Physical Elements that are connected. The Link Has Connector association describes the Physical Connector on the cable.

There are two additional concepts described by the Physical Common Models – they are Replacement Sets and Physical Capacity.

- Replacement Sets group objects that should be removed and reinstalled together for the purpose of repair or replacement.

- Physical Capacity describes the capability/capacity of related hardware to contain and/or connect to a minimum number and a maximum number of objects. The hardware being described is indicated using the ElementCapacity 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 1.


| | | | |
|---|---|---|---|
| Applications | Event | Network | Support |
| Database | Interop | Physical | Systems |
| Devices | Metrics | Policy | User |


# CIM Schema - Policy Model

Overview | CIM Schema | Extension Schema | CIM Certification | CIM Query Language

Core Model | Common Models

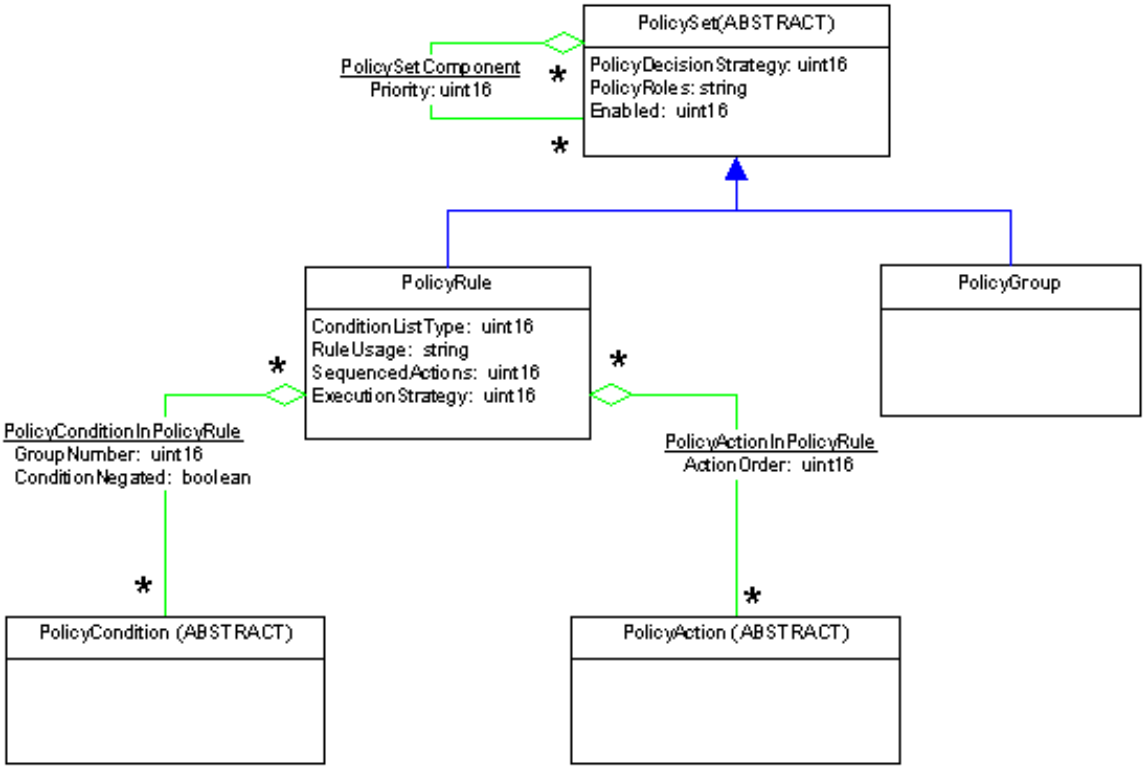
When used in the context of computer systems, "policy" is a term that is frequently used to describe any system configuration that controls system behaviors such as in "security policies" or "quality of service policies." In general, a "policy" is a "definite goal, course or method of action to guide and determine present and future decisions." [RFC3198] Policies can specify resource management directives and, at a higher level of abstraction such as in a service-level objective, policies can also specify user experience management directives. The DMTF Policy Model provides a common framework for specifying system behaviors that are both sufficiently abstract to be independent of implementation-specific details and scalable to configuring large complexes of computer systems, i.e., the DMTF Policy Model is a specific model for expressing such policies in a general and scalable way.

Developed jointly [1] by the IETF and the DMTF, the Policy Model is an object-oriented model that enables constructing policy rules of the form:
if <condition(s)> then <action(s)>.
The <condition(s)> term is a Boolean expression used to specify the rule selection criteria. These criteria may include temporal conditions (when does the rule apply), scoping conditions (to what does the rule apply) and state-related conditions (under what circumstances should the action(s) of the rule be attempted). When, for a resource under management, the <condition(s)> expression evaluates to True, the <action(s)> portion of the rule is attempted. These conditional directives are an implementation-independent description for the common aspects of system administration. However, the Policy Model does not, by itself, provide the abstracted specification. For a discipline, e.g., IPsec-based VPN, network quality of service, etc., a set of subclasses are defined to capture the necessary semantics to express rules in that discipline.

The scalability is provided, in part, by the abstractions that also enable implementation-independence and, in part, by a role-based deployment framework. A *policy role* is an administratively assigned name for the enforcement role played by an entity that consumes sets of policy rules. The sets of rules are flagged with one or more policy roles for which they apply and the enforcement points know the roles that they play. The deployment infrastructure, then, delivers the appropriate policy rules to the enforcement points based on these named relationships. For example, rules marked with the policy role “DMZ Web Server” would be delivered to those web servers that are located in the DMZ of the administrative domain. Similarly, rules that are intended for use with Ethernet interfaces facing the Internet in the DMZ might be flagged with the policy role "DMZ External Ethernet" and the security gateways playing that role, i.e., in the DMZ and with externally facing Ethernet interfaces, would be recipients of these rules.

The CIM PolicyRule class is the central class used to aggregate PolicyCondition and PolicyAction object instances. PolicyRule instances may be aggregated into instances of PolicyGroup instances to form coherent sets with the same Policy Role value(s). Actually,

a PolicySet is a common abstraction for PolicyRule and PolicyGroup because both classes can aggregate PolicySet (PolicyRule and PolicyGroup) instances.




A PolicySet, then, forms a coherent set with a common set of PolicyRole values and a decision strategy for selecting, based upon priority, which of the rules or sets of rules in the aggregation are to be evaluated. The PolicyRoleCollection class (not shown here) is used to identify the roles played by enforcement points that need to consume the PolicySets identified by PolicyRole. The PolicyRole, and therefore the elements collected in the PolicyRoleCollection, identifies the resource(s) to be managed using the PolicySet. So, for example, if the PolicyRole is "Ethernet Interface" the enforcement points that consume these PolicySets are the enforcement points that manage Ethernet interfaces and it is the Ethernet interfaces that are collected into the PolicyRoleCollection instance.

For CIM 2.7, there is only one decision strategy defined, FirstMatching, for which the aggregated PolicySet instances are evaluated by priority and the actions of only the highest priority PolicySet that matches (conditions evaluate to True) are attempted. Clearly, when the aggregated policy set is a PolicyGroup instance, it doesn't have conditions that can be evaluated, but it is selected in turn by priority and then, recursively, its decision strategy and priorities are applied.

For further details, the Policy White Paper, UML diagram and MOF should be consulted.

1. The IETF Policy Core Information Model [RFC3060] and Policy Core Information Model Extensions [RFC3460] is a superset of the CIM 2.7 Policy Model. There are a few functions included in the IETF specifications that are not included in the DMTF model pending further analysis.


| | | | |
|---|---|---|---|
| Applications | Event | Network | Support |
| Database | Interop | Physical | Systems |
| Devices | Metrics | Policy | User |


# CIM Schema - Support Model

Overview | CIM Schema | Extension Schema | CIM Certification | CIM Query Language

Core Model | Common Models

In the past decade, there has been exponential growth in both the complexity and interdependence of products in the computing industry. This is due to rapid advances and growth of technology as well as the increased openness between products. A fundamental principle of system design in the development of modular, "plug-and-play" approaches, such as client/server in the computing industry, is to allow diverse products to work together. Customers increasingly require and expect products and the companies supporting them to work together to provide a total solution to their needs. These trends have created a demand in many industries for support providers to access support information about related products.

In response to this demand, many support providers have attempted to publish their information to each other on a more intensive basis, and to engage in partnerships that allow support analysts to collaborate on multi-vendor issues. However, without any standardized way to represent and communicate information, the process of gathering, publishing and interpreting the immense variety of support information remains costly, inconsistent and largely ineffective. A solution exchange standard has broad applicability in the customer support domain, and has the potential to promote richer communication and collaboration between two or more support partners, both in solving specific problems and in evolving a more effective overall relationship.

Several factors are driving the support industry to adopt a standard method for exchanging Service Incidents. These factors stem from the multi-vendor nature of the computer and software industry. Many different companies create the computer components, peripherals, operating systems, and application software that comprise the personal computer market. Successfully providing customer support in today's multi-vendor environment requires high levels of cooperation between support organizations. Providing such support has created efficiency and cost challenges to support providers and product vendors. Standardized incident exchange provides a mechanism for support organizations to share incident information effectively and ultimately reduce the cost of supporting computers, thus reducing the total cost of ownership to customers.

The Problem Resolution Standard (PRS) and associated MOF and UML is the merger of two prior standards from the Distributed Management Task Force (DMTF) and the Consortium for Service Innovation (CSI) known as the Solution Exchange Standard (SES) and Service Incident Exchange Standard (SIS). The primary purpose of PRS is to define an open exchange standard that facilitates Solution exchange and Service Incident processing between cooperating parties, both within an organization and across organizational boundaries.

All classes defined by this specification begin with the letters "PRS_" as opposed to the use of "CIM_" for other CIM-related object models. This prefix is due to historical reasons and all classes within PRS are to be treated as standard CIM extensions.

Before Solutions can be exchanged, they must be encoded so that they are consistently created by any compatible Producer and understood by any compatible Consumer. The exchange of Solutions is more than simply exchanging data (bits and bytes) and it is more than the exchange of random information (properties and classes). The exchange of Solutions requires an understanding between the parties of the exchange on the type of information being exchanged and the relationships (associations) between exchanged information to convey complete understanding or knowledge.

Consumers and Producers vary widely on the complexity and detail of the knowledge they process, so exchange must support this variety. Exchange participants "mine" the knowledge to the depth of their ability. Any exchange standard must also address the need to support extensibility so Consumers and Producers may extend the object model for their own unique needs.

It is important to note that this standard is focused on Solution exchange and not Solution storage. It is intended to facilitate the exchange of Solution knowledge without favoring any method of Solution storage.

Service Incident processing builds on the Solution exchange object model. All of the objects defined for Solution exchange may be used in Service Incident processing. Service Incident processing adds five new classes, a transaction model and some new associations to the Solution object model.


| | | | |
|---|---|---|---|
| Applications | Event | Network | Support |
| Database | Interop | Physical | Systems |
| Devices | Metrics | Policy | User |


# CIM Schema - Systems Model

Overview | CIM Schema | Extension Schema | CIM Certification | CIM Query Language

Core Model | Common Models

The CIM System Common Models defines computer-system related abstractions. Many of the concepts related to computer system derive from the CIM_System abstraction in the Core Model. CIM_System describes the aggregation of 'parts' (or components) into a single manageable 'whole' (the system).

### Important concepts related to a CIM_System are:

- Systems act as aggregation entities.
- Systems are not modeled as a collection. A system is more than the sum of its parts. Systems have status and they host services and access points.
- Systems are top-level objects that are frequently used to scope their aggregated entities.

Besides the concept of the computer system itself, the System Model also addresses compute components and functionality, associated with most computer systems. These include concepts such as file systems and files, operating systems, jobs, processes and threads, and diagnostics. In addition, both general purpose and 'dedicated' systems can be described. There are no specific subclasses to describe system functionality (i.e. routing, storage, storage array, and etc). The functionality of these systems is described by the services that are hosted or are capable of being hosted. Otherwise, you end up with individual subclasses routing, storage, etc. It is conceivable that a single system could provide all of these. Therefore, the dedicated property should be used to describe the primary functionalities that the system supports.

### Server Example:

Monitor

Storage SubSystem

When examining a typical server, we are likely to find the following logical elements:

- Several installed operating systems, one running operating system which has jobs and processes
- Local and remote file System that are composed of directories and files.

- Logical devices such as a Monitor, Keyboard, Mouse, Hard Disk, Processor, Power Supply, Fan, and etc.
- Services that are hosted on the server itself, such as spell checker or a diagnostic service.
- Services that are available to the server via service access points, such as a print service.

In addition, the server has physical aspects such as:

- The server is contained in chassis or multiple chassis which may be mounted in a rack
- The server consists of cards and components (chips)
- The server occupies space at a known location

In the CIM environment the server is modularized into its elements. These elements are then associated to the computer system.

- The elements denoted in blue are defined in the System Common Models.
- The high level concepts of the elements denoted in white are defined in the Core Model.
- The specific subclasses of PhyscialElements (denoted in green) are defined in the Physical Common Models.
- The specific subclasses of Logical Device (not shown) are defined in the Device Common Models.
- The asset aspects of Software are defined as SoftwareIdentity in the Core Model. While the deployment aspects of applications are defined in the Application Common Models.


| | | | |
|---|---|---|---|
| Applications | Event | Network | Support |
| Database | Interop | Physical | Systems |
| Devices | Metrics | Policy | User |


# CIM Schema - User Model

Overview | CIM Schema | Extension Schema | CIM Certification | CIM Query Language

Core Model | Common Models

The focuses of the CIM User/Security Common Model are twofold, defining classes to manage:

- General contact and white pages information for organizations, organizational units and people
- "Users" of services, and the related security information to authenticate and authorize those "users"

The contact and white pages information is contained in the class hierarchy under CIM_OrganizationalEntity. These classes capture organizational data (such as addresses and phone numbers) and relationships (using the OrgStructure association).

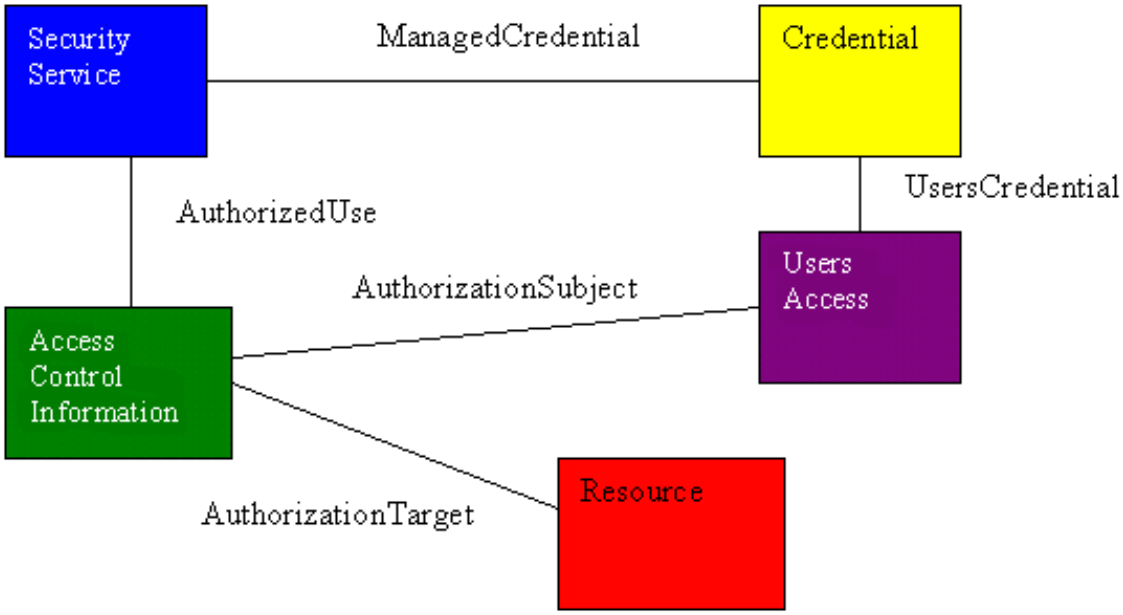
Regarding "users", users may be people, or they may be non-human entities - such as a service running as part of an application system - and they may be collections thereof. The User and Security Model factors the user into several classes. There are managed elements that have a user relationship to a system or set of systems (conveyed using the CIM_ElementAsUser association), and two classes that represent the users' access to system resources: CIM_UsersAccess and CIM_Account.

CIM_UsersAccess is the nexus of a user's system access information, such as credentials and system accounts, independent of the associated element that has access. That is, a managed element such as a Person instance might have several user accesses: for example, one could be for an administrative set of authorities in an administrative domain, and another for access for other general business processes (such as routine access of mail). The CIM_UsersAccess class instances, then, provide a user's view of their relationship to the systems with which they interact. The CIM_ElementAsUser association is used to convey the "ownership" relationship between the managed element that has access and the CIM_UsersAccess instances.

CIM_Account, on the other hand, can be used as the nexus of a system's information *about* a user. The CIM_UsersAccount association provides the relationship back to the user (for traversals for information such as a person's name or the credentials that may be used for access to the account, etc.). A system instance (e.g., CIM_ComputerSystem, CIM_AdminDomain, CIM_ApplicationSystem) provides namespace scoping via the weak aggregation of accounts. Instances of CIM_Account are defined within the scope of their aggregating system. The management of these account instances, however, need not be from a service on that system. CIM_AccountManagementService instances may have CIM_ManagesAccountOnSystem relationships for accounts on any system and, therefore, CIM_ManagesAccount relationships as well. For example, this might occur when the accounts are on an administrative domain and the account management service instances are hosted on a subset of the computers in that administrative domain.

Although not complete in this release of the Model, several classes are defined to provide operational implementation of some security policies. (This is distinct from the specification of a device-independent security policy, or the resulting device-specific configuration of those policies). The CIM_AuthenticationRequirement class permits the specification of the credentials, required for authentication, for access to specific target resources. On the other hand, CIM_AccessControlInformation permits the specification of authorization policies that match users (subjects) and resources (targets) with a set of permissions (access type, access qualifier, and permission).

The concepts and relationships of Credentials, Users Access and Access Control Information are shown in the figure below.





| | | | |
|---|---|---|---|
| Applications | Event | Network | Support |
| Database | Interop | Physical | Systems |
| Devices | Metrics | Policy | User |

# Extension Schema

Overview | CIM Schema | Extension Schema | CIM Certification | CIM Query Language

Utilizing extensions developers can expand from the basic model class and associations, their own proprietary model to cover the management area particular to them. One of the many goals of CIM is that it is a much broader and cleaner modeling language to capture management characteristics. Allowing developers much greater control between the provider and the consumer of information (customers/agents) and between providers (systems).

Extension Schemas represent technology specific extensions of the Common Schema. Extending the CIM or a proprietary schema can mean several things. It can mean:

- Adding a property to an existing class or subclass of the CIM or a proprietary schema.
- Adding a new class or set of classes to the CIM or a proprietary schema.
- Creating your own namespace and schema.

Below is a list of supported schema modifications, some of which, when used, will result in changes in application behavior.

1. A class can be added to or deleted from a schema.
2. A property can be added to or deleted from a class.
3. A class can be added as a subtype or super type of an existing class.
4. A class can become an association as a result of the addition of an Association qualifier, plus two or more references.
5. A qualifier can be added to or deleted from any Named Element.
6. The Override qualifier can be added to or removed from a property or reference.
7. A class can alias a property (or reference, if the class is a descendent of an association), using the Alias qualifier. Both inherited and immediate properties of the class may be aliased.
8. A method can be added to a class.
9. A method can override an inherited method.
10. Methods can be deleted, and the signature of a method can be changed.
11. A trigger may be added to or deleted from a class.

In defining an extension to a schema, the schema designer is expected to operate within the constraints of the classes defined in the Core model. With respect to classification, it is recommended that any added component of a system be defined as a subclass of an appropriate Core model class. It is expected that the schema designer will address the following question to each of the Core model classes: "Is the class being added a subtype of this class?" Having identified the Core model class to be extended, the same question should be addressed with respect to each of the subclasses of the identified class. This process, which defines the superclasses of the class to be defined, should be continued until the most detailed class is identified.

Certain modifications to a schema can cause failure in applications that operated against

the schema prior to the modification. These modifications are:

1. Deletion of classes, properties, or methods.
2. Movements of a class anywhere other than down a hierarchy.
3. Alteration of property type or method signature.
4. Altering a reference range to anything other than the original specification.

### Modeling Techniques and Models

Schema design can be a complex process. There are numerous questions to answer when designing a schema, ranging from:

- From which schema do you need to start?
- Where does your application or device fit in that particular schema?
- What are the associations and dependencies?
- What future scenarios could come up that relate to your extension?

While there are any number of methods you can use to come up with answers to these questions, the most important factors to keep in mind when designing a schema are *efficiency* and *usability.*

### Purpose of a Schema

Before beginning to extend or design a schema, it's helpful to be cognizant of the purpose of a schema. A schema comes about because there is a need to model things that exist in the real world. So that others can understand us when we refer to these real things, we formalize them into statements in a language of some sort. The rules and procedures that link the real world to the formal world are known as *operational semantics*.

People use operational semantics to interpret the information supplied by a schema. For example, an operator, seeing that a database audit file is 70 percent full, switches to a new audit file, backs up the old one, and removes it. The operator's *interpretation* of the information ("audit file is 70 percent full") is what lends it meaning. Without the interpretation, or to put it another way, if the schema has no consequence, the schema and the information it provides is literally meaningless.

A schema is dependent on the existence of operational semantics; on its own, the schema is not meaningful. It acquires meaning only if someone is willing and able to interpret it. As a schema designer, you are dependent on, and must cater to, the level of understanding of the users and programmers that will make use of your schema. A schema is worthless if no one understands it or is willing to use it. The operational semantics (i.e., the ability of someone to interpret the schema) is an integral part, possibly the most important part, of the schema, and should always be in the forefront of your mind when putting a schema together.

When designing a schema, keep in mind that you are creating a language with which to communicate the structure of things (both physical and logical elements) within an information system. This structure will grow over time and will be interpreted by its various users. In the nature of things, you cannot define the absolute meaning of the structure, but you must strive for a reasonable balance between level of detail and understandability. You need to provide enough detail to be useful, but not so much as to make the schema overwhelmingly complex.

### Design Approaches

Schema design, like any human activity, takes place as a sequence of steps. A phased

iteration approach is a commonly favored means of building up the schema through a series of "passes" over the same material. One such approach to schema design involves working through iterations of three design phases.

Remember that schema design is an *iterative* process. Coming up with an effective schema requires that you try things that don't work and then start again until you come up with something that does work.

### Relational Model

Since many software developers are familiar with relational databases, it often helps to take the relational data model as a starting point for schema design. For schema design, it is important that you understand the relational model in that this model has a strong theoretical foundation that applies to data modeling in general. For the most part, if you can't express something in a relational model, there's a good chance you can't express it at all.

### General Goals of Relational Design

In creating a schema using the relational model, you generally strive to:

- Avoid redundancy
- Avoid inconsistencies between values
- Avoid modification anomalies (inconsistencies that occur when you insert or delete data)

### Record Design Issues

Records represent a convention for presenting information that has been elaborated upon in many different ways. Essentially, a record is a block of information arranged as a sequence of fields, each of which has an identifier, a type, and a value. For a given type of record the sequence of fields is generally the same from one record to the next.

Records are a convenient way of storing information and representing information in contexts such as user interfaces and reporting systems. Records have some severe limitations with respect to the information they are able to represent. Following are some of these limitations together with a discussion of their implications for designing records in a relational model. These implications also apply to schema design in the CIM and WBEM object-oriented models.

### Identifiers and Naming

There are a host of issues in the area of how to identify a record, not the least of which is that identifiers have at least two quite different uses. An identifier may be used to identify things to the *system*, commonly referred to as a **surrogate** or **object identifier**. A record may also be used to identify things to the *user*, commonly referred to as a **label**. In the relational model, CIM keys are used for both purposes.

### Relationships

Relationships are equally as important to an object-oriented model as to a relational model. Following are some issues related to relationships that you should keep in mind when designing schema.

Although there are no specific recommendations as to how to handle many of the relationship issues in object-oriented modeling, it is usually possible to come up with a

reasonable, workable solution if you design with these issues in mind.

### Object Model

Any object model implies a set of rules that should be observed in formulating a schema:

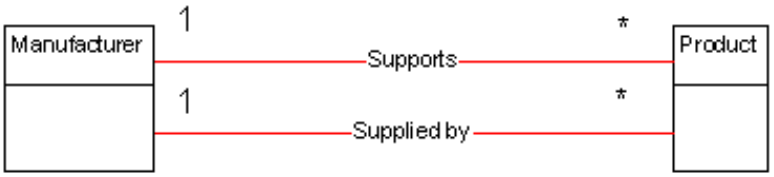
- **Distinct entity rule.** You should not have more than one instance in the same class that refers to the same object in the environment. It is quite acceptable to have instances in *different* classes that refer to the same object in the environment. For example. a network-attached printer may be represented by a system instance, a network node instance and a logical device instance, and a physical device instance. It should not be represented by two logical device instances.

- **Hierarchy rule.** The instances of every subclass should be a subset of the instances of its superclass. The subclass shouldn't have any instances that are not members of the superclass.

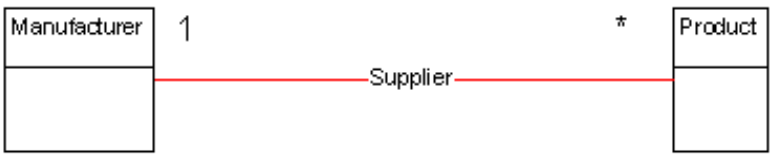
- **Attribute placement rule.** Attributes must be placed as low in the inheritance hierarchy as possible. This avoids null values. Attributes must be attached to the class they apply to. For example, it is a bad practice to represent Manufacturer as a property of a software feature if the manufacturer really is a property of the product of which the feature is a part. Manufacturer may be represented as a *derived* property, but its original definition must be attached to the class to which it properly applies.

- **Attribute derivation rule.** If you have a derived property, you should state that it is derived and from where in the schema it is derived. (Currently you may state this derivation in the description. In future versions of the CIM there will be a "derivation" qualifier).

- **Referential integrity rule.** Referential integrity is intact when structures do not break when you add and delete data. In particular, look for associations that represent two different directions of the same thing. If these kinds of associations exist, you may have a problem with referential integrity. For example, consider the context of **Manufacturer** and **Product** classes. Assume that a "supplied-by" relationship relates a product to a manufacturer and a "supports" relationship is between manufacturer and product. This can be represented as follows.




Further assume that it must be the case that if a manufacturer supplies a product it must also support it. And, if a product is supplied by a manufacturer, it must also be supported by it and none other. The schema must be reformulated as follows.



That is, the dual relationship, "Supports" and "Supplied-by" has been replaced by a single relationship, "Supplier." Note that whereas in the first schema example, a product can be supported by a manufacturer other than the one that supplied it, this is impossible in the second schema.

### Property Constraints

When extending an object model, you must also work with these Property Constraints:

- Intrinsic Types
- REQUIRED
- Max/Min
- MaxLen
- Range restrictions—syntax
- Key

These object property constraints are what are available in the CIM. Not everything you want to model can be expressed in these terms. You may need to write code or add information in the MOF descriptions to accommodate your particular needs.

### Design Issues with Object Models

There are certain issues that you will encounter when extending schemas. Many of these issues revolve around what you need to do to minimize maintenance issues that will invariably come about as the schema changes or evolves.

- **Be careful with relationships that have attributes.** These kind of relationships have a natural solution in WBEM in that the WBEM model directly supports relationships that have attributes. Be careful with this feature, however, as it is rare for relationships to have attributes without the relationship becoming an object in its own right. Always ask, "Does it make sense for these attribute values to disappear if the object on either end of the association disappears?"

- **Avoid N-ary relationships.** Relationships between more than 2 objects are rare in CIM and WBEM. Beware of creating multi-faceted relationships. It is almost always preferable to represent the objects as classes and then use associations between them.

- **Avoid complex entities.** A complex entity implies embedded objects. It is preferable to create objects with associations around them.

- **Beware of instances that have instances.** If you end up with instances that have instances, you will have to deal with a wide range of modification anomalies.

- **Handle time and state issues on your own.** CIM and WBEM do not currently include any structures designed to handle time, such as capturing "state," or saving logs or generating historical records. If your schema extension requires that kind of data modeling, you will need to come up with creative solutions on your own.

- **Look out for unknown values and uncertain values,** (as opposed to inapplicable values). There are three basic types of these values:

    1. The value exists, but you can’t get to it at that point in time. For example, once data that relates to the boot sequence is booted, that information is not available if it wasn’t saved anywhere.

2. The value is inapplicable. The value of "radius" is not applicable to a triangle.

3. The value is invalid. For example, a value that's not in the value map in the CIM Object Manager. It is possible for the CIM Object Manager to return a value that isn't in the value map, but that value is invalid and so there's nothing you can do with it.


# CIM Certification

Overview | CIM Schema | Extension Schema | CIM Certification | CIM Query Language

CIM Certification is a process which allows DMTF member hardware and software vendors the means to measure and publicly register compliance to the CIM Specification. CIM compliance offers the advantages of improved integration and data interoperability, and can help assist customers meet the increasing challenge of managing networked business systems and services.

The certification process requires the submission of a MOF File and a unique identifier string along with identifying which version of the standard CIM Schema to which the MOF file is compliant. The MOF must represent the model implemented in a software service/feature of the supplier's product and the supplier must “warrant and represent” that the MOF meets, and will continue to meet, the applicable conformance requirements. Developers can choose which CIM Schema(s) they would like to be recognized for complying with. The Certification process also allows the developer to submit the result(s) for certification or to delete the test results.

The CIM Certification Program is comprised of a software tool available for download from the DMTF Web site. For more information on how to participate in the CIM Certification Program, visit the DMTF web site. The CIM Certification Registration process allows the DMTF Member company to complete the registration form, sign the certification agreement form, submit any fees that might apply and identify the MOF file and a unique identifier string for which they are seeking certification.

Once a DMTF Member Company has completed the registration process successfully, they will begin formal testing of their MOF file and the unique identifier string utilizing multiple or single versions of the standard CIM Schema to which the MOF is compliant. Vendors can download and apply the compliance software tool to their CIM-based schema in the privacy of their development environments. The software tool is designed to check a vendor's schema for correct usage and model definition based on the CIM meta-schema, and maintenance of the CIM property and method definitions in subclasses.

Once satisfied with the results of the test, vendors can self-certify their schema as being compliant to the CIM Specification in a public registry on the DMTF Web site. In self-certifying, vendors may also publicly register any extensions they may have made to the CIM model. If the test failed, the member company can fix and resubmit the MOF, or they may elect to report the problem to the Certification Authority to assist in determining the source of the problem for correction and re-testing.

All information is confidential during certification process. Once certification is achieved, the DMTF Member company information and MOF become public information and are entered into the certification register. Test results are always confidential and the option to remain confidential for up to 6 months can be exercised once the MOF has been certified.

Any changes to a MOF  that has been previously Certified constitutes a new MOF. This includes any modifications, additions, or removal of any class, property or method. Any

changes to Certified MOF's will require re-testing and re-certification. If the MOF was only Renamed (modify identifier) with no changes to the MOF it would only require an amendment to the certification register. Member companies have the option to replace an existing entry in the certification register or add the new one.

The CIM Certification Program is only available to DMTF members. Certification of CIM-compliance must be renewed on an annual basis. Certification is valid only during the period in which the supplier is a DMTF member in good standing. The DMTF Member company must demonstrate that the Certified MOF continues to meet conformance requirements.








# CIM Query Language


Overview | CIM Schema | Extension Schema | CIM Certification | CIM Query Language

Work-in-progress by the DMTF WBEM Interoperability WG
To be released in 3Q2003

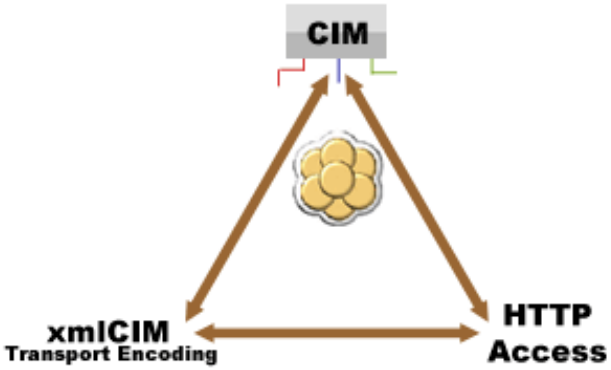

# Web Based Enterprise Management

WBEM Operations | xmlCIM | CIM-XML

The mission of the DMTF (Distributed Management Task Force) is to lead the development of management standards for distributed desktop, network, enterprise and internet environments. One of the goals of the DMTF is to "Promote interoperability among management solution providers".

This tutorial explains one of the ways in which the DMTF is achieving the goal of interoperability, called WBEM. WBEM is the acronym for Web Based Enterprise Management, a set of management and internet standard technologies developed to unify the management of enterprise computing environments. WBEM provides the ability for the industry to deliver a well-integrated set of standard-based management tools leveraging the emerging Web technologies.

WBEM has three main components, the Common Information Model (CIM), which provides a common format, language and methodology for collecting and decribing management data.The xmlCIM Encoding Specification defines XML elements, written in Document Type Definition (DTD), which can be used to represent CIM classes and instances. The CIM Operations over HTTP specification defines a mapping of CIM operations onto HTTP that allows implementations of CIM to interoperate in an open, standardized manner and completes the technologies that support WBEM.



CIM Tutorial > WBEM > WBEM Operations

# WBEM Operations

WBEM Operations | xmlCIM | CIM-XML

WBEM Operations (as described in the CIM Operations over HTTP Specification) define a set of operations that a WBEM client implements to operate in an open, standardized manner. WBEM Operations are Protocol Independent.

WBEM Operations can be Single (Individual) or Multiple Operations (Batched). Operation types include Data, Meta Data, Queries, and Methods.

All CIM Operation Message requests are defined as invocations of one or more methods. A method may be either:

- *Intrinsic*, which means that it is defined by the CIM Operations over HTTP Specification for the purposes of modeling a CIM operation, Intrinsic methods are further characterized by the fact that they are made against a CIM Namespace;
- *Extrinsic*, which means that it is defined as a method on a CIM Class in some Schema. Extrinsic methods are invoked on a CIM Class (if static) or Instance.

WBEM Operations include:

**GetClass** - used to return a single CIM Class from the target Namespace.

**EnumerateClasses** - used to enumerate subclasses of a CIM Class in the target Namespace.

**EnumerateClassNames** - used to enumerate the names of subclasses of a CIM Class in the target Namespace.

**GetInstance** - used to return a single CIM Instance from the target Namespace.

**EnumerateInstances** - used to enumerate instances of a CIM Class in the target Namespace.

**EnumerateInstanceNames** - used to enumerate the names (model paths) of the instances of a CIM Class in the target Namespace.

**GetProperty** - used to retrieve a single property value from a CIM Instance in the target Namespace.

**SetProperty** - used to set a single property value in a CIM Instance in the target Namespace.

**CreateInstance** - used to create a single CIM Instance in the target Namespace. The Instance MUST NOT already exist.

**ModifyInstance** - used to modify an existing CIM Instance in the target Namespace. The Instance MUST already exist.

**DeleteInstance** - used to delete a single CIM Instance from the target Namespace.

**CreateClass** - used to create a single CIM Class in the target Namespace. The Class MUST NOT already exist.

**ModifyClass** - used to modify an existing CIM Class in the target Namespace. The Class MUST already exist.

**DeleteClass** - used to delete a single CIM Class from the target Namespace.

**Associators** - used to enumerate CIM Objects (Classes or Instances) that are associated to a particular source CIM Object.

**AssociatorNames** - used to enumerate the names of CIM Objects (Classes or Instances) that are associated to a particular source CIM Object.

**References** - used to enumerate the association objects that refer to a particular target CIM Object (Class or Instance).

**ReferenceNames** - used to enumerate the association objects that refer to a particular target CIM Object (Class or Instance).

**ExecQuery** - used to execute a query against the target Namespace.

**GetQualifier** - used to retrieve a single Qualifier declaration from the target Namespace.

**SetQualifier** - used to create or update a single Qualifier declaration in the target Namespace. If the Qualifier declaration already exists it is overwritten

**DeleteQualifier** - used to delete a single Qualifier declaration from the target Namespace.

**EnumerateQualifiers** - used to enumerate Qualifier declarations from the target Namespace. The table below partitions the intrinsic methods into functional groups.

| Functional Group | Dependency | Methods |
|---|---|---|
| Basic Read | None | GetClass<br>EnumerateClasses<br>EnumerateClassName<br>GetInstance<br>EnumerateInstances<br>EnumerateInstanceName<br>GetProperty |
| Basic Write | Basic Read | SetProperty |
| Instance Manipulation | Basic Write | CreateInstance<br>ModifyInstance<br>DeleteInstance |
| Schema Manipulation | Instance Manipulation | CreateClass<br>ModifyClass<br>DeleteClass |

| | | |
|---|---|---|
| Association Traversal | Basic Read | Associators<br>AssociatorNames<br>References<br>ReferenceNames |
| Query Execution | Basic Read | ExecQuery |
| Qualifier Declaration | Schema Manipulation | GetQualifier<br>SetQualifier<br>DeleteQualifier<br>EnumerateQualifiers |

For more information on WBEM Operations refer to the CIM Operations over HTTP specification.


CIM Tutorial > WBEM > xmlCIM


# xmlCIM


WBEM Operations | xmlCIM | CIM-XML

The Extensible Markup Language (XML) is a simplified subset of SGML that offers powerful and extensible data modeling capabilities. An XML Document is a collection of data represented in XML. An XML Schema is a grammar that describes the structure of an XML Document.

The goal of xmlCIM is to create an XML grammar which can be written in DTD (Document Type Definition), and can be used both to represent CIM declarations (Classes, Instances and Qualifiers) and CIM Messages for use by CIM protocols.

There are potentially many different ways in which CIM information could be represented within XML. In the interests of interoperability between different implementations of CIM there is an obvious requirement for standardization of this representation. The following criteria have been applied in the design of the representation presented here:

- Fully standardized technologies are used wherever possible, in preference to Working Drafts.
- Completeness is favored over conciseness (all aspects of CIM should be modeled).

While the DMTF makes no restrictions on the use of this mapping, it is recognized that a number of possible usage scenarios exist for which the mapping should cater:

1. XML documents conforming to this mapping that express CIM declarations should be capable of being rendered or transformed using standard techniques into other formats. In particular the mapping should contain sufficient information to be rendered into MOF (Managed Object Format) syntax.
2. The mapping should be applicable to the wire-level representation of CIM Messages defined by the CIM mapping onto HTTP.

There are two fundamentally different models for mapping CIM in XML:

- A *Schema Mapping* is one in which the XML Schema is used to describe the CIM classes, and CIM Instances are mapped to valid XML Documents for that schema.
- A *Metaschema Mapping* is one in which the XML schema is used to describe the CIM metaschema, and both CIM classes and instances are valid XML documents for that schema.

Although there are obvious benefits to employing a schema mapping (more validation power, and a slightly more intuitive representation of CIM in XML), the metaschema mapping was adopted here for the following reasons:

- It only requires one standardized metaschema DTD for CIM, rather than an unbounded number of DTDs. This considerably reduces the complexity of management and administration of XML mappings.
- XML DTD does not allow an unordered list of elements. In a static mapping this

would mean either:
    - Fixing an arbitrary order for property, method and qualifier lists (making it harder for a receiving application to process), or;
    - Defining a very unwieldy mapping which took account of all list orderings explicitly (and whose size would grow exponentially with the number of list elements).
- In a schema mapping, the names of CIM Schema elements (class, property, qualifier and method names) populate the XML element namespace. In order to replicate the scoping rules on CIM element names within XML DTD, it would be necessary to employ XML Namespaces to define XML Schema to per-property level of granularity. This would be extremely cumbersome to administer and process. A metaschema mapping only introduces a small, fixed number of terms into the XML element namespace (such as "Class", "Instance", "Property", and so forth). As an alternative to the introduction of additional XML Namespaces, some renaming of CIM elements could be used (for example, prefixing a qualifier name with the name of its owning property and its owning class), but this would result in XML documents that were verbose and difficult to understand.
- Although a schema mapping could allow XML-based validation of instances against classes, this would only be possible if the entire class hierarchy were flattened prior to mapping the CIM Class to an XML Schema. If this flattening were not performed then inherited properties may be absent from the DTD, which would cause validation to fail against an instance that included the value of an inherited property.

The subset of the DTD for CIM presented in WBEM is concerned with object declarations, and is intended to allow expression of CIM objects in XML sufficient for rendering into a number of formats, of which MOF is one.

The semantic content of a MOF file is fully captured by the DTD presented herein, so that it is possible to express any MOF conformant to in an equivalent XML representation using this DTD. This includes the ability to express any of the standard MOF pragmas defined in, with the exception of the locale and instancelocale pragmas.





CIM Tutorial > WBEM > CIM-XML


# CIM-XML


WBEM Operations | xmlCIM | CIM-XML

CIM-XML is currently the only standards based protocol for exchanging CIM information. Before CIM-XML, the only standard way to exchange CIM information was through a MOF (Managed Object Format) file.

CIM-XML uses xmlCIM as the payload and HTTP as the transport. CIM-XML defines all interactions between CIM products as CIM messages.

A *CIM Message* is a well-defined request or response data packet used to exchange information between CIM products. There are currently two types of *CIM Messages*: *CIM Operation Messages* and *CIM Export Messages*.

- A *CIM Operation Message* is a CIM message used to invoke an operation on the target namespace.
- A *CIM Export Message* is a CIM message used to communicate information about a CIM namespace or element that is foreign to the target. A *CIM Export Message* is informational only and does not define an operation on the target CIM namespace or even imply the existence of a target namespace.

XML DTDs are restricted to certain terms to describe XML documents, but this document requires a further classification of an XML document with respect to a DTD. Henceforth the term *loosely valid* is defined to apply to an XML Document with the following characteristics:

- If any attributes or elements in the XML document which do not appear in the CIM XML DTD are removed, then the resulting document is valid with respect to the CIM XML DTD.

In effect, a loosely valid document is one which is valid with respect to the CIM XML Document Type Definition (DTD) apart from having additional attributes or elements not defined by that DTD. The concept is very similar to that of an *open content model* as defined by the working draft on XML Schemas, expressed within the more limited scope of DTDs. One corollary of this definition is that any XML document that is valid with respect to the CIM XML DTD is also loosely valid.

The motivation for introducing this class of XML Documents is to relax the restrictions on a CIM Client, CIM Server or CIM Listener when parsing received XML documents defined within the scope of this mapping. It is recognized that not all Clients (respectively, CIM Servers or CIM Listeners) should be required to validate each received CIM Message Response (respectively, CIM Message Request) as this would place too great a processing burden on the validating entity at the expense of footprint and performance, most notably in communication between robust and conformant implementations of this mapping.

Instead the following requirements are made by this specification:

- A CIM Client (respectively, CIM Server or CIM Listener) MAY include a DOCTYPE element in a CIM Message Request (respectively, CIM Message Response). If so, an External declaration SHOULD be used (in-lining of the complete DTD within a message is discouraged).
- A CIM Client (respectively, CIM Server or CIM Listener) MAY elect to validate a received CIM Message Response (respectively, CIM Message Request).
- If a CIM Client (respectively, CIM Server or CIM Listener) elects not to validate a received CIM Message Response (respectively, CIM Message Request), then loose validation MUST be enforced.

Finally, all CIM Operation requests are defined as invocations of one or more methods. A method may be either:

- *Intrinsic*, which means that it is defined by this specification for the purposes of modeling a CIM operation, or;
- *Extrinsic*, which means that it is defined as a method on a CIM Class in some Schema.





CIM Tutorial > Directory Enabled Networking


# Directory Enabled Networking


DEN Mapping

The Directory Enabled Network (DEN) initiative is designed to provide the building blocks for more intelligent management by mapping concepts from CIM (such as systems, services and policies) to a directory, and integrating this information with other WBEM elements in the management infrastructure. This utilizes existing user and enterprise-wide data already present in a company's directory, empowers end-to-end services, and supports distributed network-wide service creation, provisioning and management.

The use of CIM in defining a directory schema enables consistent schema for, and a common understanding of, directory information. Common schema and semantics are especially important when defining and decomposing platform-neutral, high-level policies. Integration within the WBEM infrastructure ties high-level, infrequently-changing directory data to the other, more real-time components of the management infrastructure.

### The goals of the DEN effort are:

1. to "direct" clients to relevant management services
2. to hold a subset of management data

### Current efforts are focused on defining directory schema and usage for:

- Common identity and security administration
- Common understanding of managed systems and services
- Information related to locations, groupings and policy

The data that must be shared in a directory is based on analyses of business scenarios and profiles, such as mobility management.





CIM Tutorial > Directory Enabled Networking > DEN Mapping


# DEN Mapping




CIM is defined to be technology and implementation-neutral, but to be useful, must be mapped into existing repositories and accessed via established transports and interfaces. One of the possibilities is to map CIM to a directory that supports the LDAP protocol and an X.500 model. However, there are several items to consider when doing this:

## 1. Appropriate Use of a Directory

Since LDAP directory servers are tuned for very fast reads and searches, they are usually poor performers when it comes to writes (edits), adds (new objects), and deletes. The CIM object lifecycle includes creation (adds), modification (writes), and deletion. Therefore, most aspects of the lifecycle are burdensome for the LDAP server, and LDAP servers generally should not be used for CIM object manager repositories. However, there are many CIM applications that spend a lot of time reading and searching for CIM objects and associations. When the ratio of reads and searches to CIM lifecycle operations is reasonably high, LDAP directories make excellent repositories, especially for CIM namespaces that span many hosts and devices. So what types of CIM classes and associations make good candidates for CIM to LDAP mapping? As a rule, those whose lifetimes are long relative to their number and those whose properties are not highly volatile. As it turns out, many areas of the CIM Core and Common Models fall into this category.

Users, computers, services, operating systems and software, service access points, policies, configurations and settings, physical elements, and others are examples of CIM managed elements that once created, have long lifetimes, and whose properties are pseudo-static. In the case of users and computers, the number of distinct objects and associations to those objects is generally quite high in an enterprise or service provider environment. In the case of configurations and policies, their modeled properties may change more rapidly than those of users or computer systems, but in general, that is balanced by there being relatively fewer of them. On the other hand, CIM_StatisticalData is an example of a class whose properties are highly volatile, and in general not a good candidate for LDAP mappings.

## 2. Naming Considerations

Although the rules for uniqueness of CIM objects are strict (basically, their key properties must have correct values), the rules for the Relative Distinguished Names (RDNs) of mapped CIM objects are flexible and vary by implementation. When choosing the RDN, one must consider whether the directory implementation restricts the choice of RDN attributes to "well-known" values (such as O, OU, etc.). DEN applications desiring interoperability should consider the DN of an entry to be a set of opaque attributes, without any particular semantic assigned to them. Specific semantics and searchable fields should be restricted to named class attributes, and not implied in a DN. An attribute lookup is preferred over the possibility that a particular attribute might be used as the RDN.

If it is desired to use the specified CIM "key" properties for naming, there are some items to consider. In combination with a deep namespace design, the use of CIM keys for directory RDNs can result in very long Distinguished Names (DNs). Depending on the directory server implementation, long DNs can have a negative impact on search speed and indexing. In general, short RDNs that are reasonably unique (at least unique with respect to their siblings in the namespace) are preferable.

### 3. Definition of the CIM Object "Tree"

All CIM classes are mapped as abstract classes, and subclassed as defined in the CIM Schema. (There is a minor exception to this rule in the case of ManagedElement, which subclasses from the directory's "top".) For each concrete class, there are two subclasses - the first is an auxiliary class and the second is a structural one. To understand this, consider the following example - a CIM_FooService class is defined that subclasses from CIM_Service. In the DEN mapping, it would be represented as follows (where inheritance is shown via indenting):

```
dn: dlmName=foobar, ou=corporate, dc=acmenetworks,dc=net
dlmName: foobar
objectClass: top
  objectClass: dlmlManagedElement (Abstract)
    objectClass: dlmlManagedSystemElement (Abstract)
      objectClass: dlmlLogicalElement (Abstract)
        objectClass: dlmlEnabledLogicalElement (Abstract)
          objectClass: dmllService (Abstract)
            objectClass: dlmlFooService (Abstract)
              objectClass: dlmlFooServiceAuxClass
              objectClass: dlmlFooServiceInstance
(Note that the prefix, "dlm" is an acronym for "DEN LDAP Mapping".)
```

The abstract classes are included in the hierarchy for two reasons:

1. they provide a natural hierarchy for modeling CIM inheritance
2. they provide a convenient object class for searches

By searching for the abstract class, an application can find both instance and auxiliary entries and therefore interoperate with other directory applications. The structural classes are used in cases where an entry does not pre-exist in the directory, and a new structural object is needed for instances to be defined. The auxiliary class form is used to overlay a new DEN semantic over an already existing directory entry - and add the CIM attributes to that entry.

### 4. Lack of Associations in the Directory

In many cases, associations are relatively static. This is not the main concern regarding their implementation in a directory. Because association classes define relationships between objects, directory namespace issues start to come into play. There are a few different ways that CIM associations can be mapped into LDAP directories, and namespace issues are a factor in all of these ways.

First, let us consider the alternatives for mapping associations into a directory. Associations can be handled by three approaches. The first uses auxiliary classes to attach a reference to the "other" endpoint of the association. The second approach uses a structural entry for the association and "helper" references to point to the association entry, from each endpoint of the association. Finally, some associations are not explicitly mapped, but are captured through DIT containment. Examples of the latter are the weak associations from the Physical Model, that logically fit into the directory concept of DIT

containment.

Returning to the namespace issue, every environment and application will have different requirements for their directory namespaces. Therefore, each implementation must carefully consider the implications of which CIM objects and associations will be mapped, where they will be placed in the namespace, and how the namespace will be partitioned. For a non-partitioned namespace, some of these problems may go away. However, in the general case, directory services will span physical servers, and in the absence of multi-master replication capability, the nature and namespace location of objects that are replicated must be carefully considered.

For example, consider what will happen if an association relates objects from partitioned portions of a directory namespace. Following the association may cause the directory server to generate LDAP referrals to the client, forcing the client to connect and (usually) authenticate to the server partition holding the associated object. Chasing referrals is a relatively inefficient operation for most application clients. Given CIM's heavy usage of associations, the number of referrals required for common association traversals must be minimized. This can be accomplished by a namespace design that is sensitive to the association considerations of a CIM mapping.

For example, a HostedService association and a ServiceToServiceDependency are two examples where it behooves the namespace designer to place the associated objects within portions of the namespace that will not be partitioned. ElementSetting may be another example since finding a CIM_Setting for a given element may be done frequently. However, ServiceAccessBySAP may be less of a problem, unless the connection span for accessing the service by a client is very short, requiring may association traversals over short time intervals.

### 5. Versioning

Class names include a version number. This is necessary because the mapping of OID to object class name to object class description is 1:1:1. That is, once the object class definition changes, it is required to assign both a new name and OID. And, a change to the definition results in a new branch of the object class tree - as all children classes must also have their version information incremented. This makes finding a specific class very cumbersome.

To support multiple versions, an auxiliary "indicator" class is defined in the DEN mappings. This takes the form, "dlmIndicatorAuxClass". This auxiliary class can be attached to any object class to permit locating a particular class, regardless of version number. For example, consider the CIM_FooService class from the previous example. This class may be modified to add a property. So, the two possible class names in the directory would be dlm1FooService or dlm2FooService. But, either version could be found by searching on dlmFooIndicatorAuxClass!








# Distributed Management Task Force


Organization | Working Groups | Policies | Membership | Standards | Contact Info

The goal of this section is to introduce the DMTF, the DMTF organizational structure and the DMTF processes. This section will introduce the membership levels as well as the advantages and value of membership. Finally, it will describe how member and non-member companies can get involved, provide feedback, and ask for help.

## DMTF Introduction

The DMTF is a not-for-profit, vendor-neutral collaborative body that is leading the development, adoption and unification of management standards and initiatives for desktop, enterprise and internet environments. The DMTF is chartered to adopt, create and maintain the specifications and technologies that provide management tools with the ability to discover, deploy and control management data in a standard way. Working with key technology vendors and affiliated standards groups, the DMTF is enabling a more integrated, cost-effective, and less crisis-driven approach to management through interoperable management solutions.

## DMTF Mission Statement

To lead the development of management standards for distributed desktop, network, enterprise and internet environments.

## DMTF Goals

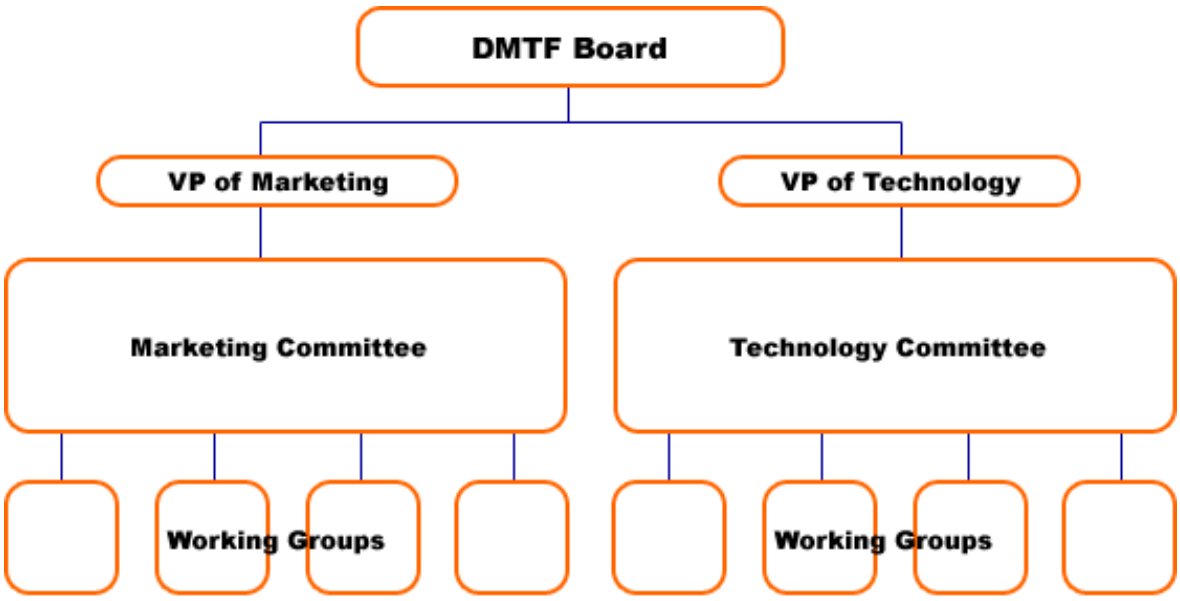
1. Accelerate adoption of management standards.
2. Unify industry management initiatives.
3. Promote interoperability among management solution providers.


CIM Tutorial > DMTF > Organization

# DMTF Organization

Organization | Working Groups | Policies | Membership | Standards | Contact Info

This diagram shows the DMTF organizational structure. The next page explains the roles and responsibilities of each group.






CIM Tutorial > DMTF > Working Groups


# DMTF Working Groups & Committes


Organization | Working Groups | Policies | Membership | Standards | Contact Info

## DMTF Board

The DMTF Board is responsible for the overall direction, strategy and activity of the DMTF, including managing DMTF finances, approving technical and marketing initiatives and leading DMTF committee work.

## Marketing Committee

The Marketing Committee communicates with the industry, the public, and members about the activities of the organization. The working groups listed below are overseen by the Marketing Committee.

## Marketing Committee Working Groups

- WBEM
- Customer Advisory Board

## Technical Committee

The Technical Committee develops standards and initiatives for the DMTF and is responsible for coordinating all the technical activities of the DMTF including the Common Information Model (CIM), The Desktop Management Interface (DMI), the Directory Enabled networks (DEN) and the associated technologies required to deliver Web Based Enterprise Management (WBEM) to the industry. The Technical Committee oversees the working groups listed below.

## Technical Committee Working Groups

- Applications
- Architecture Working Group
- Database
- Desktop Management Interface (DMI)
- Lightweight Directory Access Protocol (LDAP)
- Networks
- Policy
- Pre-OS
- Security Protection and Management WG
- Support
- System and Devices
- User and Security
- WBEM Interoperability

*Note: DMTF working groups are established and dissolved on an as-needed basis.*

*Workgroups List Compiled June 09, 2003.*

To learn more about either the Marketing Committee or the Technical Committee refer to the DMTF website.





CIM Tutorial > DMTF > Policies


# DMTF Policies


Organization | Working Groups | Policies | Membership | Standards | Contact Info

Every organization should have a set of Operating Policies to use as a guide to running their operations. The DMTF is no exception and has made public the information on the technology adoption process, how to gain access to the various DMTF committees, and policies regarding privacy and the distribution of other information.

## Bylaws

The bylaws define the purpose of the organization, the principal office of the corporation, the members, directors, officers, and conflicts of interest.

## Committee / Information Access

An overview of DMTF committees and organization, working groups, the voting process, and access policy.

## Patent Policy

Instructions and rules dictating submittals to the DMTF for consideration of a patent.

## Press Release Policy

Policy regarding joint press releases and supporting quotes; what may be requested, requirements, the approval process for each.

## Privacy Policy

The DMTF's policy on the collection of personal data, domain information, cookies, security, and disclosure.

## Release Process

Details the process of releasing DMTF specifications and schema. The release process is five phases, from development, comment, review, and preliminary and final standards. Also details the numbering and errata methods.

## Member Rules of Conduct

A DMTF objective is also to position its members as leaders in the management community. These guidelines will help the DMTF supply its members with quality public relations opportunities while strengthening, clarifying, and protecting DMTF initiatives, specifications, and standards.

## Technology Adoption Policy

Rules dictating new technology adoption: instructions for submittal of new technology, possible courses of action, press release/resubmit options.

If you have questions after reading the information here, please contact dmtf-info@dmtf.org . If you want additional information on any of the above polices, please refer to the DMTF web site Operating Policies page.





CIM Tutorial > DMTF > Membership


# DMTF Membership


Organization | Working Groups | Policies | Membership | Standards | Contact Info

The DMTF offers several levels of membership for corporations, industry organizations, and academic institutions. These memberships offer different levels of responsibility and privilege, allowing vendors to gain the most value from the DMTF.

| Membership Level | Annual Dues | Voting Rights | Closed/Open | Working Group Participation | Marketing Activities |
|---|---|---|---|---|---|
| Board Member | $25,000 | Full | Closed (Elected) | Yes (unlimited) | Yes |
| Contributing Member | $10,000 | Limited | Open (Public) | Yes (unlimited) | Yes |
| Associate Member | $5,000 | None | Open (Public) | Yes (1 per committee) | Yes |
| Monitoring Member | $2,500 | None | Open | No | Yes |
| Alliance Partner | FREE | None | Closed (not-for-profit standards organizations) | Yes (determined by DMTF board) | No |
| Academic Alliance | FREE | None | Closed (accredited Institutions of higher learning) | Yes (1 per committee) | No |

Members also benefit from front-line access to information about activities, the opportunity to participate in the process of defining standards, and the synergy of working with other vendors who are addressing similar implementation issues. In addition, all members gain access to the member-only web site and other materials for review, including specifications, white papers and e-mail aliases.





CIM Tutorial > DMTF > Standards


# DMTF Standards & Initiatives


Organization | Working Groups | Policies | Membership | Standards | Contact Info

The DMTF is the industry organization that is leading the development, adoption and unification of management standards and initiatives for desktop, enterprise and internet environments.

Some of the approved Standards and Initiatives that the DMTF has made public are listed below.

Common Information Model (CIM)

This is a common data model of an implementation-neutral schema for describing overall management information in a network/enterprise environment.

Web-Based Enterprise Management (WBEM)

This initiative is a set of management and Internet standard technologies developed to unify the management of enterprise computing environments.

Directory Enabled Network (DEN) Initiative

The Directory Enabled Network (DEN) initiative is designed to provide building blocks for intelligent management by mapping concepts from CIM (such as systems, services and policies) to a directory, and integrating this information with other WBEM elements in the management infrastructure.

Desktop Management Interface (DMI)

These standards generate a standard framework for managing and tracking components in a desktop pc, notebook or server.

Alert Standard Format (ASF)

This specification defines remote control and alerting interfaces that best serve clients' OS-absent environments.

System Management BIOS (SMBIOS)

The SMBIOS Specification addresses how motherboard and system vendors present management information about their products in a standard format by extending the BIOS interface on Intel architecture systems.





CIM Tutorial > DMTF > Contact Info


# DMTF Contact Info


Organization | Working Groups | Policies | Membership | Standards | Contact Info

The DMTF provides various e-mail aliases for questions and support. The following aliases are available for both DMTF members and non-members.

- If you have questions or comments on the interpretation of the CIM Specification, CIM Schema, or WBEM Specifications, send email to cim-support@dmtf.org
- If you have questions or comments on the interpretation of the DMI specification, please send email to dmi-support@dmtf.org
- For information regarding DMTF public relations activities, or if you are an industry journalist or analyst and need specific information about the DMTF or DMTF technologies, please contact the DMTF's public relations firm, Crane Communications, at 503-220-1668.
- Any questions or comments about the website or this document, may be directed to webmaster@dmtf.org

If you have any general questions regarding the working groups, please contact the appropriate working group chair:

| | |
|---|---|
| **Membership Committee** | vp-membership@dmtf.org |
| **Marketing Committee** | vp-marketing@dmtf.org |
| Customer Advisory Board | cab-chair@dmtf.org |
| WBEM Working Group | wg-wbem-chair@dmtf.org |
| **Technical Committee** | vp-technology@dmtf.org |
| Applications Working Group | wg-apps-chair@dmtf.org |
| Database Working Group | wg-database-chair@dmtf.org |
| DMI Working Group | wg-dmi-chair@dmtf.org |
| LDAP Working Group | wg-ldap-chair@dmtf.org |
| Networks Working Group | wg-network-chair@dmtf.org |
| Policy Working Goup | wg-policy-chair@dmtf.org |
| Pre-OS Working Group | wg-pos-chair@dmtf.org |
| Security Protection and Management | wg-secpam@dmtf.org |
| Support Working Group | wg-support-chair@dmtf.org |
| Systems & Devices Working Group | wg-sysdev-chair@dmtf.org |
| User & Security Working Group | wg-user-chair@dmtf.org |
| WBEM Interoperability Working Group | wg-interop-chair@dmtf.org |

If you have general questions about the DMTF or DMTF technologies, you can reach the DMTF at:

Distributed Management Task Force
c/o Kavi Corporation
225 SE Main Street
Portland, OR 97214
(503) 963-3505
(503) 238-8721 (fax)
dmtf-info@dmtf.org








# DMTF Glossary

A B C D E F G H I J K L M
N O P Q R S T U V W X Y Z

## - A -

**aggregation relationship**
A relationship in which one entity is made up of the aggregation of some number of other entities.

**alias**
A symbolic reference in either a class or instance declaration to an object located elsewhere in a MOF file. Alias names follow the same rules as instance and class names. Aliases are typically used as shortcuts to lengthy object paths. To define an alias, add the phrase "as $AliasName" to the instance or class declaration.

**application life cycle**
The concept of applications used in an organization falling into various stages including deployable, installable, executable, and executing.

**association class**
A class that describes a relationship between two classes or between instances of two classes. The properties of an association class include pointers, or references, to the two classes or instances. The **Association** qualifier is attached to every association class for identification. Although inclusion is not mandatory, in the Microsoft implementation of the CIM classes can be included in one or more association classes.

**asynchronous method**
A method that returns to the caller immediately regardless of whether processing has completed. The results of processing are returned through another call on another thread. Asynchronous methods free the caller from having to wait until processing has finished. Asynchronous methods have an **Async** suffix.

Back to Top

## - B -

**Backus-Naur Form (BNF)**
Backus-Naur Form. A metalanguage that specifies the syntax of programming

languages.

Back to Top

## - C -

**cardinality**

The number of instances of a class related to another class' instance via an association. For example, if an association relates ClassA to ClassB, then A's cardinality indicates how many instances of ClassA may be associated with a single instance of ClassB. Cardinality does NOT dictate the number of instances of the association or of the individual classes.

**CIM Schema**

A collection of class definitions used to represent managed objects that occur in every management environment. Also see core model , common model , and extension schema .

**Common Information Model (CIM)**

Describes the components of a managed computing and networking environment using an object-oriented modeling approach. CIM is comprised of a Specification and a Schema. The Specification defines the details for integration with other management models (i.e., SNMP's MIBs or the DMTF's MIFs) while the Schema provides the actual model descriptions.

**Common Information Model Object Manager (CIM Object Manager)**

A component in the CIM Server that handles the interaction between management applications and providers . The CIM Object Manager supports services such as event notification, remote access, and query processing .

**common model**

The second layer of the CIM Schema , which includes a series of domain-specific but platform-independent classes. The domains are systems, networks, applications, and other management-related data. The common model is derived from the core model . Also see extension schema .

**core model**

The first layer of the CIM Schema , which includes the top-level classes and their properties and associations . The core model is implementation independent. Also see common model and extension schema .

Back to Top

## - D -

**Directory**

A directory is, in general, an approach to organizing information, the most familiar example being a telephone directory.
On the World Wide Web, a directory is a subject guide, typically organized by major

topics and subtopics.
In computer file systems, a directory is a named group of related files that are separated by the naming convention from other groups of files.
In computer networks, a directory is a collection of users, user passwords, and, usually, information about what network resources they can access.

**Directory Enabled Network (DEN)**

A mapping of CIM to various repositories with the goal of providing complete management data. Specifically, it is a mapping to a form that can be implemented in a directory that uses (L)DAP as its access protocol.

**Distributed Management Task Force (DMTF)**

A not-for-profit organization, delivering technology standards for end-to-end management of distributed computing environments.

**Desktop Management Interface (DMI)**

An initiative by the DMTF. The DMI allows desktop computers, hardware and software products, and peripherals — whether they are standalone systems or linked into networks — to be manageable and intelligent. It allows them to communicate their system resource requirements and to coexist in a manageable PC system. The DMI is independent of operating system and processor, enabling the development of manageable PC products and applications across platforms.

**domain**

The class to which a property or method belongs. For example, if **status** is a property of Logical Device, it is said to belong to the Logical Device domain.

Back to Top

## - E -

**extension schema**

The third layer of the CIM Schema , which includes platform-specific extensions of the CIM Schema. Also see common model and core model .

Back to Top

## - I -

**indication**

An operation executed as a result of some action such as the creation, modification, or deletion of an instance , access to an instance, or modification or access to a property . Indications can also result from the passage of a specified period of time. An indicationtypically results in an event.

**Information Model**

An operation executed as a result of some action such as the creation, modification, or deletion of an instance , access to an instance, or modification or access to a property . Indications can also result from the passage of a specified period of time.

An indicationtypically results in an event.

**inheritance**

The relationship that describes how classes and instances are derived from parent classes, or superclasses . A class can spawn a new subclass , also called a child class. A subclass contains all the methods and properties of its parent class. Inheritance is one of the features that allows the CIM classes to function as templates for actual managed objects in the CIM environment.

**instance**

A representation of a real-world managed object that belongs to a particular class, or a particular occurrence of an event. Instances contain actual data.

**Interface Definition Language (IDL)**

A generic term for a language that lets a program or object written in one language communicate with another program written in an unknown language.

Back to Top

## - K -

**key**

A property that is used to provide a unique identifier for an instance of a class. Key properties are marked with the **Key** qualifier.

**Key qualifier**

A qualifier that must be attached to every property in a class that serves as part of the key for that class.

Back to Top

## - L -

**LDAP (Lightweight Directory Access Protocol)**

A "lightweight" (smaller amount of code) version of Directory Access Protocol (DAP). It is a standard for directory services in a network.

**local property**

A property defined for a class but not inherited from a superclass.

Back to Top

## - M -

**managed object**

A hardware or software system component that is represented as an instance of the

CIM class. Information about managed objects is supplied by data and event providers, as well as by the CIM Object Manager.

**Managed Object Format (MOF)**
A compiled language for defining classes and instances . A MOF compiler offers a textual means of adding data to the CIM Object Manager repository. It is based on the Object Management Group's (OMG's) Interface Definition Language (IDL), and can be encoded using either UTF-8 or Unicode.

**management application**
An application or service that uses information originating from one or more managed objects in a managed environment. Management applications retrieve this information through calls to the CIM Object Manager API from the CIM Object Manager and from providers .

**management client**
A management client is a program that initiates management requests. The management client interacts with a management infrastructure or agent (via a protocol and data model) to perform management operations.

**Management Information Format (MIF)**
Part of DMI that stores and manages information and passes it to management applications on request. MIFs define the standard manageable attributes of PC products in categories including PC systems, servers, printers, LAN adapters, modems, and software applications.

**metamodel**
A CIM component that describes the entities and relationships representing managed objects . For example, classes, instances, and associations are included in the metamodel.

**metaschema**
The metaschema is a formal definition of the model. It defines the terms used to express the model and its usage and semantics.

**method**
1. A function describing the behavior of a class. Including a method in a class does not guarantee an implementation of the method. The **Implemented** qualifier is attached to the method to indicate that an implementation is available for the class.
2. A function included in a CIM Object Manager API interface.

Back to Top

## - N -

**Named Element**
An entity that can be expressed as an object in the meta schema.

**namespace**
A unit for grouping classes and instances to control their scope and visibility. Namespaces are not physical locations; they are more like logical databases containing specific classes and instances.

Back to Top

## - O -

**object path**
A formatted string used to access namespaces , classes, and instances . Each object on the system has a unique path which identifies it locally or over the network. Object paths are conceptually similar to Universal Resource Locators (URL).

**override**
Indicates that the property, method, or reference in the derived class overrides the similar construct in the parent class in the inheritance tree or in the specified parent class.

Back to Top

## - P -

**property**
A name/value pair that describes a unit of data for a class. Property names cannot begin with a digit and cannot contain white space. Property values must have a valid Managed Object Format (MOF) data type.

**provider**
Communicates with managed objects to access data and event notifications from a variety of sources, such as the system registry or an SNMP device. Providers forward this information to the CIM Object Manager for integration and interpretation.

Back to Top

## - Q -

**qualifier**
A modifier containing information that describes a class, an instance , a property , a method , or a parameter.

**qualifier flavor**
The CIM defined flag that governs the use of a qualifier . The CIM flavors describe rules that specify whether a qualifier can be propagated to derived classes and instances and whether or not a derived class or instance can override the qualifier's original value.

Back to Top

## - R -

**range**
The class that is referenced by a reference property.

**reference**
A special string property typethat is marked with the reference qualifier , indicating that it is a pointer to other instances.

**required property**
A property that must have a value.

Back to Top

## - S -

**schema**
A collection of class definitions that describe managed objects in a particular environment.

**selective inheritance**
The ability of a descendant class to drop or override the properties of an ancestral class.

**Simple Network Management Protocol (SNMP)**
A protocol for network management and monitoring. SNMP is defined by the Internet Engineering Task Force.

**standard schema**
A common conceptual framework for organizing and relating the various classes representing the current operational state of a system, network, or application. The standard schema is defined by the Distributed Management Task Force (DMTF) in the Common Information Model (CIM) .

**subclass**
A class that is derived from a superclass . The subclass inherits all features of its superclass, but can add new features or redefine existing ones.

**subschema**
A part of a schema owned by a particular organization.

**superclass**
The class from which a subclass inherits .

Back to Top

## - T -

**trigger**

A recognition of a state change (such as create, delete, update, or access) of a class instance, and update or access of a property. Note: The CIM implementation does not have an explicit object representing a trigger. Triggers are implied either by the operations on basic objects of the system (create, delete, and modify on classes, instances and namespaces) or by events in the managed environment.

Back to Top

## - U -

**Unified Modeling Language (UML)**

A graphical representation (using boxes and lines) of objects and relationships. The Object Management Group (OMG) publishes the UML Specification.

**Unicode**

A 16-bit character set capable of encoding all known characters and used as a worldwide character-encoding standard.

**UTF-8**

An 8-bit transformation format that may also serve as a transformation format for Unicode character data.

Back to Top

## - W -

**Web-Based Enterprise Management (WBEM)**

Web-Based Enterprise Management (WBEM) is an initiative based on a set of management and Internet standard technologies developed to unify the management of enterprise computing environments. WBEM provides the ability for the industry to deliver a well-integrated set of standard-based management tools leveraging the emerging technologies such as CIM and XML.

Back to Top