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

This tutorial is designed for both beginner and intermediate users. This tutorial 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, you should have 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 later
- **800 x 600** or higher screen resolution

**Tutorial Navigation**

You can navigate through this tutorial in the following ways:

- Index - Click any link in the index on the left-hand side of the screen to go directly to that topic. For example, if you are already familiar with the Common Information Model and are interested only in learning about the CIM Schema, click the Common Information Model link in the index and then click CIM Schema from the navigation tool bar.
- Next/Back - To read the tutorial in order, just click on the Next image in the lower right-hand corner of the screen.

**Note:** If you click on a link or access the glossary you must use the index to return to the tutorial.

**Downloading the Tutorial**

To run the tutorial locally on your computer, download the [DMTF Tutorial](#) file.
Managing the Distributed Enterprise

During the evolution of the information technology (IT) industry many advances in IT that have provided businesses with better efficiency and new opportunities. Processes that once were costly both in human effort and time resources can now be completed with progressively less human intervention and at much faster speeds. Such improvements have created higher Return on Investment (ROI) from IT budgets and new business opportunities made possible by newly available capital and resources. However, along with the productivity gains, added revenues, reduced overhead and increased flexibility that these IT advances have brought, management complexities have been introduced.

Because many of the systems that support these new business models were introduced sporadically and adopted on an as-needed basis, many companies are now have a plethora of disparate networks, systems, applications, and management software. A complex web of ad hoc integration frequently emerges to support the flow of information among these applications. Continuous business changes add to the complexity of interrelationships among networks, systems and applications. This situation is currently impeding the ability of many companies to evolve their current systems to accommodate new business requirements and organizational needs.

Recognizing this problem, many companies are demanding a strong, standards-based integration solution that enables them to leverage their existing IT assets and better position themselves for future growth. Furthermore, companies are reluctant to make expensive up-front investments in integration technologies and services that might take years to pay off.

In an effort to address these issues, the Distributed Management Task Force (DMTF) was founded as a standards-based organization with a charter to lead 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 and cost-effective approach to IT management through interoperable solutions.

One standard developed by the DMTF is the Common Information Model (CIM), a model for describing management information. The DMTF provides both a specification and a schema. the CIM Infrastructure specification defines the CIM rules and semantics. The CIM Schema provides the actual model definitions.

The CIM Specification is the language and methodology for describing management data. The CIM Schema includes models for Systems, Applications, Networks, Databases and Devices among other management areas. The CIM Schema enables applications from different vendors 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.

Companies implementing solutions base on CIM and Web-Based Enterprise Management (WBEM) are able to realize the following benefits:

1. Reduce total cost of ownership (TCO) by enabling interoperable management of systems and devices in less time and with less effort.
2. Improve time to market and gain a competitive advantage by using standard-based models.
3. Reduce development costs by using and re-using existing standards models.
4. Leverage new opportunities by extending existing standard models.

Some advantages of CIM and WBEM are as follows:

- **Independence from platform, programming language and compiler.** The WBEM transport protocol is independent of platform, programming language and compiler. Developers do not need to create and support development tools for specific platforms or programming languages.

- **Independence from information model.** The WBEM transport protocol is independent of the data that it communicates. Introducing new devices or features does not affect communications between management applications and the devices that they manage.

- **Extensibility.** To add new management capabilities, an instrumentation developer can simply extend their existing management model to include new management information.

- **Easy integration of new management capabilities.** Management applications can easily leverage evolving management capabilities without needing to consume additional management interfaces.

- **Security and Internet accessibility.** The WBEM transport protocol is secure and Internet-capable. Instrumentation developers can safely expose prototype management interfaces to management application developers over the Internet to accelerate development and debugging. Field test and development support hardware become unnecessary.

- **Development tools and resources.** Several commercial and open-source tools facilitate development of CIM and WBEM management interfaces.
DMTF technologies are designed as building blocks. When used in conjunction they enable solving real world problems for distributed enterprise management.

**Technology Diagram**

![Technology Diagram](image)

### Common Information Model (CIM)

The Common Information Model (CIM) is the foundation for the DMTF technology solution to distributed enterprise management. CIM is an object-oriented management information model based on UML which provides a conceptual framework for describing management data. The *CIM Infrastructure Specification* defines the meta schema, syntax, rule and Managed Object Format (MOF). The MOF syntax is based on the Interface Definition Language (IDL) and provides a way to describe CIM Object definitions in a textual form. The CIM Schema provides a common conceptual framework needed to describe a managed environment.

### Web-Based Enterprise Management (WBEM)

Web-Based Enterprise Management (WBEM) is a set of management and Internet standard technologies developed to unify the management of distributed computing environments. WBEM provides the ability to exchange CIM information in an interoperable...
and efficient manner. WBEM includes protocols, query languages, discovery mechanisms, mappings, and anything else needed to exchange CIM information.

Profiles

A Profile is a specification that defines the CIM model and associated behavior for a management domain. The CIM model includes the CIM classes, associations, indications, methods, and properties. The management domain is a set of related management tasks. A Profile is uniquely identified by name, organization name, and version.

Management Initiatives

An Initiative is designed to deliver a solution for a specific area of a vertical market. An Initiative includes a set of Profiles and a reference to the applicable WBEM specifications to address a specific area of management. Examples are the DMTF Systems Management Architecture for Server Hardware (SMASH) Initiative for managing servers, and the Storage Networking Industry Association (SNIA) Storage Management Initiative (SMI).

The rest of this tutorial details these technologies.
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 a more specific portion of CIM, please use the navigation tool bar above to jump to the topic of interest.

Introduction to CIM

The Common Information Model (CIM) is a conceptual information model for describing computing and business entities in Internet, enterprise and service provider environments. It provides a consistent definition and structure of management information using object oriented techniques. CIM includes expressions for common elements that must be clearly presented to management applications like 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 in a text format is Managed Object Format (MOF).
The CIM is a hierarchical, object-oriented management information model that facilitates defining the various interdependencies and relationships between 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 upon which it depends.

The CIM is an information model, a conceptual view of the managed environment, that unifies and extends existing instrumentation and management standards (SNMP, DMI, CMIP, etc.) using object-oriented constructs and design. The CIM does not require any particular instrumentation or persistent information repository format. It is only an information model defining the management information in an object-oriented fashion.

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 common areas of management and is independent of implementation.

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 describes the schema and includes a description of the core and common models.
Object Orientated Overview

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 model and how 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. The following are some basics to keep in mind with regards to object-oriented modeling:

- **Instances** are things.
- **Properties** are attributes.
- **Relationships** are sets 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, object-oriented modeling is used to model hardware and software elements.

For illustrative purposes the following “Cheeseburger Example” is provided 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 Cheeseburger - is good to eat and fun to cook.

**Modularity**: Decomposition of abstractions into discrete units.

Example: The various “layers” of a Cheeseburger. (e.g., bun, lettuce, ketchup, mayonnaise, burger, cheese, onions, pickles, 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?
**Hierarchy**: A ranking or ordering of abstractions.

Example: A Cheeseburger is really a subclass of a Hamburger with the addition of cheese, which is a subclass of Sandwich, which is in turn a subclass of the Hierarchical superclass Food.

**Key Elements:**

- **Classes** – A collection of the definitions of state, behavior, and/or identity.
  - Properties
  - Methods
- **Objects**– Instances of a class.
- **Associations**– Relationships between classes or instances of classes.
  - Dependency
  - Identity
  - Aggregation
  - Composition
  - And others

Copyright © 2002-2006 Distributed Management Task Force, Inc. and WBEM Solutions, Inc.
All rights reserved.
The Common Information Model (CIM) is an approach to the management of systems and networks that applies the basic structure and conceptualization techniques of the object-oriented modeling paradigm. The approach uses a uniform modeling formalism that supports the cooperative development of an object-oriented schema.

The *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 (e.g., classes, properties, methods, indications and associations).

The specification defines the syntax and rules of the model. 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 additionally defines the CIM Naming mechanism.

The CIM Specification does not describe specific CIM implementations, API's, or communication protocols. These 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 defined independently.
The Meta Schema is a formal definition of the model. It defines the terms used to express the model and the model's usage and semantics.

The elements of the model are Schemas, Classes, Properties and Methods. The model also supports Indications and Associations as special types of Classes and References as a special type of Property. The rest of this section describes each of the elements in detail.
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 schema. Each class name includes the schema name and follows this format:

```
SchemaName_ClassName
```

The rules for schema naming are as follows:

1. "CIM" and "PRS" are reserved schema names and **MUST NOT** be used by any company or organization other than the DMTF.
2. The schema name **MUST** be unique and **MUST** begin with an alphabetic character. It is recommended to use one of the following methods to assure uniqueness:
   1. 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_”
   2. Make use of your company or organization's registered DNS entry in the manner outlined below:
      - 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.
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 must belong to only one schema and the class name must be unique within that schema. A fully qualified class name includes the schema name using the following format: SchemaName.ClassName.

Example: MOF definition of class CIM_ManagedElement

```moindent
class CIM_ManagedElement {
    string Caption;
    string Description;
    string ElementName;
};
```
A property is a value used to denote a characteristic of a class. A property is scoped by the class in which it is defined and must be unique within the class. A property has a name, data type, value and optionally a 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 these datatypes.

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

Examples
string Caption;

uint16 EnabledDefault = 2;

[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") ]
A method is an operation that can be invoked. Methods are scoped by the class in which they are defined 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 or array of object references. A parameter can be an input parameter, an output parameter or both. The type of parameter is specified using the IN and/or OUT qualifiers.

**Examples**

```cim.Content
[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 "

```
```
```cim.Content
[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.")]
```
```
"the subclass as a Values array qualifier.") }
uint32 StartService();
Qualifiers 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 defined and the qualifier type must be unique within that namespace.

All qualifiers have a name, type, value, scope, flavor and an optionally a 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 in which it was defined. Qualifiers can also specify 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.

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Definition</th>
<th>Default</th>
</tr>
</thead>
<tbody>
<tr>
<td>EnableOverride</td>
<td>The qualifier can be overridden</td>
<td>yes</td>
</tr>
<tr>
<td>DisableOverride</td>
<td>The qualifier can not be overridden</td>
<td>no</td>
</tr>
<tr>
<td>ToSubClass</td>
<td>The qualifier is inherited by any subclass</td>
<td>yes</td>
</tr>
<tr>
<td>Restricted</td>
<td>The qualifier applies to the class in which it is declared</td>
<td>no</td>
</tr>
<tr>
<td>Translatable</td>
<td>Indicates the value of the qualifier can be specified in multiple locales</td>
<td>no</td>
</tr>
</tbody>
</table>

The scope defines the meta elements to which the qualifier can be applied. The scope must contain at least one meta element, but can contain a combination of meta elements or the 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.
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 {

Copyright © 2002-2006 Distributed Management Task Force, Inc. and WBEM Solutions, Inc.
All rights reserved.
A reference is a special property data type that is declared with the REF key word and the name of the class to which it refers.

Associations define two or more reference properties as Key qualified properties. The references identify the relationships that associations define between classes.

Example:

```cim
[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;
};
```
An association is a type of class that contains two or more references which are defined as Key qualified properties. 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 definition of an association has no effect on any of the classes its reference relate. An association cannot be a subclass of a non-association class.

**Example:**

```cim
[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;
}
```

Copyright © 2002-2006 Distributed Management Task Force, Inc. and WBEM Solutions, Inc. All rights reserved.
An indication is the active representation of the occurrence of an event. Indications are classes that have the Indication qualifier applied. Since an indication is a type of class, it can have properties and methods, and can be hierarchically defined. Instances of an indication are transient and can not be retrieved. Indications can only be received by having subscribed for them before they occur.

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

An Indication Subscription is expressed by the creation of a CIM_IndicationSubscription association instance that references a CIM_IndicationFilter (a Filter) instance, and a CIM_ListenerDestination (a Handler) instance. A Filter contains the query that selects an Indication class or classes. The Handler identifies where the indication is to be delivered when it occurs. 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)

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**

```cim
[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 CreationClassName,  
   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**

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

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

```plaintext
alias = AS aliasIdentifier

aliasIdentifier = "$" 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
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"
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).
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 comprised of a number of discrete elements. The CIM Schema supplies a set of classes and associations with properties and methods that provide a well-defined 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, properties and methods 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, associations, properties 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.
The core model establishes a basic classification of the classes 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 management domains and relationships between managed entities.

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

- Qualifiers
- Core Elements/Base Classes (e.g, ManagedElement, LogicalElement, System, Service, Dependency, Component, LogicalIdentity)
- PhysicalElements & Location
- SoftwareIdentity
- Devices
- StorageExtents (subclass of LogicalDevice)
- Collections
- Product and FRUs (Field Replaceable Unit)
- Statistics
- Capabilities
- Settings
- 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 transitonally) 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 related to 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 independent of the other) or existence dependencies (the object cannot exist independent of the other) between Managed Elements.
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, associations, properties 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 continually working on new Common Models which, over time, will help streamline the development process and add to the value proposition of CIM and WBEM. The Common Models for CIM Schema 2.10 are:

<table>
<thead>
<tr>
<th>Applications</th>
<th>Event</th>
<th>Network</th>
<th>Support</th>
</tr>
</thead>
<tbody>
<tr>
<td>Database</td>
<td>Interop</td>
<td>Physical</td>
<td>Systems</td>
</tr>
<tr>
<td>Devices</td>
<td>Metrics</td>
<td>Policy</td>
<td>User</td>
</tr>
</tbody>
</table>

Each of the Common Models are discussed in the following pages. It is important that developers working with CIM understand the data they wish to manage as well as gain an understanding of the various Common Information Models with similar concepts.

Developers are not required to take entire Common Models intact. It is expected that they will leverage the hierarchies applicable 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 defined higher in the class hierarchy. It is also valuable for developers to understand "rendering" and "infrastructure".

Copyright © 2002-2006 Distributed Management Task Force, Inc. and WBEM Solutions, Inc.
All rights reserved.
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 distributable 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
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:

```
Checks + actions  Checks + actions  Checks + actions
Deployable       Installable       Executable
```

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.
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 difficult 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.

<table>
<thead>
<tr>
<th>Applications</th>
<th>Event</th>
<th>Network</th>
<th>Support</th>
</tr>
</thead>
<tbody>
<tr>
<td>Database</td>
<td>Interop</td>
<td>Physical</td>
<td>Systems</td>
</tr>
<tr>
<td>Devices</td>
<td>Metrics</td>
<td>Policy</td>
<td>User</td>
</tr>
</tbody>
</table>
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 store 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).

<table>
<thead>
<tr>
<th>Applications</th>
<th>Event</th>
<th>Network</th>
<th>Support</th>
</tr>
</thead>
<tbody>
<tr>
<td>Database</td>
<td>Interop</td>
<td>Physical</td>
<td>Systems</td>
</tr>
<tr>
<td>Devices</td>
<td>Metrics</td>
<td>Policy</td>
<td>User</td>
</tr>
</tbody>
</table>

Copyright © 2002-2006 Distributed Management Task Force, Inc. and WBEM Solutions, Inc.

All rights reserved.
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

For this tutorial, the occurrence of an event is represented by an instance of an CIM_Indication subclass.

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

- CIM_InstIndication for modeling CIM life cycle events; instance creation, deletion, modification, method invocation and read access
- CIM_ClassIndication for CIM Schema life cycle events; class creation, deletion and modification.
- CIM_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.

The properties of CIM_Indication can be used by clients to perform sophisticated post processing computations performed of Indication streams. The IndicationIdentifier can be used to help distinguish between Indication instances. However, in this model, Indications are not guaranteed to be uniquely identifiable. For example, the IndicationTime property may have the same value if two Indications generated in rapid succession. The CorrelatedIndications property can be used to provide aggregation and correlation of Indications.

**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 CIM_IndicationFilter instances. A Subscription is expressed by the creation of an CIM_IndicationSubscription association instance that references an CIM_IndicationFilter (a Filter) instance, and an CIM_ListenerDestination (a Listener) 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 CIM_Indication subclass. A Handler subclass instance is used to specify the destination that is to receive the associated Indication stream. This version of the specification defines the CIM_ListenerDestinationCIMXML 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 CIM_IndicationHandler (rather than CIM_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 CIM_IndicationSubscription SHOULD fail. Likewise, if there is no instance of the requested CIM_IndicationHandler the instantiation of the CIM_IndicationSubscription SHOULD fail. The Modeling Events Section describes the properties and semantics of the CIM_Indication and CIM_IndicationSubscription class hierarchies.

**Namespace and Subscription Management**

Indications and their properties are to be interpreted in the context of a single namespace. The CIM_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 CIM_IndicationFilter and CIM_IndicationHandler 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 must be sufficiently extensible to allow schema designers to add new types of indications (events) in arbitrary ways reflecting unforeseen Indication structure and usage. Second, it must 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.

<table>
<thead>
<tr>
<th>Applications</th>
<th>Event</th>
<th>Network</th>
<th>Support</th>
</tr>
</thead>
<tbody>
<tr>
<td>Database</td>
<td>Interop</td>
<td>Physical</td>
<td>Systems</td>
</tr>
<tr>
<td>Devices</td>
<td>Metrics</td>
<td>Policy</td>
<td>User</td>
</tr>
</tbody>
</table>
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.

**WBEM Client**

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

**WBEM 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. 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.

<table>
<thead>
<tr>
<th>Applications</th>
<th>Event</th>
<th>Network</th>
<th>Support</th>
</tr>
</thead>
<tbody>
<tr>
<td>Database</td>
<td>Interop</td>
<td>Physical</td>
<td>Systems</td>
</tr>
<tr>
<td>Devices</td>
<td>Metrics</td>
<td>Policy</td>
<td>User</td>
</tr>
</tbody>
</table>
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_BaseMetricDefinition 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.
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.

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.
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 ChassisInRack 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.
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." 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 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, 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.
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.

---

**Table:**

<table>
<thead>
<tr>
<th>Applications</th>
<th>Event</th>
<th>Network</th>
<th>Support</th>
</tr>
</thead>
<tbody>
<tr>
<td>Database</td>
<td>Interop</td>
<td>Physical</td>
<td>Systems</td>
</tr>
<tr>
<td>Devices</td>
<td>Metrics</td>
<td>Policy</td>
<td>User</td>
</tr>
</tbody>
</table>
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:

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 PhysicalElements (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.

<table>
<thead>
<tr>
<th>Applications</th>
<th>Event</th>
<th>Network</th>
<th>Support</th>
</tr>
</thead>
<tbody>
<tr>
<td>Database</td>
<td>Interop</td>
<td>Physical</td>
<td>Systems</td>
</tr>
<tr>
<td>Devices</td>
<td>Metrics</td>
<td>Policy</td>
<td>User</td>
</tr>
</tbody>
</table>
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.
Utilizing extensions developers can leverage the basic model classes and associations to add their own model to address their own management needs. One of the many goals of CIM is that it is a much broader and cleaner modeling language for capturing management characteristics.

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 new 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 supertype 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. An inherited method can be overridden.
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 ask the following question of 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 asked 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 is 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 developers 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 designing a schema.

When designing a schema, keep in mind that you are creating a language with which to communicate the structure of information (both physical and logical elements) within a managed environment. This structure will grow over time and will be interpreted by its various consumers. 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 is 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 of the same class that refers to the same object in the environment. It is quite acceptable to have instances of 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 to which they apply. 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.

```
Manufacturer  1 Supports * Product
               1 Supplied by *
```

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, the product must also be supported by that manufacturer 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 arise 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 CIM 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. 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 contain instances.** If you end up with instances that contain 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.
Web-Based Enterprise Management (WBEM) is a set of management and Internet standard technologies developed to unify the management of distributed computing environments. WBEM provides the ability to exchange CIM information in an interoperable and efficient manner. WBEM includes protocols, query languages, discovery mechanisms, mappings, and anything else needed to exchange CIM information.

The DMTF technologies are distinctly isolated at an architectural level to provide for maximum flexibility. The CIM defines the syntax that is used to define the structure of management information. The CIM Schema builds upon CIM to provide a robust set of common management information. WBEM provides an interoperable and extensible means of managing CIM information; the set of standards defined in the WBEM Initiative, when used in conjunction with CIM provide the information management infrastructure. Taken separately, each of these DMTF technologies is interesting in and of themselves, but only when they used combined do they provide a powerful end-to-end enterprise management solution.

This section describes the following WBEM specifications. The items listed as "under development" are included for informational purposes. As the standards evolve in these areas, descriptions of them will be added to this tutorial.

Mappings

URI
- WBEM URI Mapping Specification 1.0, DSP0207

XML
- Representation of CIM using XML 1.2, DSP0201
- CIM DTD 1.2, DSP0203

XML Schema
(under development)

Protocols

CIM-XML
- CIM Operations over HTTP 1.2, DSP0200

CLP
- Command Line Protocol 1.0, DSP0214

WSDM
(under development)

WS-Management
(under development)

Discovery
Query Language

CQL
CIM Query Language 1.0, DSP0202
The WBEM URI Mapping Specification (DSP0207) defines the Universal Resource Identifier (URI) format for WBEM protocols. A WBEM URI is a compact string of characters that identify a CIM Element. This specification defines the mapping of CIM Naming as defined in the CIM Specification to the URI syntax.

The WBEM URI is used by WBEM protocols as the method for identifying CIM Elements.

The components that compose a WBEM URI are defined below. Optional components of the URI are in brackets (“[]”). The comma (","), is used to explicitly designate concatenation of components with all intervening white space removed.

The Backus-Naur Form (BNF) productions shown are expressed using the Augmented BNF (ABNF) with the following exceptions:

- A pipe ("|"), rather than a slash ("/") is used to represent choices.
- A comma (",","), is used to explicitly designate concatenation of rules rather then implicitly assuming concatenation.

Unless explicitly stated to the contrary, the productions defined in this specification do not allow whitespace between terms.

NOTE: This version of the WBEM URI specification defines the following types of encoding:

- legacy CIM Object Path encoding.
- new typed CIM Object Path encoding.

**WBEM URI Components**

```
<wbem-uri> = [<namespace-type>"."](<net-path>|<abs-path>)
```

The components that compose the WBEM URI above are defined as follows:

- `<net-path> = "/",<server>[<abs-path>]`
- `<abs-path> = "/",<path_segments>`
- `<path_segments> = [<namespace-name>],[<model-path>|<typed-model-path>]`

Specifications that refer to the WBEM URI Specification must define the required components.

**Examples:**
<table>
<thead>
<tr>
<th>IETF URI</th>
<th>Scheme Type</th>
<th>Server</th>
<th>Path Segment</th>
<th>WBEM URI Object Path Encoding</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Object Path</strong></td>
<td><strong>Namespace Type</strong></td>
<td><strong>Server</strong></td>
<td><strong>Namespace Name</strong></td>
<td><strong>Instance Definition Qualifying Class</strong></td>
</tr>
<tr>
<td>Example</td>
<td><a href="https://jdd.test@aome.com:5959">https://jdd.test@aome.com:5959</a></td>
<td>cimv2</td>
<td>test</td>
<td>CIM_RegisteredProfile</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th><strong>Typed Object Path</strong></th>
<th><strong>Namespace Type</strong></th>
<th><strong>Server</strong></th>
<th><strong>Namespace Name</strong></th>
<th><strong>Instance Definition Qualifying Class</strong></th>
<th><strong>Typed Key-Value Pair</strong></th>
</tr>
</thead>
<tbody>
<tr>
<td>Example</td>
<td><a href="https://jdd.test@aome.com:5959">https://jdd.test@aome.com:5959</a></td>
<td>cimv2</td>
<td>test</td>
<td>CIM_RegisteredProfile</td>
<td>InstanceID: &quot;aome-1&quot;</td>
</tr>
</tbody>
</table>

Copyright © 2002-2006 Distributed Management Task Force, Inc. and WBEM Solutions, Inc.
All rights reserved.
The mapping of CIM to XML (also known as, xmlCIM) is defined in the following specifications:

- Representation of CIM using XML (DSP0201)
- CIM DTD (DSP0203)

NOTE: The DMTF is currently defining a mapping of CIM to XML Schema.

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.

xmlCIM is designed as a meta-schema mapping, which means that each CIM Element, can be represented in XML. xmlCIM can be used both to represent CIM Declarations (Classes, Instances, and QualifierTypes) and the CIM Messages used by the CIM protocols. A Document Type Definition (DTD) is a document that defines the structure in which an XML document is created. The DMTF has provided a CIM DTD to allow XML documents to be verified as conforming to the DTD.

An example of a CIM Declaration and a CIM Message follow.

**CIM Declaration Example**

```xml
<INSTANCE CLASSNAME="CIM_Namespace">
  <PROPERTY PROPAGATED="false" NAME="Name" TYPE="string">
    <VALUE>interop</VALUE>
  </PROPERTY>
  <PROPERTY PROPAGATED="false" NAME="CreationClassName" TYPE="string">
    <VALUE>CIM_Namespace</VALUE>
  </PROPERTY>
</INSTANCE>
```

**CIM Message Example**

```xml
<?xml version="1.0" encoding="utf-8" ?>
<CIM CIMVERSION="2.3" DTDVERSION="2.2">
  <MESSAGE ID="xxx" PROTOCOLVERSION="1.0">
    <SIMPLEREQ>
      <IMETHODCALL NAME="GetClass">
        <LOCALNAMESPACEPATH>
          <NAMESPACE NAME="interop"/>
        </LOCALNAMESPACEPATH>
        <IPARAMVALUE NAME="ClassName">
        </IPARAMVALUE>
      </IMETHODCALL>
    </SIMPLEREQ>
  </MESSAGE>
</CIM>
```
<CLASSNAME NAME="CIM_Namespace"/>
</IPARAMVALUE>
<IPARAMVALUE NAME="LocalOnly">
  <VALUE>FALSE</VALUE>
</IPARAMVALUE>
</IMETHODCALL>
</SIMPLEREQ>
</MESSAGE>
</CIM>
CIM-XML is a protocol for exchanging CIM information. CIM-XML has the following components:

1. Common Information Model (CIM)
2. xmlCIM encoding
3. set of operations to retrieve and manipulate CIM data
4. HTTP encapsulation

CIM-XML uses xmlCIM as the payload and HTTP as the transport. CIM-XML defines all interactions between management clients and management infrastructure as CIM messages, which are well-defined request or response data packets used to exchange information. The two types of *CIM Messages* are as follows:

- A *CIM operation message* is a CIM message that is used to invoke an operation.
- A *CIM export message* is a CIM message that is 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.

A XML Document Type Definition (DTD) is a document that sets the rules for how the XML is structured. CIM-XML has defined the CIM-XML DTD. The XML payload of CIM messages is *loosely valid*. The term *loosely valid* applies to an XML Document as follows:

- 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 that is valid with respect to the CIM XML Document Type Definition (DTD) apart from having additional attributes or elements that are not defined in that DTD. The concept is very similar to that of an open content model. 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 it is parsing received XML documents that are defined within the scope of this mapping. Not all CIM Clients (CIM Servers or CIM Listeners) should be required to validate each received CIM Message Response (CIM Message Request) because 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, this specification makes the following requirements:

- 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 (inclusion inline 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 is either:

- **Intrinsic**, which means that it is defined by this specification for the purposes of modeling a CIM operation
- **Extrinsic**, which means that it is defined as a method on a CIM class in some Schema.
The Command Line Protocol (CLP) is a human-oriented command-line protocol that is suitable for scripting management environments. CLP defines:

- a mapping to a subset of the CIM Schema
- a command syntax and grammar
- output syntax format
- session protocol
- transport protocols

The CLP is a command/response protocol (and not a command-line interface). Management requests are initiated by a CLP Client and are transmitted through a text-message-based transport protocol on the wire (for example, Telnet or SSHv2) to a Management Access Point (MAP). The MAP hosts a CLP Service which processes received commands and returns an appropriately formatted response to the requesting Client.

NOTE: Although the CLP currently defines mappings to Telnet and SSHv2, any transport capable of carrying command/response messages can also be used.

The CLP supports internationalization in that a Client can specify a desired language in the commands that it issues to a MAP. If the MAP may supports the requested language, output can be represented in the response to the requesting Client in the appropriate language translation.

The CLP syntax is extensible through its four supported mechanisms. The general CLP syntax is as follows:

```
<Verb> [<Options>] [<Target>] [<Properties>]
```

- Verb is the specific command or action being performed. These verbs enable retrieving and managing data ("set", "show"), creating and deleting data ("create", "delete"), modifying the state of a managed element ("set", "reset", "start", "stop"), managing the current session ("cd", "version", "exit") and providing command information ("help").
- Options are command options that modify the action or behaviour of the Verb. Options enable changing output format, applying the command to nested levels and requesting help or version information.
- Target is the address or path of the target of the issued command. The format of this address or path is defined in the *SM Managed Element Addressing Specification*. Only one target specified per issued command, but the Target can refer to an individual managed element (for example, a disk or NIC) or a collection of managed elements.
• Properties are the attributes of the Target that might contain values that are needed to process the command. Properties identify the properties of the Target's class that are to be retrieved or acted upon by the command. (that is, these are properties defined in the CIM class)

The following examples show CLP syntax. The first example demonstrates how to retrieve the speed and port type of a given Ethernet port. The second example demonstrates how to modify the current Ethernet port speed.

    show -display properties=speed,porttype /hostABC/ethernet1

    set /hostABC/ethernet1 speed=1000

NOTE: The CLP syntax is not case sensitive.
WBEM Discovery is any mechanism that allows the discovery of the location of WBEM Servers. Although WBEM has already defined one discovery specification, this does not preclude additional discover mechanisms from being added in the future.

**WBEM Discovery using the Service Location Protocol (SLP)**

WBEM Discovery using SLP is defined in two specifications:

1. WBEM Discovery using The SLP ([DSP0205](#))
2. WBEM SLP Template ([DSP0206](#))

**Objectives**

- Provide a mechanism that allows WBEM Clients to discover WBEM Servers
- Use existing standards and protocols for rapid development and deployment
- Provide a mechanism that scales from small environments to enterprise environments
- Provide WBEM Clients sufficient information in the advertisement related to management capabilities to determine the WBEM Servers of interest
- Scope the level of advertisement to avoid security holes

**SLP**

WBEM Discovery using SLP requires Service Location Protocol (SLP) version 2.0. The SLP is defined by the Internet Engineering Task Force (IETF) in [RFC 2608](#).

The SLP provides a flexible and scalable framework for providing clients, represented by User Agents (UA), with access to information about the existence, location, and configuration of services, represented by Service Agents (SA).

Traditionally, clients have had to know the name and access method of services. The SLP eliminates this requirement. With SLP, the client requests a type of service that contains information regarding the requested services.

The SLP uses Directory Agents (DA) that offer a centralized repository for advertised services. This feature enables the SLP to scale from very small to very large environments.

WBEM Servers acting as Service Agents (SA) advertise their services. WBEM Clients acting as User Agents (UA) query for the WBEM Server(s). A Directory Agent (DA) might be deployed in environments where there are many User Agents and Service Agents exist.

For each service access point (protocol/port) supported, the WBEM Server must supply an advertisement. For example, if a WBEM Server supports CLP and CIM-XML, it would supply at least two advertisements. Note that the WBEM Server would supply more than two advertisements if multiple ports were supported by the supported protocols. For example, it would supply four advertisements if it supported CLP over Telnet and SSH and CIM-XML over
HTTP and HTTPS.

**WBEM SLP Template**

The information that can be discovered is defined in the WBEM SLP Template. The following attributes are defined in this template:

- template-type=wbem
- template-version=1.0
- template-description=This template describes the attributes used for advertising WBEM Servers.
- template-url-syntax
  The WBEM URI that addresses the WBEM Server.
  Example: template-url-syntax=https://192.168.0.1:5989
- service-hi-name
  This string is used as a name of the WBEM Service for human interfaces.
- service-hi-description
  A description of the WBEM Service.
- service-id
  The ID of this WBEM Server.
- CommunicationMechanism
  The protocol supported by the WBEM Server.
- OtherCommunicationMechanismDescription
  The description of the protocol supported if it is not standard.
- InteropSchemaNamespace
  The namespace within the target WBEM Server where the CIM Interop Schema is implemented.
- ProtocolVersion
  The version of the protocol.
- FunctionalProfilesSupported
  The operations supported by the WBEM Server.
- FunctionalProfileDescriptions
  Descriptions of the operations supported by the WBEM Server.
- MultipleOperationsSupported
  This specifies whether the WBEM Server supports batch operations.
- AuthenticationMechanismsSupported
  The authentication mechanism supported by the WBEM Server.
- AuthenticationMechanismDescriptions
  Descriptions of the authentication mechanisms supported by the WBEM Server.
- Namespace
  The namespace(s) supported by the WBEM Server.
- RegisteredProfilesSupported
  The Profile(s) supported by the WBEM Server.

Examples:
CIM Query Language

This tutorial describes the basic syntax and features of the DMTF CIM Query Language (CQL), which can be very complex. For detailed information on CQL refer to the CIM Query Language Specification, defined in DSP0202. This specification is preliminary and currently available for public review.

The DMTF CIM Query Language (CQL) provides the capability to select properties from sets of CIM instances. CQL support is subdivided into a number of independent features:

- Basic Query (required)
- Simple Join
- Complex Join
- Subquery
- Result Set Operations
- Extended Select List
- Embedded Properties

- Aggregations
- Regular Expression Like
- Array Range
- Satisfies Array
- Foreign Namespace Support
- Arithmetic Expression

CQL features are advertised through a property in an instance of CIM_QueryCapabilities.

CQL queries are expressed as a statement in the following form:

```
SELECT <items> FROM <this> [ WHERE <conditions> ] [ ORDERED BY <sort specification> ]
```

**SELECT**

SELECT defines the columns of the table that results from the SELECT in its basic form (for example, the properties of the CIM Instance data to be returned as part of the result set).

**FROM**

FROM specifies the CIM Element from which to retrieve the data. For example, FROM could specify the name of one CIM class. All instances of this class and its subclasses are candidates for inclusion in the information that is returned as part of the result set.

**WHERE**
WHERE acts as a filter. The result set will not include items that do not match the set of conditions.

**ORDERED BY**

ORDERED BY sorts the results in the specified order.

**Examples**

The following examples describe some of the ways in which CQL can be used.

- Get all StorageExtent and MediaAccessDevice instances. Note that the projection is limited to instances that are either CIM_StorageExtent or CIM_MediaAccessDevice, however only properties of CIM_LogicalDevice and its superclasses are returned.

  Required feature: Basic Query

  ```cql
  SELECT * FROM CIM_LogicalDevice WHERE CIM_LogicalDevice ISA CIM_StorageExtent OR CIM_LogicalDevice ISA CIM_MediaAccessDevice
  ```

- Get the object path, ElementName and Caption for all StorageExtents

  Required features: Basic Query, Extended Select List

  ```cql
  SELECT OBJECTPATH(CIM_StorageExtent) AS Path, ElementName, Caption FROM CIM_StorageExtent
  ```

  A set of instances would be returned, each containing the object path that identify the name of the instance, the ElementName and Caption properties.

- Select all LogicalDevices on a particular ComputerSystem that have an OperationalStatus not equal to “OK” (value = 2), and return their object paths and OperationalStatus.

  Required features: Basic Query, Extended Select List, Complex Join, Array Range

  ```cql
  ```

  A set of instances would be returned, each containing the object path identifying the name of the instance of CIM_LogicalDevice on the specified CIM_ComputerSystem and the OperationalStatus property.

- List all ComputerSystems and the object paths of any instances dependent on the system as described by the Dependency association.

  Required Features: Basic Query, Extended Select List, Complex Join

  ```cql
  SELECT CIM_ComputerSystem.*, OBJECTPATH(CIM_ManagedElement) AS MEObjectName FROM CIM_ComputerSystem, CIM_ManagedElement,
  ```
CIM_Dependency
WHERE CIM_Dependency.Antecedent = OBJECTPATH(CIM_ComputerSystem)
AND CIM_Dependency.Dependent = OBJECTPATH(CIM_ManagedElement)

This query returns a set of instances that are defined by the references of the
Dependency association's instances. The instances that are returned contain all the
properties of CIM_ComputerSystem and a string representing the associated
ManagedElement's object path.

Copyright © 2002-2006 Distributed Management Task Force, Inc. and WBEM Solutions, Inc.
All rights reserved.
A DMTF Management Profile is a specification that defines the CIM model and associated behavior for a management domain. The CIM model includes the CIM classes, associations, indications, methods and properties. The management domain is a set of related management tasks.

The CIM Schema defines a vast set of management information. Profile specifications use a small subset of the full CIM Schema and define rigid requirements regarding the classes and associations used to represent the management information in a given management domain, as well as the properties, methods and indications of those classes that must be supported and also the behavior of how the model must be implemented for a specific management domain. Additionally, DMTF Management Profiles might require a specific value or set of values to be populated for a particular property of a class, the format of property values and the cardinality requirements of the associations that define the relationships in the model. These requirements enable management applications to consume a known set of information with a well-defined format, meaning and behaviour in a cross-platform, interoperable manner.

DMTF Management Profiles are defined in the Management Profile Specification Usage Guide (DSP1001). The DMTF also defines the related DMTF Management Profile Template (DSP1000) to accelerate the definition of new Management Profiles to address management domains of industry-wide interest.

DMTF Management Profiles provide solutions for both end-users and vendors. End-users can determine which Profiles support the management tasks that they want to leverage and include them in a Request For Quote or requirement document. Vendors can use a Profile specification as a high-level functional specification for the management capabilities they want to support in their products. Vendors can also create vendor-specific (that is, vendor extension) Profiles that define added-value management capabilities or interoperability with different groups inside a company, other companies or customers.

The different types of Management Profiles are as follows:

- **An Autonomous Profile** defines an autonomous and self-contained management domain. This type includes Profiles that are stand-alone or have relationships to other Profiles.

- **A Component Profile** describes a subset of a management domain. A Component Profile includes CIM Elements that are scoped within an Autonomous Profile or in rare cases, another Component Profile. Multiple Autonomous Profiles can and often do reference the same Component Profile, enables the encapsulation of a management domain that is applicable in multiple management contexts. For example, network port management is applicable to servers, switches, routers and storage devices.

- **A Specialized Profile** is based on and constrains another Profile specification. For
example Fibre Channel, Ethernet and iSCSI port management require more specialized information than general network port management, but build upon a common set of management information.

An **Abstract Profiles** specifies common elements and behavior that form the base for Specialized Profiles. Abstract Profiles are templates and cannot be implemented.
The *(DSP1000)* DMTF Management Profile Template is located on the DMTF website. The template sections and their intended usage and content are explained in detail in the *Management Profile Specification Usage Guide* (DSP1001).

The Management Profile Template captures a Profile definition in the following format:

1. **Synopsis**
   
   *Profile name*: The name of the Profile  
   *Version*: The version number of the Profile using the formant "M.N.U" (for example, "1.0.0")  
   *Organization*: The organization publishing the Profile. (for example, "DMTF")  
   *CIM schema version*: The earliest version of the CIM Schema upon which an implementation of the Profile must be based. (for example, "2.10.0")  
   *Central Class*: The central class/focal point of the Profile. This defines the expect client entry point for discovering management information for a device that supports the Profile. (for example, CIM_ComputerSystem is the entry point for discovering management information about a Server)  
   *Scoping Class*: The scoping class for this Profile  
   *Short Description*  
   *Table of related Profiles*

   **NOTE**: The Profile name, version and organization uniquely identify a Profile specification.

2. **Description**
   
   The Description section describes the management domain implemented by a Profile and provides an overview of the model. The Description does not include any specification requirements, but rather describes how the classes of the Profile relate to the management domain. This section will typically contain class diagrams that depict the model.

3. **Implementation**
   
   The Implementation section contains requirements of the model that are not covered by other sections and guidelines related to implementation. This section might include requirements that are referenced from other sections, and describe the relationship between the model and underlying instrumentation.

4. **Methods**
   
   The Methods section provides a list of the Extrinsic Methods of the class in the Profile that are supported by this Profile. Profile usage of both Extrinsic Methods and Generic Operations (for example, intrinsic methods) are included, but the specification formats are different.

5. **Use Cases**
The Use Cases section specifies use cases that demonstrate interesting behaviors or tasks that the Profile supports. The purpose of a use case is to illustrate the steps required to accomplish some goal and the effects on the model of accomplishing that goal. A use case defines the interaction of a client and a WBEM Server in the execution of steps that are required to realize the functionality described in the Profile. Use cases typically represent a complete task from the perspective of a client, which might involve multiple CIM operations or methods. A Profile specification can document one or more use cases, each of which has different starting conditions and ending conditions. The purpose of the use case is to illustrate the steps required to accomplish some goal and the effects on the model in the course of accomplishing that goal.

6. CIM Elements
The CIM Elements section contains the following items:
- An overview subclause that consists of a table that lists the Profile's classes, indications, and queries.
- A subclause for each class that includes a short description of the class and a table that includes the Profile's use of properties and methods.

The table (and a subclause) include all of the classes and associations that are defined in the Profile. This section can include constraints on properties, such that a property must contain a specific value or a specific format for a value, or that a value may change based on the condition of something else.
A Management Initiative is an industry effort to address an existing need for additional capabilities and improvements (that is, Initiative Subject) in distributed enterprise management particular to a specific vertical market. For such an effort to gain traction and buy-in from the broadest possible set of stakeholders in a given vertical market, multiple objectives must simultaneously be addressed. These objectives include:

- Identifying and organizing a group of technical and marketing experts to define the Initiative Subject to be addressed in the vertical market.
- Soliciting consumer feedback on the scope and prioritization of the areas within the Initiative Subject to be addressed by the Initiative.
- Building consensus internally within the vendor community that a distributed standards-based solution has a critical mass.
- Identifying existing standards that can be leveraged in defining a solution to the Initiative Subject.
- Determining new standards that need to be defined to ensure that a distributed solution to the Initiative Subject can be achieved.
- Defining the standards that address the Initiative Subject.
- Developing products that validate the distributed, interoperable goals of the standards.
- Building a brand for the resulting standards.
- Testing and certifying vendor conformance to the standards.
- Deploying real-world products in the market.
- Educating the press, analysts and customers about the value that the Initiative provides.

Two existing Management Initiatives are detailed in this tutorial. The first is the Systems Management Architecture for Server Hardware (SMASH) Initiative which is an effort within the DMTF to improve the interoperable management of servers and systems management. The second is the Storage Management Initiative (SMI) which is an effort in the Storage Network Industry Association (SNIA) to improve the interoperable management of storage networks and hardware.

NOTE: Management Initiatives evolve within the DMTF and within external standards organizations. Of fundamental importance is that a group of cross-industry experts participate in the Initiative and that the result of the effort promotes standards-based distributed management that accelerates the adoption of interoperable solutions within a market.
The Systems Management Architecture for Server Hardware (SMASH) is an initiative that represents a suite of specifications which standardize the manageability interfaces for server hardware. Particularly, the need for a command-line protocol to accomplish interoperable distributed server management was a driving force in the server market that led to the definition of several related standards.

To address the need for improved server management capabilities, the DMTF chartered the Server Management Working Group (SMWG) in December 2003 to oversee the definition of the necessary server management standards. The SMWG membership consists of individuals from industry-leading server vendors, chip manufacturers, operating system vendors, and firmware and software developers as well as end-users. In total the SMWG consists of more than 200 individuals from more than 40 companies.

In September 2004, the SMWG announced the launch of the SMASH Initiative to drive the definition of extensible standards to improve server management. The SMASH Initiative specifically addresses defining an industry-standard command-line interface to enable CIM-based management of heterogeneous server environments independent of machine state, operating system state and device access mechanisms.

To date, the SMWG has delivered the following materials:

- Server Management Command Line Protocol Architecture Whitepaper ([DSP2001](#))
- Server Management Command Line Protocol Specification (SM CLP) ([DSP0214](#))
- Server Management Managed Element Addressing Specification (SM ME) ([DSP0215](#))
- Profile Specifications
- CLP-CIM Mapping Specification for each Profile Specification that is referenced by SMASH.

The constituent standards defined as part of the SMASH Initiative address the following functional requirements:

- Unified Management to enable discovery, enumeration and aggregation of server management information.
- Visibility to enable the discovery of server hardware and topology, hardware status (for example, processors, memory, power supplies, fans and ports), and operating system and boot process status.
- System Control to enable power cycle management, system reset, LED/LCD panel manipulation and alarms, operating system shutdown, and the definition of common conventions and access mechanisms for trouble-shooting hung and crashed systems.
- Firmware Management to enable the discovery, deployment and updating firmware.
- Provisioning to enable boot process control, system configuration, and common mechanisms for identification and execution of operating system images.
Following is a high-level summary of the Management Profiles that are referenced in the SMASH Initiative:

**Autonomous Profiles:**

*CLP Service Profile* defines a management access point for human or scripted clients to manage servers in the CLP's Admin Domain.

*Base Server Profile* defines the minimum information related to a simple stand-alone server. (that is, in a single chassis form factor) The various other Profiles outlined in this section provide added information and management capabilities.

*Modular System Profile* defines the ability to discover logical and physical information for blade servers and manage their state.

**Component Profiles:**

*Authentication Profile* defines the capability to authenticate an identity

*Boot Control Profile* defines the capability to control boot configuration on a system. The boot configuration information includes boot sources, boot paths and boot order and includes a general information structure for support of specific boot settings.

*Chassis Manager Profile* defines the management of a chassis in modular systems. Note that this Profile also provides for redundant chassis management.

*CLP Admin Domain Profile* defines the ability to identify the collection of server hardware within the management domain of a CLP service.

*CPU Profile* defines the the capability to represent CPUs or processors in a managed system

*Device Tray Profile* defines the management of a device tray. A device tray acts as an aggregation point for media devices in a modular system or rack configuration. This Profile enables managing the tray as a whole.

*DHCP Client Profile* defines the ability to discover the management capabilities and configuration of a server that is acting as a DHCP client.
DNS Client Profile defines the ability to discover the management capabilities and configuration of a server that is acting as a DNS client.

Ethernet Port Profile defines a specialization to the LAN Network Port Profile that provides the ability to discover Ethernet ports and related hardware as well as active Ethernet interfaces in a server. The ethernets in which the ports participate are explicitly NOT addressed.

Fan Profile defines the ability to discover fan information, relationships to physical devices in a server and related sensors. Note that fan redundancy may also be supported.

Firmware Update Profile defines the management of firmware image updates to the devices in a system. Firmware is loaded using a Universal Resource Identifier (URI). The file type of the firmware image and its format are agnostic, which enables a wide range of programmable devices and persistent memory to be updated.

Firmware Inventory Profile defines the ability to identify the firmware versions installed on a system and the devices to which they apply.

IP Interface Profile defines the ability to discover and manage IPv4 interface and configuration (current and pending) information in a server, and optionally the relationships between the IP interfaces and DHCP and DNS clients. This Profile is specifically not intended to support network gateway or IPv6 management needs.

LAN Network Port Profile defines the ability to discover network port, physical connector and optionally related controller and network interface information on a system.

Pass-thru Module Profile defines the ability to discover and manage the devices that replace physical cables in a server to allow internal network physical ports in a chassis or rack readily accessible from an external network.

Physical Asset Profile defines the ability to discover physical asset inventory information in a server and the physical topology of those assets. This Profile is flexible in that the physical asset information is extensible to the supported Profiles that include physical (that is, rather than logical) device information.

Power Supply Profile defines the ability to discover and manage power supply information in a server. This Profile also provides for redundant power supply management.

Profile Registration Profile defines the ability to discover the registration and versioning of supported Profiles.

Record Log Profile defines the ability to discover and manage the logs and the records within those logs on a server.

Role-Based Authorization Profile defines the capability to authorize based on roles.

Sensor Profile defines the management information related to system sensors and describes the relationship of the sensors to the devices that they monitor.

Server Power State Management Profile defines the behavior of a power management service that enables control of the system power state and hardware reset management.
Shared Device Management Profile defines the management of the shared devices of a modular system.

SMASH Collections Profile - defines the ability to discover and create collections of managed elements. These collections can include the various capabilities, text consoles, logs or settings related to the server.

SSH Service Profile defines the ability to discover and manage the SSH service, related current and default configurations and active sessions on a server.

System Memory Profile defines the memory devices associated with a system.

Telnet Service Profile defines the ability to discover and manage the telnet service, related current and default configurations and active sessions on a server.

Text Console Redirection Profile defines the management of the system text console redirection. Typically the text console would be sent to a local serial port to which a terminal is connected. Redirection of the text console minimizes the number of physical connections or provides remote access to the console text through a network connection.
The Storage Network Industry Association (SNIA) was formed in 1997 to accelerate adoption of storage network technology. The SNIA membership includes a wide range of software and device vendors covering all areas of storage technology. The SNIA actively promotes expanding the breadth, depth and quality of the storage network market through its definition of management standards, educational offerings and its interoperability testing lab.

The SNIA is the governing body of the Storage Management Initiative (SMI). The Storage Management Initiative Specification (SMI-S) is an ANSI standard, defined by the SMI and based on the DMTF CIM and WBEM standards, that provides storage vendors with a common information model and management interface that enables interoperable management of heterogeneous storage network environments.

The SMI-S defines Management Profiles for various components of storage networks including RAID Disk Arrays, Fabrics, SAN Switches, HBAs, Storage Libraries, Self-contained NAS and NAS Head systems, Storage Virtualizers and Volume Managers. Additional management capabilities for these devices are defined in Management Subprofiles that address firmware installation, zoning configuration, data backup and restore, logical unit management, asynchronous event notification, target and initiator ports, masking and mapping, and device statistics.

Additionally, the SNIA hosts and operates the SMI-Lab which coordinates development and testing efforts at its headquarters in Colorado Springs, CO. To ensure that conformant implementations of the SMI-S reach end-users, SNIA defined the SMI Conformance Test Program (SMI CTP) to validate storage device and management application vendors' conformance to the standard.

For more information about the SNIA and SMI, please refer to the SMI Tutorials at

- SMI Technical Tutorial
- SMI Marketing Tutorial

Copyright © 2002-2006 Distributed Management Task Force, Inc. and WBEM Solutions, Inc. All rights reserved.
The goal of this section is to introduce the Distributed Management Task Force (DMTF), the DMTF organizational structure and the DMTF processes. This section also introduces the membership levels and the advantages and value of membership. Finally, it describes how member and non-member companies can get involved, provide feedback, and ask for help.

**DMTF Introduction**

With more than 3,500 active participants, the Distributed Management Task Force, Inc. (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. DMTF standards provide common management infrastructure components for instrumentation, control, and communication in a platform-independent and technology-neutral way. DMTF standards include the Common Information Model (CIM), communication/control protocols such as Web-Based Enterprise Management (WBEM), and the Systems Management Architecture for Server Hardware (SMASH) Initiative. Working with key technology vendors and affiliated standards groups, the DMTF is enabling a more integrated, cost-effective approach to management through interoperable management solutions.

For more information about DMTF standards and activities, go to [www.dmtf.org](http://www.dmtf.org).
DMTF Organization

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

Copyright © 2002-2006 Distributed Management Task Force, Inc. and WBEM Solutions, Inc.
All rights reserved.
DMTF Work Groups and Committees

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.

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), Web-Based Enterprise Management (WBEM), Management Profiles and Management Initiatives (i.e. SMASH, CDM). The Technical Committee oversees many working groups, select the following link to see the current list of Work Groups:

Work Groups

Interoperability Committee

The Interoperability Committee supplements the resources of the DMTF such that multi-vendor implementations of DMTF technologies can be compatible in the industry.

Marketing Committee

The Marketing Committee communicates with the industry, the public, and members about the activities of the organization.

Copyright © 2002-2006 Distributed Management Task Force, Inc. and WBEM Solutions, Inc. All rights reserved.
Every organization should have a set of operating policies to use as a guide to running its operations. The DMTF is no exception, and it has made public the information about the technology adoption process, how to gain access to the various DMTF committees, and policies regarding privacy and the distribution of other information. This page briefly describes some of these policies.

**Bylaws**

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

**Committee / Information Access**

This document provides an overview of DMTF committees and organization, work groups, the voting process, and access policy.

**Patent Policy**

This policy provides instructions and rules that dictate the submission of patent considerations to the DMTF.

**Press Release Policy**

This policy regards joint press releases and supporting quotes, what may be requested, requirements and approval processes.

**Privacy Policy**

This document outlines the DMTF policy on the collection of personal data, domain information, cookies, security, and disclosure.

**Release Process**

This document details the process of releasing DMTF specifications and schema. The release process has five phases, from development, comment, review, and preliminary standard and final standard. This document also details the numbering and errata methods for DMTF documents.

**Member Rules of Conduct**

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

**Technology Adoption Policy**

This policy outlines the rules for adopting new technology including instructions for submitting of new technology, possible courses of action and press and resubmit options.

For additional information about any of the preceding policies, see the [Operating Policies](#) page on the DMTF web site. If you have questions about DMTF policies, contact dmtf-info@dmtf.org.
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.

<table>
<thead>
<tr>
<th>Membership Level</th>
<th>Annual Dues</th>
<th>Voting Rights</th>
<th>Closed/Open</th>
<th>Work Group Participation</th>
<th>Marketing Activities</th>
</tr>
</thead>
<tbody>
<tr>
<td>Board Member</td>
<td>$30,000</td>
<td>Full</td>
<td>Closed (Elected)</td>
<td>Yes (unlimited)</td>
<td>Yes</td>
</tr>
<tr>
<td>Leadership Member</td>
<td>$12,000</td>
<td>Limited</td>
<td>Open</td>
<td>Yes (unlimited)</td>
<td>Yes</td>
</tr>
<tr>
<td>Leadership Member (End User/Government)</td>
<td>$6,000</td>
<td>Limited</td>
<td>Open</td>
<td>Yes (unlimited)</td>
<td>Yes</td>
</tr>
<tr>
<td>Participation Member</td>
<td>$6,000</td>
<td>Limited</td>
<td>Open</td>
<td>Yes</td>
<td>Yes</td>
</tr>
<tr>
<td>Participation Member (End User/Government)</td>
<td>$3,000</td>
<td>Limited</td>
<td>Open</td>
<td>Yes</td>
<td>Yes</td>
</tr>
<tr>
<td>Monitoring Member</td>
<td>$2,500</td>
<td>None</td>
<td>Open</td>
<td>No</td>
<td>Yes</td>
</tr>
<tr>
<td>Sponsored Member</td>
<td>$200</td>
<td>Limited</td>
<td>Open</td>
<td>Yes</td>
<td>Yes</td>
</tr>
<tr>
<td>Alliance Partner Member</td>
<td>No</td>
<td>None</td>
<td>Closed (not-for-profit standards organizations)</td>
<td>Yes (determined by DMTF board)</td>
<td>Yes</td>
</tr>
<tr>
<td>Academic Alliance Member</td>
<td>No</td>
<td>None</td>
<td>Closed (accredited institutions of higher learning)</td>
<td>Yes (one work group)</td>
<td>Yes</td>
</tr>
</tbody>
</table>

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.
The DMTF is leading the development, adoption, and unification of management standards and initiatives for desktop, enterprise and Internet environments. This page lists some of the approved specifications that the DMTF has made public.

**Common Information Model (CIM)**

CIM 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)**

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

**Systems Management Architecture for Server Hardware (SMASH) Initiative**

The DMTF SMASH Initiative is a suite of specifications that deliver architectural semantics, industry standard protocols and profiles to unify the systems management of the data center.

**Alert Standard Format (ASF)**

The ASF 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.
DMTF Tutorial > DMTF > Contact Information

DMTF Contact Information

Organization | Work Groups | Policies | Membership | Standards | Contact Information

Technical

For questions or comments about the interpretation of DMTF technologies or about the DMTF website, use the "Contact Us" form at http://www.dmtf.org/about/contact/.

Public Relations

For information about DMTF public relations activities, or if you are an industry journalist or analyst and need specific information about the DMTF or DMTF technologies, contact the DMTF’s public relations firm at press@dmtf.org.

General Questions

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

Copyright © 2002-2006 Distributed Management Task Force, Inc. and WBEM Solutions, Inc. All rights reserved.
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 class or instance declaration.

application life cycle
The concept of applications that are 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.

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.

- B -

Backus-Naur Form (BNF)
A metalanguage that specifies the syntax of programming languages.
cardinality
The number of instances of a class related that are to another class's instance through an association. For example, if an association relates Class_A to Class_B, then A's cardinality indicates how many instances of Class_A can be associated with a single instance of Class_B. Cardinality does not dictate the number of instances of the association or of the individual classes.

CIM Schema
A collection of class definitions that represent managed objects that occur in every management environment. See also core model, common model, and extension schema.

Common Information Model (CIM)
Describes the components of a managed computing and networking environment by using an object-oriented modeling approach. CIM is defined in the CIM Infrastructure specification.

Common Information Model Object Manager (CIM Object Manager)
A component of a WBEM Server that handles request routing to object manager adapters, services (e.g. event services, query services, ...) and providers based on the CIM model.

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

directory
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 LDAP as its access protocol.

Distributed Management Task Force (DMTF)
A not-for-profit organization that delivers technology standards for end-to-end management of distributed computing environments.

Desktop Management Interface (DMI)
Standard that 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.

extension schema
The third layer of the CIM Schema, which includes platform-specific extensions of the CIM Schema. See also common model and core model.

indication
The 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 indication typically 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.

**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.

**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 that is defined for a class but not inherited from a superclass.

**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, and 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 that originates 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 program that initiates management requests. The management client interacts with a management infrastructure or agent (through 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 that represent managed objects. For example, classes, instances, and associations are included in the metamodel.

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

**method**
A function that describes the behavior of a class. Including a method in a class does not guarantee an implementation of the method.

**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 that contain specific classes and instances.

**object path**
A formatted string that is 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.

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.

qualifier
A modifier that contains 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.

range
The class that is referenced by a reference property.

reference
A special string property type that is marked with the reference qualifier, which indicates that it is a pointer to other instances.

required property
A property that must have a value.
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 that represent 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 that is owned by a particular organization.

superclass
The class from which a subclass inherits.

trigger
A recognition of a state change (such as create, delete, update, or access) of a class instance, and the 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.
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 that is capable of encoding all known characters and is used as a worldwide character-encoding standard.

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

Web-Based Enterprise Management (WBEM)
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 that leverage the emerging technologies such as CIM and XML.