[MS-WMIO]:
Windows Management Instrumentation Encoding Version 1.0 Protocol Specification

Revision Summary

Date	Revision History	Revision Class	Comments
03/14/2007	1.0	Major	Updated and revised the technical content.
04/10/2007	1.1	Minor	Updated the technical content.
05/18/2007	1.2	Minor	Addressed EU feedback
06/08/2007	1.3	Minor	Updated the technical content.

Date	Revision History	Revision Class	Comments
07/10/2007	1.3.1	Editorial	Revised and edited the technical content.
08/17/2007	1.3.2	Editorial	Revised and edited the technical content.
09/21/2007	1.4	Minor	Updated the technical content.
10/26/2007	2.0	Major	Converted the document to unified format, and made clarification of ABNF item.
01/25/2008	2.0.1	Editorial	Revised and edited the technical content.

Table of Contents

1 Introduction

This document specifies a binary data **encoding** format used by the Windows Management Instrumentation Remote Protocol, as specified in [MS-WMI], for network communication.

The carrier protocol for this encoding is the Distributed Component Object Model (DCOM) Remote Protocol (as specified in [MS-DCOM]) used in combination with Windows Management Instrumentation (WMI) (as specified in [MS-DCOM]) interfaces (as specified in [MS-WMI]). This specification does not specify Windows Management Instrumentation Remote Protocol (as specified in [MS-WMI]) operations, but specifies the data encoding used by the protocol.

Windows Management Instrumentation (WMI) uses the **Common Information Model (CIM)**, which is published and maintained by the Desktop Management Task Force (DMTF), as specified in [DMTF-DSP004]. The Common Information Model (CIM) Infrastructure Specification (as specified in [DMTF-DSP004]) defines the object model itself, and this specification depends entirely on the metamodel and terminology specified in the DMTF specification set. The reader is referred to the Common Information Model (CIM) Infrastructure Specification (as specified in [DMTF-DSP004]) for a description of the CIM metamodel. The **Common Information Model (CIM) objects** transferred by the Windows Management Instrumentation Remote Protocol Specification (as specified in [MS-WMI]) are CIM objects encoded using the technique specified in this specification.

The DMTF CIM specifications only specify a text-based encoding called **Managed Object Format (MOF)**. However, that format is inefficient for network use. The format specified in this document is an efficient binary format for describing CIM objects within network packets.

It is important to remember that the encoded object at all times is a representation of a CIM object, and any code that manipulates a CIM object MUST observe legal CIM semantics.

Any operation in the Windows Management Instrumentation Remote Protocol Specification (as specified in [MS-WMI]) that transfers a CIM object MUST use the encoding within this specification. The encoding is specified in ABNF notation, as specified in [RFC4234], and the special binary encoding rules for specific tokens are defined in this specification.

1.1 Glossary

The following terms are defined in [MS-GLOS]:

Common Information Model (CIM)
Common Information Model (CIM) Class
Common Information Model (CIM) Instance
Common Information Model (CIM) Object
Common Information Model (CIM) Property
Common Information Model (CIM) Qualifier
Encoding
Fixup
Managed Object Format (MOF)
Superclasses and Subclasses

The following terms are specific to this document:

MAY, SHOULD, MUST, SHOULD NOT, MUST NOT: These terms (in all caps) are used as specified in [RFC2119]. All statements of optional behavior use either MAY, SHOULD, or SHOULD NOT.

1.2 References

1.2.1 Normative References

We conduct frequent surveys of the normative references to assure their continued availability. If you have any issue with finding a normative reference, please contact dochelp@microsoft.com. We will assist you in finding the relevant information. Please check the archive site, http://msdn2.microsoft.com/en-us/library/E4BD6494-06AD-4aed-9823-445E921C9624, as an additional source.

[DMTF-DSP004] Distributed Management Task Force, "Common Information Model (CIM) Infrastructure Specification", Version 2.3, October 2005, http://www.dmtf.org/standards/published_documents/DSP0004V2.3_final.pdf

[IEEE754] Institute of Electrical and Electronics Engineers, "Standard for Binary Floating-Point Arithmetic", IEEE 754-1985, October 1985, http://grouper.ieee.org/groups/754/

[MS-DCOM] Microsoft Corporation, "Distributed Component Object Model (DCOM) Remote Protocol Specification", July 2006.

[MS-GLOS] Microsoft Corporation, "Windows Protocols Master Glossary", March 2007.

[MS-WMI] Microsoft Corporation, "Windows Management Instrumentation Remote Protocol Specification", July 2006.

[RFC2119] Bradner, S., "Key words for use in RFCs to Indicate Requirement Levels", BCP 14, RFC 2119, March 1997, http://www.ietf.org/rfc/rfc2119.txt

[RFC4234] Crocker, D., Ed. and Overell, P., "Augmented BNF for Syntax Specifications: ABNF", RFC 4234, October 2005, http://www.ietf.org/rfc/rfc4234.txt

[UNICODE] The Unicode Consortium, "Unicode Home Page", 2006, http://www.unicode.org/

1.2.2 Informative References

[AHO-ULLMAN] Aho, A., Sethi, R., and Ullman, J., "Compilers: Principles, Techniques, and Tools", Addison-Wesley, January 1986, ISBN: 0201100886.

1.3 Protocol Overview (Synopsis)

The carrier protocol, as specified in [MS-WMI], is the actual protocol for transferring CIM objects specified in this specification. This specification defines a binary format used within the custom marshaling of the Windows Management Instrumentation Remote Protocol (as specified in [MS-WMI]) when CIM objects are being transferred in a protocol operation.

The protocol specified in the Windows Management Instrumentation Remote Protocol Specification (as specified in [MS-WMI]) is a management protocol for querying status and controlling the settings of real-world managed entities. These entities are modeled using CIM objects, as specified in [DMTF-DSP004].

For example, a logical drive might be modeled as a CIM object in which the class of the CIM object is Disk and the various characteristics of the Disk (such as its VolumeLabel, DriveLetter, and the active FileSystem type) are properties within the **CIM class**. CIM class definitions are thus similar to class definitions in other object-oriented database systems and programming systems.

Within Windows Management Instrumentation (WMI), each managed entity is assigned such a CIM class, and instances of that entity become **CIM instances**. Continuing with the previous example, the Disk class may contain three instances: one for the C: drive, one for the D: drive, and one for the E: drive.

To query the status of the real-world CIM objects, the Windows Management Instrumentation Remote Protocol (as specified in [MS-WMI]) is used to retrieve these instances using operations such as GetObject or ExecQuery. If updates are required, the Windows Management Instrumentation Remote Protocol (as specified in [MS-WMI]) is used to send the updated CIM instance over the wire with the new values. Note that to perform an update, the Windows Management Instrumentation Remote Protocol (as specified in [MS-WMI]) writes the complete updated instance, even if only one value is changed. Thus, the Windows Management Instrumentation Remote Protocol (as specified in [MS-WMI]) requires an encoding technique of some kind to move CIM objects across the wire when both reading and writing values.

As specified in [DMTF-DSP004] section 3, CIM classes and instances are defined and illustrated using the MOF syntax. This is a text-only format for use by tools and within documentation, but is not designed for use within a network protocol.

It is not possible to predefine binary layouts for the various types of CIM objects that can be transferred because the system is fully dynamic. New types of classes can be installed and transmitted over the Windows Management Instrumentation Remote Protocol (as specified in [MS-WMI]) by vendors and end users, and the full set of CIM object types is not known by the implementation.

This specification defines a binary encoding format as a representation format for CIM objects. When a client application needs to read a CIM class or instance, a GetObject (as specified in [MS-WMI]) operation is performed, and the CIM object is encoded in the definition in this specification. Similarly, if the CIM instance requires updating, the operation (as specified in [MS-WMI]) PutInstance is used, and the updated CIM instance is encoded using the format in this specification.

The Windows Management Instrumentation Remote Protocol (as specified in [MS-WMI]) can read and write both CIM classes and instances of those classes. This specification details how the CIM classes and their instances are encoded for use in the Windows Management Instrumentation Remote Protocol (as specified in [MS-WMI]).

When retrieving CIM objects, the binary encoding that is transmitted over the Windows Management Instrumentation Remote Protocol (as specified in [MS-WMI]) must be decoded. The binary packet is parsed using ABNF rules in a top-down, recursive descent manner, starting with the root-level grammar rule specified in section 2.2.1. The first octets are examined as the input tokens to the parser, the ABNF rules are examined, and the various branches are taken, consuming the octets until the entire packet is decoded. This is equivalent to LL(1) recursive-descent parsing (for more information, see [AHO-ULLMAN] and numerous compiler textbooks).

For example, the ABNF shows that the first non-terminal token is an octet sequence 0x12345678 (the Signature rule). If the first octets match this, the next rule is the ObjectEncodingLength rule, which specifies that the next four octets specify the encoding length of the entire packet. Once these octets are consumed, the next octets are examined according to matching rules, using the established convention, as specified in [RFC4234].

When encoding CIM objects for transmission, the ABNF grammar is traversed top-down, and the ABNF grammar rules starting in section 2.2.1 are used to emit the correct octets based on the CIM object that needs to be encoded. Decoding uses the same grammar traversal rules except the existing octet sequence is matched against the grammar token by token.

For example, the rules specify that Signature must be the first block, so the encoder emits the octet sequence 0x12345678 to match the required rule. Next, the EncodingLength is required. The encoder does not know how many octets are required to complete the encoding, so some type of placeholder is established, and the emitter continues, encoding the CIM object using the rules until the CIM object is completely examined and encoded. Next, the encoder must decide if the CIM object being encoded is a CIM class or a CIM instance, and then emits the correct octet value using the ObjectFlags rule. Before this octet is emitted, the encoder must decide if the encoded CIM object will contain the server name of origin (the DecServerName rule) and the CIM namespace name (DecNamespaceName). Once it is known if these values will form part of the encoding, the ObjectFlags octet bit values can be set, and the octet can be emitted to the encoding buffer. The rules are traversed for encoding exactly as for decoding, except that the encoder emits the octets instead of recognizing existing octets.

Implementers using the ABNF grammar (as specified in [RFC4234]) should be thoroughly familiar with recursive-descent parsing, the concepts of terminal and non-terminal productions, LL(1) grammar theory, and code emission techniques with syntax-directed translators. The techniques for the encoding and decoding are thus equivalent to those employed by high-level language compilers.

1.4 Relationship to Other Protocols

Because this specification only specifies an encoding, there are no specific relationships to other protocols other than what is specified in [MS-WMI].

1.5 Prerequisites/Preconditions

Because this specification only specifies an encoding, there are no specific prerequisites/preconditions considerations.

1.6 Applicability Statement

The encoding in this specification is used wherever CIM classes and CIM objects are transferred on the wire, as specified in [MS-WMI].

1.7 Versioning and Capability Negotiation

Only one version of the encoding exists as of this writing. There are no provisions for multiple encodings or alternate versions.

1.8 Vendor-Extensible Fields

The encoding format is not extensible.

1.9 Standards Assignments

No previous standards assignments exist for the encoding specified in this document.

2 Messages

Because this specification specifies an encoding used by the Windows Management Instrumentation Remote Protocol Specification (as specified in [MS-WMI]), there are no messages or network-level operations defined.

Windows Management Instrumentation Encoding Version 1.0 Protocol annotated object block encoding is specified in the following sections.

2.1 Introduction

The following sections specify Windows Management Instrumentation Encoding Version 1.0 Protocol annotated object block encoding.

2.2 Annotated Object Block Encoding

CIM instance and CIM class definitions, as specified in [DMTF-DSP004] section 2.1, are encoded using a binary data format. To capture the semantics of CIM classes and CIM instances, the layout of the block reflects the CIM object structure and is correspondingly complex but completely canonical according to the ABNF, as specified in section 7.

Because CIM classes and CIM instances have user-name properties and values, the data block can vary significantly, depending on the item being encoded. The traversal of the block is precisely equivalent to top-down parsing, using the well-known LL(1) parsing algorithm, and can be implemented in a recursive-descent parser (for more information, see [AHO-ULLMAN] section 4.4).

The representation of the grammar of the packet layout is presented in ABNF notation, as specified in [RFC4234]. Terminal tokens are in uppercase, such as UINT32, and have a binary encoding rule, as specified in section 3. All other grammar productions are defined within the ABNF.

In the definitions in this section, the use of the term offset always refers to an unsigned integer value representing the distance in octets from some base point. An offset of zero indicates a reference to the first octet in the block, and an offset of seven indicates a reference to the eighth octet of the block.

2.2.1 EncodingUnit

The EncodingUnit is the root node block of the encoding used for encoding classes or instances. This block is contained within transmission (as specified in [MS-DCOM]) encoding, as specified in [MS-WMI].

```
EncodingUnit = Signature ObjectEncodingLength ObjectBlock
```

The **Signature** field (as specified in section 2.2.75) acts as a verification signature that the CIM object (that follows) is encoded with, according to the algorithm specified in this document. Any other value MUST constitute an error. The ObjectEncodingLength (section 2.2.2) represents the size of the ObjectBlock (section 2.2.3) that contains the encoded CIM class or CIM instance.

2.2.2 ObjectEncodingLength

ObjectEncodingLength is a 32-bit unsigned integer that specifies the length of the ObjectBlock (section 2.2.3).

```
ObjectEncodingLength = UINT32
```

2.2.3 ObjectBlock

ObjectBlock is where the actual binary encoding of the CIM object begins.

```
ObjectBlock = ObjectFlags [Decoration] Encoding
```

ObjectFlags (section 2.2.4) indicates if the Decoration (section 2.2.5) block is present and if the CIM object is a CIM class definition or a CIM instance. The Encoding (section 2.2.8) block contains either a CIM class or CIM instance definition, depending on the value of ObjectFlags.

2.2.4 ObjectFlags

The ObjectFlags block is used to classify the currently encoded object. It MUST be one octet in length, but only bits 0, 1, and 2 MUST be used.

```
ObjectFlags = OCTET
```

The octet MUST have only one of the following values. They are mutually exclusive and cannot be combined.

Value	Meaning
0x01	Object is a CIM class, undecorated.
0x02	Object is a CIM instance, undecorated.
0x05	Object is a CIM class with Decoration block.
0x06	Object is a CIM instance with Decoration block.

Note that this is equivalent to bits 0 and 1, indicating whether the object is a ClassType or an InstanceType; and bit 2, indicating if the Decoration block is present.

Note that not all of the bits of the octet are used, and MUST be zero.

2.2.5 Decoration

The Decoration block is used to optionally decorate the CIM object with information as to what server and CIM namespace it originates from. This block MUST be present if the ObjectFlags (section 2.2.4) octet value is 0x05 or 0x06; otherwise, it MUST be absent.

```
Decoration = DecServerName DecNamespaceName
```

In the encoded sequence, the strings DecServerName (section 2.2.6) and DecNamespaceName (section 2.2.7) MUST be placed inline. If either of the strings has no value, an empty Encoded-String MUST be present. The two Encoded-String values MUST NOT be omitted, even if empty.

2.2.6 DecServerName

DecServerName is an Encoded-String representing the server name from which the CIM object originates. The format of the string is purely documentary and may be in any format such as a NetBIOS name, a DNS name, an IP address, or any other name that is expected to be useful in determining the origin of the packet.

```
DecServerName = Encoded-String
```

2.2.7 DecNamespaceName

DecNamespaceName is an Encoded-String representing the CIM namespace name from which the CIM object originates.

```
DecNamespaceName = Encoded-String
```

2.2.8 Encoding

Because the encoding carries a CIM class or a CIM instance, the Encoding block is merely a switch to select the correct block.

```
Encoding = InstanceType / ClassType
```

InstanceType (section 2.2.50) encodes CIM instance, and ClassType (section 2.2.9) encodes the CIM class object.

2.2.9 ClassType

The ClassType block is used to define a CIM class. It consists of two CIM class definitions located sequentially to each other. The first block consists of the definition of the **superclass** to the current CIM class. The second block is the actual CIM class definition being encoded in the current EncodingUnit.

```
ClassType = ParentClass CurrentClass
```

That is, if the CIM class hierarchy is:

```
class MyBase {  }
class MyDerived : MyBase { }
```

... then the ParentClass block contains the definition of MyBase, and the CurrentClass block contains the definition for MyDerived.

A class might not have a superclass, as specified in [DMTF-DSP004] appendix A. The ParentClass block MUST be present even if the class coded in CurrentClass has no superclass. The values in the ParentClass block and its subblocks are zero, NULL, empty, or otherwise logically empty, but the block MUST be present nonetheless.

2.2.10 ParentClass

ParentClass is the CIM class that is the immediate parent of the current CIM class, according to the inheritance mechanism specified in [DMTF-DSP004].

```
ParentClass = ClassAndMethodsPart
```

ClassAndMethodsPart (section 2.2.12) specifies the properties and method signatures for the class.

2.2.11 CurrentClass

CurrentClass is the encoding of the CIM class that the EncodingUnit represents. The ClassType block requires the encoding to contain both the encoding of the ParentClass for the class as well as the CIM class itself, which is specified by this rule.

```
CurrentClass = ClassAndMethodsPart
```

The InstanceType block MUST also contain a CurrentClass block as part of its own definition because CurrentClass is reachable through several subrules within this grammar.

2.2.12 ClassAndMethodsPart

The ClassAndMethodsPart block divides the CIM class definition into two sections:

- ClassPart (section 2.2.13) contains the data declarations (properties).
- MethodsPart (section 2.2.35) contains the method table.

The semantic meaning of properties and methods within a class is specified in [DMTF-DSP004].

```
ClassAndMethodsPart = ClassPart [MethodsPart]
```

MethodsPart (section 2.2.35) MUST always be present if the ObjectFlags (section 2.2.4) value indicates that the outermost object being encoded is a ClassType (section 2.2.9). MethodsPart MUST NOT be present if the ObjectFlags indicates that the outermost object being encoded is an InstanceType (section 2.2.50).

2.2.13 ClassPart

The ClassPart block contains the actual core of a CIM class definition, as specified in [DMTF-DSP004]. Each field MUST be located serially after the other.

```
ClassPart =  ClassHeader DerivationList ClassQualifierSet
    PropertyLookupTable [NdTable ValueTable] ClassHeap
```

The ClassHeader (section 2.2.14) contains information on the overall ClassPart block length and the length of various internal blocks. The DerivationList (section 2.2.15) is an encoded array that MUST contain the set of CIM class names that form the list of superclasses for the current CIM class.

The ClassQualifierSet (section 2.2.18) is the set of **CIM qualifiers** for the class.

The PropertyLookupTable (section 2.2.19) is a sorted dispatch table for looking up **CIM property** values and type information. The NdTable (section 2.2.24) indicates if any given CIM property has a default value that is locally defined within the current CIM class, or if the default is defined in a superclass. The ValueTable (section 2.2.27) contains values inline for simple numeric properties or references to the values in the ClassHeap (section 2.2.34) for all other values specified in the HeapItem rule such as arrays or strings.

NdTable and ValueTable are optional. Their inclusion is controlled by the number of properties in PropertyLookupTable. If there are zero properties, NdTable and ValueTable MUST be omitted.

Informally, the ClassHeap is simply a scratchpad area that contains long, irregular values such as strings. These are referenced by a simple integer offset from the beginning of ClassHeap.

2.2.14 ClassHeader

ClassHeader contains various details on the CIM class block.

```
ClassHeader = EncodingLength ReservedOctet ClassNameRef ValueTableLength
```

The EncodingLength (section 2.2.71) field applies to the ClassPart as a whole, not just the ClassHeader. The ReservedOctet (section 2.2.74) octet is not used, and MUST be zero. The ClassNameRef (section 2.2.17) contains a reference to the string that is the name of the current CIM class. The ValueTableLength (section 2.2.26) is the length of the encoded "ClassPart::ValueTable" block in octets.

2.2.15 DerivationList

DerivationList is an encoded array that indicates the list of superclasses that form the inheritance chain of the current class. The array only contains the names of the superclasses. The order of classes is significant. The immediate superclass of the current class is first followed by each successive parent class, terminating in the top-most class.

```
DerivationList = EncodingLength *ClassNameEncoding
```

EncodingLength (section 2.2.71) includes itself in the total. So an empty array still contains at least one UINT32 value of 0x4 hexadecimal, which is the length of the EncodingLength item. ClassNameEncoding (section 2.2.16) contains the names of the superclasses.

2.2.16 ClassNameEncoding

Each ClassNameEncoding is an Encoded-String that is suffixed by a 32-bit value indicating the length in characters of the Encoded-String. This length includes the value of the leading octet flag and NULL terminator, not just the visible character count.

```
ClassNameEncoding = Encoded-String EncodingLength
```

2.2.17 ClassNameRef

ClassNameRef is a reference to the current CIM class name. It is a HeapStringRef (section 2.2.65) in the ClassHeap (section 2.2.34).

```
ClassNameRef = HeapStringRef
```

2.2.18 ClassQualifierSet

ClassQualifierSet is the CIM qualifier set for the current class.

```
ClassQualifierSet = QualifierSet
```

As applied to classes, the ClassQualifierSet is a set of qualifiers, as specified in [DMTF-DSP004], that applies to the CIM class definition as a whole.

```
[Qualifier1, Qualifier2, ...QualifierN]
class Sample
{
      ....
}
```

This usage in CIM is distinct from qualifiers that apply to various internal declarations such as properties and methods.

2.2.19 PropertyLookupTable

PropertyLookupTable is a simple dispatching table for finding properties. The PropertyCount (section 2.2.20) indicates how many properties follow in the PropertyLookup (section 2.2.21) sequence.

```
PropertyLookupTable = PropertyCount *PropertyLookup
```

The PropertyLookup sequence MUST be sorted according to the lexical ordering established by the character set specified in [UNICODE]. This sort order is required because implementations expect to perform binary search operations on the table, and such searches require lexical ordering.

2.2.20 PropertyCount

PropertyCount is the total number of properties in the class. If zero, the optional NdTable (section 2.2.24) and ValueTable (section 2.2.27) blocks (as specified in section 2.2.13) MUST be absent.

```
PropertyCount = UINT32
```

2.2.21 PropertyLookup

The PropertyLookup structure represents a data block that allows a lookup of a specific, named CIM property within a CIM class. The PropertyNameRef (section 2.2.22) item is a reference to the string name of the Encoded-String on the ClassHeap that represents the name of the property. The PropertyInfoRef (section 2.2.23) item contains a heap reference to other information on the CIM property in a PropertyInfo item that contains its CIM type and any associated qualifiers.

```
PropertyLookup = PropertyNameRef PropertyInfoRef
```

These items are simple references into the ClassHeap, and each item is only 32 bits in length.

2.2.22 PropertyNameRef

PropertyNameRef MUST be a heap reference to the Encoded-String for the CIM property name.

```
PropertyNameRef = HeapStringRef
```

2.2.23 PropertyInfoRef

PropertyInfoRef MUST be a heap reference to the PropertyInfo (section 2.2.28) item for the property.

```
PropertyInfoRef = HeapRef
```

2.2.24 NdTable

NdTable is an encoded table that represents the behavior of the default value of properties within a CIM class.

The table is ordered according to the same order implied by the PropertyLookupTable.

Classes are allowed to establish default values for properties, as specified in [DMTF-DSP004]. In some cases, the default value for a CIM property can actually be defined in a superclass; for example, using the MOF syntax for CIM.

```
class Base
{
                    ...
    sint32 ValueX = 123;
}
class Derived : Base
{
          sint32 ValueY = 456;
}
```

In this case, both ValueX and ValueY have defaults, but they are established in different classes. Because the Derived class contains all the information from Base, the effective class declaration is similar to the following.

```
class Derived : Base
{
    sint32 ValueX = 123;
          sint32 ValueY = 456;
}
```

However, for many operations within the processing of CIM objects outside of network protocol operations, it is important to distinguish if the default value is inherited or if it is locally defined within the current class. Therefore, this information must be maintained within the encoding.

Only two bits are required to indicate this information for each property; therefore, the bit fields are packed into octets.

```
NdTable = *NullAndDefaultFlag
```

The total number of bits is the number of properties * 2 rounded up to the nearest whole octet count. Specifically, the number of octets required is the following.

```
octetCount = (PropertyCount  - 1) / 4 + 1
```

Because of this rounding effect, there may be unused bits in the octet. These bits MAY have any value.

2.2.25 NullAndDefaultFlag

NullAndDefaultFlag is a two-bit field for a property.

```
NullAndDefaultFlag = 2*BIT
```

If bit 0 is set, the default value is NULL. If bit 1 is set, the default value is inherited from some parent CIM class in the inheritance hierarchy. Combinations of Bit 0 and Bit 1 result in the Default Property value behavior in the following table.

BIT 0 state	BIT 1 state	Implication
SET	SET	Default Property value is NULL, and it is inherited from a parent class.
SET	NOT SET	Default Property value is NULL, and it is set by the current class.
NOT SET	SET	Default Property value is NOT NULL, and it is inherited from a parent class.
NOT SET	NOT SET	Default Property value is NOT NULL, and it is set by the current class.

2.2.26 ValueTableLength

ValueTableLength is the length of the ValueTable in octets.

```
ValueTableLength = UINT32
```

Unlike EncodingLength rules, this does not include its own length.

2.2.27 ValueTable

The ValueTable encodes the literal values of the properties or references to their values in the heap.

```
ValueTable = *EncodedValue
```

Depending on the type of the CIM property, each EncodedValue (section 2.2.69) has variable length. The sequence of EncodedValues is packed at the octet level with no alignment or padding.

To find the value for a property, navigate from the PropertyLookupTable (section 2.2.19) to its PropertyLookup (section 2.2.21), and from there get the PropertyInfoRef (section 2.2.23), which will give the PropertyInfo (section 2.2.28). From PropertyInfo, get the ValueTableOffset (section 2.2.31). Use this offset in the ValueTable (section 2.2.27) to discover the value.

If the value is numerical, the value MUST be located directly within this table. If the value is a string or an array type, the value table MUST contain a reference, HeapRef (section 2.2.66), into the Heap (section 2.2.63) to find the actual value.

2.2.28 PropertyInfo

The PropertyInfo element exists in the heap and is referenced through a PropertyLookup block. It contains information on a property other than its value such as: its data type declaration order, in which class it was defined in an inheritance hierarchy, and offsets to the value table and qualifier set for the property.

```
PropertyInfo = PropertyType DeclarationOrder
ValueTableOffset ClassOfOrigin PropertyQualifierSet
```

2.2.29 PropertyType

PropertyType encodes the data type of the property.

```
PropertyType = CimType
```

It MUST have the form specified in section 2.2.80.

2.2.30 DeclarationOrder

The DeclarationOrder element shows the actual order of the CIM property as it appears in the order within the CIM declaration of the MOF for the class, as specified in [DMTF-DSP004].

```
DeclarationOrder = UINT16
```

2.2.31 ValueTableOffset

ValueTableOffset MUST be the offset within the ValueTable (section 2.2.27) that contains the value for the property. Depending on the type of the property, the ValueTable entry is interpreted differently. The type for the CIM property and other information are located in the PropertyType (section 2.2.29) entry, a sibling of this ValueTableOffset within the larger PropertyInfo (section 2.2.28) encoding.

```
ValueTableOffset = UINT32
```

2.2.32 ClassOfOrigin

ClassOfOrigin defines what CIM class in the DerivationList the CIM property comes from, where 0 indicates the first CIM class in the DerivationList, and so on. If the CIM property is local to the current class, the ClassOfOrigin is equal to the number of items in the DerivationList.

```
ClassOfOrigin = UINT32
```

2.2.33 PropertyQualifierSet

PropertyQualifierSet is a set of qualifiers that applies to the preceding property. There is no count of qualifiers. Qualifiers within the QualifierSet are decoded and recognized until the "QualifierSet::EncodingLength" is exhausted.

```
PropertyQualifierSet = QualifierSet
```

2.2.34 ClassHeap

ClassHeap is structured like any other heap except that the items contained within it only apply to the CIM class definition.

```
ClassHeap = Heap
```

Because instances also contain class definitions as part of their encoding, it is important to ensure that the heap references are not intermixed between the class and instance parts.

All heap references that occur within the ClassPart (section 2.2.13) block MUST be limited to references within the ClassHeap.

2.2.35 MethodsPart

The MethodsPart block is the second half of the ClassType encoding rule and defines the methods for the class.

```
MethodsPart = EncodingLength MethodCount
         MethodCountPadding *MethodDescription MethodHeap
```

A class encoding with no methods still MUST contain the fields indicated. MethodCount MUST be zero, and there MUST be a zero-length MethodHeap encoded according to their respective rules.

2.2.36 MethodCount

MethodCount is the number of methods in the class.

```
MethodCount = UINT16
```

2.2.37 MethodCountPadding

MethodCountPadding is a two-octet sequence that is not used, and MUST be set to zero.

```
MethodCountPadding = 2*OCTET
```

2.2.38 MethodDescription

MethodDescription specifies one method.

```
MethodDescription = MethodName MethodFlags MethodPadding
      MethodOrigin MethodQualifiers InputSignature OutputSignature
```

2.2.39 MethodName

MethodName MUST be a simple HeapStringRef to the MethodHeap (section 2.2.49) for the method name.

```
MethodName = HeapStringRef
```

2.2.40 MethodFlags

MethodFlags is reserved, and MUST be zero.

```
MethodFlags = OCTET
```

2.2.41 MethodPadding

MethodPadding is reserved, and SHOULD be zero when encoding objects.

```
MethodPadding = 3OCTET
```

When decoding objects, values other than zero MUST be ignored. Because the fields are not used, some implementations such as Windows may place random values in these octets; and thus decoders should not consider such random values to be errors in the encoding.

2.2.42 MethodOrigin

MethodOrigin is a zero-origin array index to a CIM class name in the DerivationList showing what CIM class owns the method.

```
MethodOrigin = UINT32
```

A value of zero refers to the first element in the DerivationList. A value of 1 refers to the second element in the DerivationList, and so on. If the method is local to the current class, the MethodOrigin is equal to the number of items in the DerivationList.

2.2.43 MethodQualifiers

MethodQualifiers is a set of qualifiers applicable to the method.

```
MethodQualifiers = HeapQualifierSetRef
```

MethodQualifiers MUST be a HeapQualifierSetRef (section 2.2.44) in the MethodHeap (section 2.2.49). The QualifierSet (section 2.2.56) referred to by the HeapQualifierSetRef is the CIM qualifiers set applicable to the method. For example, in the following CIM class, the execute CIM qualifier and performance CIM qualifier are method-level qualifiers, while in and out are parameter-level qualifiers.

```
class MyClass2  :  MyClass
{
   [execute, performance={"fast", "sideffects"}]
      uint32 Restart([in] string ServiceName, [out] int Status);
}
```

2.2.44 HeapQualifierSetRef

HeapQualifierSetRef MUST be a HeapRef (section 2.2.66) to a single QualifierSet (section 2.2.56) in the current heap.

```
HeapQualifierSetRef = HeapRef
```

2.2.45 InputSignature

InputSignature specifies the input signature for the method.

```
InputSignature = MethodSignature
```

2.2.46 OutputSignature

OutputSignature specifies the output signature for the method.

```
OutputSignature = MethodSignature
```

2.2.47 MethodSignature

The InputSignature and OutputSignature fields MUST be a HeapRef to MethodSignatureBlock (section 2.2.67) within the MethodHeap (section 2.2.49). This is because the input and output signatures for a method are encoded as a ClassPart where each CIM property represents a parameter in the method.

```
MethodSignature = HeapMethodSignatureBlockRef
```

To encode a MethodSignature as a CIM class object, the encoding rules, as specified in section 3.3, MUST be used. These rules do not affect the structure of the encoding but establish conventions for content such as the name of the class and how to indicate in and out parameter flow by using qualifiers.

2.2.48 HeapMethodSignatureBlockRef

HeapMethodSignatureBlockRef MUST be a HeapRef to MethodSignatureBlock (section 2.2.67) in the current Heap (section 2.2.63).

```
HeapMethodSignatureBlockRef = HeapRef
```

2.2.49 MethodHeap

MethodHeap contains information on all the methods: their names, parameters, types, and so on.

```
MethodHeap = Heap
```

All HeapItem entries within the Heap MUST be referenced by a valid HeapRef within the MethodsPart encoding block.

2.2.50 InstanceType

The InstanceType block is used to encode a CIM instance of a CIM class.

```
InstanceType = CurrentClass EncodingLength InstanceFlags
     InstanceClassName NdTable InstanceData
        InstanceQualifierSet InstanceHeap
```

As indicated in the encoding rule, a CIM instance is prefixed by the CIM class definition to which it belongs.

The EncodingLength field specifies the length in octets of itself and all the following fields. This is equivalent to the length of the InstanceType block, excluding the CurrentClass block.

InstanceFlags is a reserved octet, and MUST be zero.

The CIM class name to which the CIM instance belongs is referenced by InstanceClassName.

The actual instance-level data is in NdTable and InstanceData, and any instance-level qualifiers are in InstanceQualifierSet. Because default values from CIM class definitions may be used in a CIM instance, as specified in [DMTF-DSP004], the NdTable bits are set to indicate whether NULL or a default value is in use for each property.

The values for any referenced items anywhere within the InstanceType encoding block MUST be contained within the InstanceHeap.

2.2.51 InstanceFlags

InstanceFlags is reserved, and must be zero.

```
InstanceFlags = OCTET
```

2.2.52 InstanceClassName

InstanceClassName is a string reference to a class name in the InstanceHeap.

```
InstanceClassName = HeapStringRef
```

2.2.53 InstanceData

InstanceData values are stored in a ValueTable just like classes are. The only difference is that the values in ValueTable MUST contain references to the InstanceHeap whenever a HeapRef occurs.

```
InstanceData = ValueTable
```

2.2.54 InstanceQualifierSet

InstanceQualifierSet is the CIM qualifier set that SHOULD apply to the entire instance, as opposed to qualifiers within individual properties.

```
InstanceQualifierSet = QualifierSet InstancePropQualifierSet
```

2.2.55 InstanceHeap

InstanceHeap is the value heap for the current instance.

```
InstanceHeap = Heap
```

2.2.56 QualifierSet

QualifierSet represents a set of qualifiers. Qualifiers are applied to a CIM class, a CIM instance, properties within a CIM class or instance, methods, and individual parameters within methods, as specified in [DMTF-DSP004].

```
QualifierSet = EncodingLength *Qualifier
```

The length of the QualifierSet is indicated by the EncodingLength.

This is followed by a series of CIM qualifier values of variable length. Each one begins where the previous one ends. There are no delimiters between qualifiers or other means indexing to specific qualifiers.

Because each CIM qualifier block is of a known length, the end of the QualifierSet is reached where the value (EncodingLength - 4) is equal to the length of the set of CIM qualifier blocks that follow it.

2.2.57 Qualifier

Qualifier defines a single qualifier.

```
Qualifier = QualifierName QualifierFlavor
    QualifierType QualifierValue
```

The CIM qualifier consists of the name, flavor, and data type of the qualifier, and the actual value, as specified in [DMTF-DSP004] section 4.5.4.

2.2.58 QualifierName

QualifierName is a CIM qualifier name, and MUST be a HeapRef to an Encoded-String in the current heap.

```
QualifierName = HeapStringRef
```

Class qualifiers should be located in the ClassHeap, CIM instance qualifiers should be located in the InstanceHeap, and method qualifiers should be located in the MethodHeap.

2.2.59 QualifierFlavor

QualifierFlavor indicates the origin and propagation rules for the qualifier.

```
QualifierFlavor = OCTET
```

The following bit encodings MUST apply. Services SHOULD ignore any other bit values.<1>

Qualifier flavor	Meaning	Bit values
WBEM_FLAVOR_FLAG_PROPAGATE_TO_INSTANCE	If set, the qualifier is progagated to instances. If not set, the qualifier is not progagated to instances.	0x01
WBEM_FLAVOR_FLAG_PROPAGATE_TO_DERIVED_CLASS	The qualifier is propagated to derived classes.	0x02
WBEM_FLAVOR_NOT_OVERRIDABLE	The qualifier value cannot be overridden in a derived class or an instance.	0x10
WBEM_FLAVOR_ORIGIN_PROPAGATED	Origin propagated.	0x20
WBEM_FLAVOR_ORIGIN_SYSTEM	Origin system.	0x40
WBEM_FLAVOR_AMENDED	The qualifier is localized.	0x80

The meanings and combinations of usage for these CIM qualifier flavors are as specified in [DMTF-DSP004].

Although the value of 0x0 contains no bits set, a value of zero has a specific meaning, which indicates that the qualifier is present on a class definition and should not be propagated to the instance encoding.

2.2.60 QualifierType

QualifierType is a CIM qualifier, and MUST be any valid CimType (section 2.2.80).

```
QualifierType = CimType
```

2.2.61 QualifierValue

QualifierValue is the value of a CIM qualifier, and MUST be a valid EncodedValue based on the QualifierType.

```
QualifierValue = EncodedValue
```

2.2.62 InstancePropQualifierSet

InstancePropQualifierSet is a CIM qualifier set for instances that have properties with instance-level qualifiers. Because this rarely occurs, there is a flag octet that signals whether or not there are CIM qualifier sets for the properties. Typically there are none, and the flag value MUST be set to 1.

```
InstancePropQualifierSet = InstPropQualSetFlag *QualifierSet

InstPropQualSetFlag = %x1 / %x2
```

If the InstPropQualSetFlag is set to 2, the QualifierSet sequence MUST be populated. There MUST be one QualifierSet for each CIM property in the class, and the properties are in the same order that occurs in the PropertyLookupTable.

Once the flag value is set to 2, all of the CIM qualifier sets for all of the properties MUST be present, even if they are empty. For example, the following CIM instance has a CIM qualifier on the CIM property Data1 (the test qualifier).

```
instance of MyClass
{
 Array = {1, 2, 3};
 [test] Data1 = "StringField";
 Id = 123;
};
```

The binary encoding of this CIM instance contains CIM qualifier sets for each of its properties whether or not there are any qualifiers for that property (there is at least an EncodingLength for that qualifier set).

For examples, see section 4.1.

2.2.63 Heap

A Heap consists of a length and a linear series of HeapItem entries. A Heap is loosely defined and consists of the HeapItem blocks in any order. However, there are three separate Heaps that MUST be maintained distinctly: ClassHeap (only applies to CIM class data), InstanceHeap (only applies to CIM instance data), and MethodHeap (only appears within ClassType blocks and only contains information relating to the methods for a CIM class). These Heaps MUST be separate, and they only apply within their respective encoding blocks. That is, ClassHeap only occurs within ClassType, InstanceHeap only occurs within InstanceType, and MethodHeap only occurs within MethodsPart.

This is because ClassHeap (references) to HeapItem entries are encoded as simple integer offsets from the beginning of the relevant Heap, so the actual target Heap is implied by the block that the HeapRef occurs in.

```
Heap = HeapLength *HeapItem
HeapLength = UINT32    ; 31 bits with MS bit set
```

HeapLength is a 32-bit value with the most significant bit always set (using little-endian binary encoding for the 32-bit value), so that the length is actually only 31 bits in length.

The items appear in any order and do not need to be packed. Heaps MAY occur when there are holes between items in the Heap or when there is padding before or after the first and last items. As long as any HeapRef type is properly adjusted to point to items within the Heap, such gaps are acceptable and are permitted to accommodate garbage collection mechanisms in the encoders and decoders.

Any HeapRef value MUST be the offset (in total octets) of the corresponding HeapItem, and any HeapItem MUST have exactly one HeapRef in some other data structure that points to it.

HeapItem entries MUST NOT be shared. That is, there MUST NOT exist two HeapRef values that point to the same HeapItem.

2.2.64 HeapItem

HeapItem is one of the following data block types. Every HeapItem MUST have a corresponding HeapRef (section 2.2.66).

```
HeapItem = PropertyInfo / Encoded-String /
     ENCODED-ARRAY / QualifierSet / ObjectBlock / MethodSignatureBlock
```

The HeapRef that points to a specified HeapItem is not inferable from the HeapItem itself. Although all HeapRefs point to HeapItems, there is no way to navigate from the HeapItem back to the HeapRef that points to it. HeapRefs can only be located by following the various encoding rules in EncodingUnit (section 2.2.1).

2.2.65 HeapStringRef

HeapStringRef MUST be a reference to an Encoded-String on the current Heap.

```
HeapStringRef = HeapRef
```

2.2.66 HeapRef

HeapRef is a reference to any HeapItem and is expressed in 31 bits. If the HeapItem (section 2.2.64) referred to is a string, and the most significant bit of the 32-bit HeapStringRef (section 2.2.65) value is set, the reference is actually to an implied dictionary-based string entry and does not point to a literal Encoded-String within the Heap.

```
HeapRef = UINT32 / DictionaryReference
```

If the most significant bit of the 32-bit value is clear, the reference is an offset to a HeapItem in the Heap.

2.2.67 MethodSignatureBlock

MethodSignatureBlock is a block used to encode a set of in parameters or out parameters for a method definition in a CIM class. MethodSignatureBlock is simply an ObjectBlock using the Method encoding format rules, as specified in section 3.3. The length of the entire block MUST be prefixed by an EncodingLength, which includes its own length as well as the length of the ObjectBlock.

```
MethodSignatureBlock = EncodingLength [ObjectBlock]
```

2.2.68 HeapPropertyInfoRef

HeapPropertyInfoRef is a HeapRef that points to a PropertyInfo structure.

```
HeapPropertyInfoRef = HeapRef
```

2.2.69 EncodedValue

EncodedValue is an encoded value that is used everywhere to represent numerical and string values. If the value is numerical, the encoded value is inline. If the value is an Encoded-String or an Encoded-Array, the EncodedValue is a HeapRef (section 2.2.66) to that Encoded-String or Encoded-Array.

```
EncodedValue = NumericValue / HeapRef / BOOL / NoValue
```

2.2.70 NumericValue

NumericValue is any numerical value whether integer or real that is valid within the CIM type system.

```
NumericValue = BYTE / SINT16 / UINT16 / SINT32 /
       UINT32 / SINT64 / UINT64 / REAL32 / REAL64
```

For each of these types, the binary encoding rules are specified in the following table.

The CIM model defines standard numerical data types, as specified in [DMTF-DSP004] section 2.2.

CIM type as specified in [DMTF-DSP004]	ABNF representation	Binary representation
uint8	OCTET, BYTE	8-bit unsigned integer.
sint8	OCTET, BYTE	8-bit signed integer.

CIM type as specified in [DMTF-DSP004]	ABNF representation	Binary representation
uint16	UINT16	16-bit unsigned integer.
sint16	SINT16	16-bit signed integer.
uint32	UINT32	32-bit unsigned integer.
sint32	SINT32	32-bit signed integer.
real32	REAL32	Specification [IEEE754] 4-byte floating point format.
real64	REAL64	Specification [IEEE754] 8-byte floating point format.
sint64	SINT64	Signed 64-bit integer.
uint64	UINT64	Unsigned 64-bit integer.

- The Binary representations MUST be used to encode the specified CIM data types.
- All signed and unsigned integer types consisting of more than one octet MUST be encoded as little-endian.
- The CIM Boolean type has its own encoding specified in the BOOL (section 2.2.73) encoding rule.
- Specification [IEEE754] floating point values MUST be encoded as little-endian.

2.2.71 EncodingLength

EncodingLength is a simple 32-bit unsigned value that establishes the encoding length in octets of one of the other defined units in this specification. This value MUST include its own length as part of any length it is describing. Because of this, the minimum encoding length is 0x4, which is the size of the EncodingLength UINT32.

```
EncodingLength = UINT32
```

2.2.72 NoValue

NoValue is used when a default value does not occur in a CIM class definition for a given CIM property, and a slot in the ValueTable has to be filled for that property. It is simply a four-octet sequence with all of the bits set.

```
NoValue = %xFF %xFF %xFF %xFF     ; 32 bits all set to 1
```

2.2.73 BOOL

BOOL is used to represent logical TRUE or logical FALSE and consists of a 16-bit value.

```
BOOL = 2OCTET
```

The encoding for logical FALSE is all bits set to zero (0x0), and the encoding for logical TRUE is all bits set to 1 or 0xFFFF.

2.2.74 ReservedOctet

ReservedOctet is a reserved OCTET that MUST be set to zero, and is used in several places in the encoding.

```
ReservedOctet = OCTET
```

2.2.75 Signature

Signature is the leading signature on the entire EncodingUnit block, and MUST consist of a literal 32-bit value.

```
Signature = UINT32  ;0x12345678 little-endian
```

This is used to verify that the CIM object being processed conforms to this specification.

2.2.76 Encoded-String

Encoded-String is a special data type that is the sole means of representing strings.

```
Encoded-String = Encoded-String-Flag *Character Null
Encoded-String-Flag = OCTET
Character = AnsiCharacter / UnicodeCharacter
Null = Character
AnsiCharacter = OCTET
UnicodeCharacter = 2*OCTET
```

The Encoded-String string data type is encoded using an encoding flag that consists of one octet followed by a sequence of character items using one of two formats followed by a null terminator.

The FlagOctet is set to 0x00000001 if the sequence of characters that follows consists of UTF-16 characters (as specified in [UNICODE]) followed by a UTF-16 null terminator.

For optimization reasons, the implementation MUST compress the UTF-16 encoding. If all of the characters in the string have values (as specified in [UNICODE]) that are from 0 to 255, the string MUST be compressed. The compression is done by representing each character as a single OCTET with its Unicode value. That is, for each Unicode character, only the lower-order byte is included in the output. A terminating null character MUST be represented by a single OCTET. When the string is compressed, FlagOctet is set to 0x00. This is distinct from UTF-8, which may contain multiple-byte encodings for single characters.

When the string contains characters (as specified in [UNICODE]) outside of this range, this optimization MUST NOT be used. For example, the character K (which is UTF+004B) follows.

0	1	2	3	4	5	6	7	8	9	10	1	2	3	4	5
0	0	0	0	0	0	0	0	0	1	0	0	1	0	1	1

The upper eight bits are all zero bits. If all of the characters for a string have this quality, the string MUST be reduced to its eight-bit equivalent on a character-by-character basis. If any character (as specified in [UNICODE]) contains nonzero bits in both the upper and lower octets of the character, this optimization MUST NOT be used for the string.

This compression technique applies to characters in the U+0000 through U+00FF, and MUST be accompanied by the appropriate Encoded-String-Flag value at the beginning of the encoding.

For any specified CIM object encoding as a whole, the individual strings may or may not use the optimization, depending precisely on what characters are present in the string.

2.2.77 Encoded-Array

Encoded-Array is used to encode an array in the Heap.

```
ENCODED-ARRAY = ArrayCount *EncodedValue
ArrayCount = UINT32
```

Encoded-Array consists of a UINT32 value that specifies how many EncodedValue entries follow. Every element of an array must be of the same CimBaseType.

ArrayCount MUST be present, but there can be zero EncodedValue entries if ArrayCount is zero.

2.2.78 DictionaryReference

DictionaryReference is used to encode extremely common strings to prevent them from taking up space in the Heap. Whenever a reference to an Encoded-String occurs, if the string matches any of the values listed, the most significant bit MUST be set, and the rest of the offset is replaced by the ordinal position of the string in the following dictionary.

```
DictionaryReference = %d0 / %d1 / %d2 / %d3 /
    %d4 / %d5 / %d6 / %d7 / %d8 / %d9 / %d10
  ; %d0   Encoded/Decoded as quote character
  ; %d1   Encoded/Decoded as "key"
  ; %d2   Encoded/Decoded as ""
  ; %d3   Encoded/Decoded as "read"
  ; %d4   Encoded/Decoded as "write"
  ; %d5   Encoded/Decoded as "volatile"
  ; %d6   Encoded/Decoded as "provider"
  ; %d7   Encoded/Decoded as "dynamic"
  ; %d8   Encoded/Decoded as "cimwin32"
  ; %d9   Encoded/Decoded as "DWORD"
  ; %d10  Encoded/Decoded as "CIMTYPE"
```

For example, if the string dynamic is required, a 32-bit binary value of 0x80000007 is used instead of a normal HeapRef.

This technique only applies if the type of the item being pointed to is a string.

2.2.79 BIT

BIT is a simple bit field consisting of 1 bit either set or clear.

```
BIT = %x0 / %x1 ; one bit, either clear or set
```

It is used only by the NdTable rule.

2.2.80 CimType

CimType is a 32-bit value of which only the lower 16 bits are used, and it indicates the type of the value according to the CIM type system.

```
CimType = CimBaseType / CimArrayType

CimBaseType = CIM-TYPE-EMPTY / CIM-TYPE-ILLEGAL /
   CIM-TYPE-SINT8     / CIM-TYPE-UINT8 /
   CIM-TYPE-SINT16    / CIM-TYPE-UINT16 /
   CIM-TYPE-SINT32    / CIM-TYPE-UINT32 /
   CIM-TYPE-SINT64    / CIM-TYPE-UINT64 / CIM-TYPE-REAL32 /
   CIM-TYPE-REAL64    / CIM-TYPE-BOOLEAN /
   CIM-TYPE-STRING    / CIM-TYPE-DATETIME /
   CIM-TYPE-REFERENCE / CIM-TYPE-CHAR16 /
   CIM-TYPE-OBJECT

CimArrayType = CIM-ARRAY-SINT8 / CIM-ARRAY-UINT8 /
   CIM-ARRAY-SINT16   / CIM-ARRAY-UINT16 /
   CIM-ARRAY-SINT32   / CIM-ARRAY-UINT32 /
   CIM-ARRAY-SINT64   / CIM-ARRAY-UINT64 /
   CIM-ARRAY-REAL32   / CIM-ARRAY-REAL64 /
   CIM-ARRAY-BOOLEAN  / CIM-ARRAY-STRING /
   CIM-ARRAY-DATETIME / CIM-ARRAY-REFERENCE /
   CIM-ARRAY-CHAR16   / CIM-ARRAY-OBJECT

CimArrayFlag = %x20 %x00                    ; 0x2000 bit flag
```

The CimType is a 16-bit encoding unit that always contains a CimBaseType and an optional CimArrayFlag. If the type is actually an array type, the CimBaseType MUST be combined by using the bitwise OR operation with the CimArrayFlag value (0x2000) that results in the most significant octet containing 0x20 and the lower octet containing the value of the CimBaseType.

For example, to encode an array of CIM-TYPE-STRING, the CimType binary encoding would be 0x2008, in which the upper octet indicates that an array is being encoded, and the lower octet indicates that the array is of type CIM-TYPE-STRING.

The bit values for the individual types are constants from the following table. These are mutually exclusive to each other, but all (except CIM-TYPE-EMPTY and CIM-TYPE-ILLEGAL) are compatible with the CimArrayFlag value. CIM-TYPE-EMPTY and CIM-TYPE-ILLEGAL cannot be combined with CimArrayFlag. Note that the values are defined in decimal.

```
CIM-TYPE-EMPTY = %d0
```

```
CIM-TYPE-ILLEGAL = %d4095
CIM-TYPE-SINT8 = %d16
CIM-TYPE-UINT8 = %d17
CIM-TYPE-SINT16 =%d2
CIM-TYPE-UINT16 =%d18
CIM-TYPE-SINT32 = %d3
CIM-TYPE-UINT32 = %d19
CIM-TYPE-SINT64 = %d20
CIM-TYPE-UINT64 = %d21
CIM-TYPE-REAL32 = %d4
CIM-TYPE-REAL64 = %d5
CIM-TYPE-BOOLEAN = %d11
CIM-TYPE-STRING = %d8
CIM-TYPE-DATETIME = %d101
CIM-TYPE-REFERENCE = %d102
CIM-TYPE-CHAR16 = %d103
CIM-TYPE-OBJECT = %d13
```

Each base type can be combined with the array bit (0x2000), which results in an array of that base type. CimArrayType values are as follows.

```
CIM-ARRAY-SINT8 = %d8208
CIM-ARRAY-UINT8 = %d8209
CIM-ARRAY-SINT16 =%d8194
CIM-ARRAY-UINT16 =%d8210
CIM-ARRAY-SINT32 = %d8195
CIM-ARRAY-UINT32 = %d8201
CIM-ARRAY-SINT64 = %d8202
CIM-ARRAY-UINT64 = %d8203
CIM-ARRAY-REAL32 = %d8196
CIM-ARRAY-REAL64 = %d8197
CIM-ARRAY-BOOLEAN = %d8203
CIM-ARRAY-STRING = %d8200
CIM-ARRAY-DATETIME = %d8293
CIM-ARRAY-REFERENCE = %d8294
CIM-ARRAY-CHAR16 = %d8295
CIM-ARRAY-OBJECT = %d8205
```

3 Special Data Type Encodings

The various CIM data types have special binary encodings implied by the ABNF rules specified in sections 2.2.70 and 2.2.77. However, there are three special cases that require further techniques: the CIM DateTime type, CIM reference types, and the encoding of method signatures for CIM methods. These encodings affect only the format of the values, and do not introduce new binary level encoding rules.

3.1 CIM DateTime Type

The CIM DateTime type is a string with the specific format specified in [DMTF-DSP004] section 2.2.1.

Because DateTime types are strings, a provision is included in the encoding to ensure that they can be distinguished semantically.

Any datetime value type:

- MUST be encoded as an Encoded-String, as specified in the ABNF.
- MUST contain a CIM qualifier whose name is CIMTYPE, whose type is string, and whose value is datetime.

If the CIM qualifier is omitted, the system MUST treat the DateTime as a standard string.

3.2 CIM Reference Types

A CIM reference type is a string containing the CIM object path to another CIM object, as specified in [DMTF-DSP004] section 5.3.2ff. It is essentially a pointer type allowing CIM objects to reference one another.

Because references are encoded as strings, a provision is included in the encoding to ensure that they can be distinguished semantically.

Any reference type:

- MUST be encoded as an Encoded-String, as specified in the ABNF, and must conform to the CIM object reference syntax, as specified in [DMTF-DSP004] section 5.3.2.
- MUST contain a CIM qualifier whose name is CIMTYPE, whose type is string, and whose value is "ref:<cimClass>" where "<cimClass>" MUST be the name of the CIM class that is being referenced. If the reference is untyped, "<cimClass>" MUST be set to the string value of "ref:object".

The CIMTYPE CIM qualifier MUST be specified.

3.3 CIM Methods

The method signature (that is, the return type) input parameters and output parameters are encoded using embedded CIM object encodings. Methods are as specified in [DMTF-DSP004] section 4.9, and are specified syntactically in the methodDeclaration rule, as specified in [DMTF-DSP004] appendix A.

The method signature consists of two embedded CIM objects of a CIM class called __PARAMETERS. Within the embedded objects, the parameters appear as properties. The parameter name as it

appears in the method is the CIM property name, and the type of the parameter is the CIM property type.

This is illustrated in the following example:

```
class MyClass2  :  MyClass
{
  [execute, performance={"fast", "sideffects"}]
  uint32 Restart([in] string ServiceName, [out] int32 Status);
}
```

In the CIM class example above, there is a method called Restart. The parameters are encoded in the same way as other CIM class definitions. Each method definition contains two CIM class definitions: one for the input parameters and one for the output parameters. These classes always have the same name: __PARAMETERS, but reflect the parameters of the current method, which is being encoded, so there is no immutable definition for the class. In this example, the two CIM class definitions would appear as follows.

```
[abstract]
class __PARAMETERS
{
  [in, ID(0)] string ServiceName;
}
[abstract]

class __PARAMETERS
{
 [out, ID(1)] sint32 Status
 uint32 ReturnValue;
}
```

Remarks

- Both CIM class definitions MUST be marked with an abstract qualifier. The first CIM class definition is used to package any in parameters to the method, and the second is used to package any out parameters.
- There is one definition to contain all in parameters (regardless of where they appear in the method signature), and one definition that encodes all out parameters and the return value.
- The order of declaration in these classes is the order in which the parameters appear. Because the parameters appear in an explicit order in the Managed Object Format (MOF) signature but are split between two separate CIM class definitions in the encoding, an ID attribute is added for each parameter, which represents the ordinal position of that parameter in the original signature.
- The return value, which has no name in the CIM method declaration, does have an explicit name in the output CIM class definition and is always ReturnValue. Because of this reserved name, a method cannot explicitly contain ReturnValue as a named parameter.
- The __PARAMETERS CIM class is not a true CIM class because the format changes for each method, and the format is not separately usable as a real CIM class definition. It is just a way of

legally reusing the encoding mechanism for classes. Because classes require names, __PARAMETERS is nothing more than that name.

A method encoding thus contains two apparent CIM class definitions (in the InputSignature and OutputSignature rules in the ABNF) that encode the parameters for the method.

Any qualifiers on the individual parameters become qualifiers on the properties of those names within the __PARAMETERS CIM class definition.

3.4 Heap Encoding

In general, HeapItems within the Heap occur in any order as long as the **fixups** to them (that is, any rules that reduce to HeapRef) are correct.

PropertyInfo blocks, for example, occur in an order different than the lexical order of the properties, and Encoded-String occurs at any location. This implementation of the Heap is intended to allow for best-fit algorithms when doing updates. Strings that fit into the original Encoded-String, even if they are shorter than the original strings, SHOULD be written into the old location. However, it is not an error if each new string update is written into a new location in the Heap.

Because the Heap is loosely organized, garbage space is inevitably created, and the Heap becomes fragmented. There are sequences of octets within the Heap that have no corresponding references to them by any HeapRef, and there may be large sequences of NULL octets near the end of the Heap. This situation is permitted to enable garbage collection algorithms and easy reuse of large blocks without having to perform HeapRef fixups after each operation, which changes the Heap. Encoders with such garbage collectors MAY transmit encoded objects without previously performing garbage collection. Decoder implementations MUST be prepared to deal with the presence of Heaps that have not been garbage collected.

This is of importance in decoding, as code that processes the Heap and HeapItems SHOULD NOT fault if it encounters blocks with no reference to them or garbage octets at the end of the Heap.

The client MUST NOT alter a CIM class definition, including its Heap, once instances for it have been created and are in use. A client MAY only alter a ClassHeap or a MethodHeap when creating or updating a CIM class definition for which no instances currently exist. This is because copied images of the ClassPart are made for CIM instances as part of their encoding. CIM objects that have this image altered MUST be rejected by the server.

4 Encoded Examples

This section illustrates a simple example of the binary encoding for a simple CIM class definition and its instances. The MOF textual representation is used, as specified in [DMTF-DSP004].

CIM class Base:

```
{
      [key]
      sint32  Id;

}
[Description("MyClass Example")]
class MyClass  :  Base
{
      [read, write]
      string   Data1;

      string   Data2 = "defaultValue";

uint32 Array[ ];
}
```

This is a simple CIM class hierarchy of two classes: a Base CIM class and a derived CIM class called MyClass. The values in square brackets are metadata items called qualifiers; and the individual fields are called properties and are identical to member variables, properties, or fields of popular object-oriented programming languages such as C++, C#, and Java.

The binary encoding is presented for both classes: first Base, and then MyClass. Since each CIM class contains the encoding of itself and its base class, this encoding illustrates all of the concepts involved in encoding classes.

The raw hexadecimal encoding of Base is as follows:

```
1)  78 56 34 12 D0 00 00 00
2)  05 00 44 50 52 41 56 41 54 2D 44 45
3)  56 00 00 52 4F 4F 54 00 1D 00 00 00 00 FF FF FF
4)  FF 00 00 00 00 04 00 00 00 04 00 00 00 00 00 00
5)  00 00 00 00 80
6)  0C 00 00 00 00 00 00 00 00 00 00 80
7)  66 00 00 00 00 00 00 00 00 05 00 00 00 04 00
8)  00 00 04 00 00 00 01 00 00 00 06 00 00 00 0A 00
9)  00 00 05 FF FF FF FF 3C 00 00 80 00 42 61 73 65
10) 00 00 49 64 00 03 00 00 00 00 00 00 00 00 00 00
11) 00 00 00 1C 00 00 00 0A 00 00 80 03 08 00 00 00
12) 34 00 00 00 01 00 00 80 13 0B 00 00 00 FF FF 00
13) 73 69 6E 74 33 32 00 0C 00 00 00 00 00 34 00 00
14) 00 00 80 00 80 13 0B 00 00 00 FF FF 00 73 69 6E
15) 74 33 32 00
```

For the EncodingLength, see the note in section 2.2.1. The above sample EncodingLength value is 0xD0, which is larger than the actual required number of octets.

The following table decodes Base using the ABNF.

Relevant offset	Octet values	Comments
		EncodingUnit.
	78 56 32 12	Standard CIM object Signature (line 1).
	D0 00 00 00	EncodingUnit::EncodingLength UINT32 length of entire CIM class encoding (0xD0, 208 decimal octets).
		ObjectBlock.
	05	Decoration (line 2, shaded octet). Binary = 00000101. Bit 0 set == this is a CIM class definition (not a CIM instance). Bit 2 set == this CIM object is decorated with a server and CIM namespace name.
	00 44 50 52 41 56 41 54 2D 44 45 56 00	DecServerName (Encoded-String) on lines 2–3 that is the server name decoration indicating what machine on the network this CIM object originated from. The first octet indicates that this string is encoded in ANSI 8-bit characters, not 16-bit UNICODE, and the value is DPRAVAT-DEV followed by an 8-bit NULL terminator, the last octet.
	00 52 4F 4F 54 00	DecNamespace (Encoded-String) that indicates what CIM namespace the CIM object originates from. The first octet indicates that this string is encoded in ANSI 8-bit characters, not 16-bit UNICODE, and the value is ROOT, followed by an 8-bit NULL terminator, the last octet.
		Encoding.
	1D 00 00 00	ParentClass::ClassPart::ClassHeader::EncodingLength. This is the length in octets of the encoding unit, or 0x1D octets.
	00	ReservedOctet (shaded octet line 3). This must be zero.
	FF FF FF FF	ClassNameRef. This value indicates that there is no parent CIM class name since Base is the basest class.
	00 00 00 00	ValueTableLength (italics line 4). This is zero, indicating there is no value table for the parent class because there is no parent CIM class to Base.
	04 00 00 00	DerivationList. This indicates the length of the list of superclasses to this class. Because the

Relevant offset	Octet values	Comments
		list only consists of the EncodingLength UINT32, it is four octets. The ClassNameEncoding list is empty.
	04 00 00 00	ClassQualifierSet::EncodingLength. There is no CIM qualifier set for the base CIM class of Base because it has no base, so this CIM qualifier set is empty and consists of the length only of the EncodingLength UINT32, which is four octets.
	00 00 00 00	PropertyLookupTable::PropertyCount (shaded octets, end of line four, beginning of line five). There are zero properties in the base CIM class (which does not exist because Base is the basest class).
		NdTable and ValueTable are empty and contain no octets because there are no properties.
	00 00 00 80	ClassHeap::Heap::HeapLength. Heaps are always prefixed by their length in 31 bits with the MS bit set. This indicates a zero-length Heap.
	0C 00 00 00	MethodPart::EncodingLength. There are 12 octets in the encoding of the method table.
	00 00	MethodPart::MethodCount. 16-bit value indicating how many methods there are in the class, or zero.
	00 00	MethodPart::Padding. This is padding, and can be zero or any random value.
	00 00 00 80	MethodPart::MethodHeap. The heap length of the method heap, or zero. The MS bit is always set on HeapLength values, and only 31 bits are significant. This is located on the shaded portion of line 6.
		At this point, the non-existent ParentClass ends, and CurrentClass begins, which is where Base is specified.
	66 00 00 00	ClassHeader::EncodingLength, indicating that this encoding for Base is 0x66 octets in length (102 decimal). Beginning of line 7.
	00	ReservedOctet.
	00 00 00 00	ClassHeader::ClassNameRef. The offset into the heap of the CIM class name Base. The CIM class name is the first item in the heap, or an Encoded-String with the value of Base.
	05 00 00 00	ClassHeader::ValueTableLength. The value table is five octets in length.

Relevant offset	Octet values	Comments
	04 00 00 00	DerivationList::EncodingLength. There is no list of superclasses because Base is the basest class. The encoding length is just four octets, which is the length of the EncodingLength.
	04 00 00 00	ClassQualifierSet::EncodingLength. There are no class-level qualifiers on this class, so the encoding length for the set of qualifiers is just the length of the EncodingLength field, or four octets.
	01 00 00 00	PropertyLookupTable::PropertyCount. There is one CIM property in this CIM class (italic line 8).
		PropertyLookupTable::PropertyLookup.
	06 00 00 00	PropertyNameRef. The location in the heap of the Encoded-String of the CIM property name, or offset 6 into the heap.
	A0 00 00 00	PropertyInfoRef. The location in the heap of the PropertyInfo.
	05	NdTable. This has the binary value 00000101. Note that because there is only one property, only the two least-significant bits are valid, and the other bits may be any value. In this case, the bit value 01 indicates the property has a default value of NULL, but the default is not inherited from a superclass.
	FF FF FF FF	EncodedValue::NoValue. No value defined for the CIM property by default in the CIM class definition.
	3C 00 00 80	ClassHeap::HeapLength.
Heap Offset 0	00 42 61 73 65 00	Encoded-String Base. The first octet indicates ANSI encoding, and the last octet is the null terminator.
Heap Offset 6	00 49 64 00	Encoded-String ID, the name of the property.
	03 00 00 00	PropertyInfo::PropertyType. CIM-TYPE-SINT32 == 3.
	00 00	PropertyInfo::DeclarationOrder.
	00 00 00 00	PropertyInfo::ValueTableOffset. Offset in ValueTable of default value, or zero.

Relevant offset	Octet values	Comments
	00 00 00 00	PropertyInfo::ClassOfOrigin. A value of zero indicates the current class.
	1C 00 00 00	PropertyQualifierSet::EncodingLength. There are 1C octets of encoding for the QualifierSet for this property.
	0A 00 00 80	Qualifier::QualifierName. This is a DictionaryReference instead of a plain HeapRef because the MS bit is set (0x80). The dictionary reference is cimtype.
	03	Qualifier::QualifierFlavor. Bits 00000011. Bit 0 indicates that the CIM qualifier is a SYSTEM CIM qualifier (key). Bit 1 indicates that the CIM qualifier should be propagated to derived classes.
	08 00 00 00	Qualifier::QualifierType. This indicates CIM-TYPE-STRING, which is the data type of the CIM qualifier.
	34 00 00 00	Qualifier::QualifierValue. This is an EncodedValue, depending on the type in the previous field. Because the CIM qualifier type is CIM-TYPE-STRING, this value is the HeapRef to an Encoded-String.
		Another CIM qualifier follows for the CIM property because all of the octets in PropertyQualifierSet::EncodingLength have not been used completely yet.
	01 00 00 80	Qualifier::QualifierName. This is a DictionaryReference instead of a plain HeapRef because the MS bit is set (0x80). The dictionary reference is key.
	13	QualifierFlavor.
	0B 00 00 00	QualifierType. CIM-TYPE-BOOLEAN.
	FF FF	EncodedValue for the CIM qualifier that is BOOL. The value in this case is logical TRUE (all bits set).
Offset 0x34	00 73 69 6E 74 33 32 00	Encoded-String with a value of sint32. This is the value of the first CIM qualifier in this table.
		At this point, the Heap for the CIM class has encoded because all of its EncodingLength has been used. The MethodsPart for the CIM class now begins.

Relevant offset	Octet values	Comments
	0C 00 00 00	MethodsPart::EncodingLength, or 12 octets.
	00 00	MethodsPart::MethodCount, or zero methods.
	34 00	MethodsPart::MethodCountPadding. Any two octets with random values.
	00 00 00 80	MethodsPart::MethodHeap::Heap::HeapLength. This is a zero-length heap, indicating no methods. The MS bit is always set for HeapLength values.
	Remaining Octets	The remaining octets are not significant. Also see EncodingUnit (section 2.2.1).

The raw hexadecimal encoding of MyClass is as follows.

```
78 56 34 12 2E 02 00 00 05 00 44 50
52 41 56 41 54 2D 44 45 56 00 00 52 4F 4F 54 00
66 00 00 00 00 00 00 00 00 05 00 00 00 04 00 00
00 04 00 00 00 01 00 00 00 06 00 00 00 0A 00 00
00 05 FF FF FF FF 3C 00 00 80 00 42 61 73 65 00
00 49 64 00 03 00 00 00 00 00 00 00 00 00 00 00
00 00 1C 00 00 00 0A 00 00 80 03 08 00 00 00 34
00 00 00 01 00 00 80 13 0B 00 00 00 FF FF 00 73
69 6E 74 33 32 00 0C 00 00 00 00 00 34 00 00 00
00 80 76 01 00 00 00 00 00 00 00 11 00 00 00 0E
00 00 00 00 42 61 73 65 00 06 00 00 00 11 00 00
00 09 00 00 00 00 08 00 00 00 16 00 00 00 04 00
00 00 27 00 00 00 2E 00 00 00 55 00 00 00 5C 00
00 00 99 00 00 00 A0 00 00 00 C7 00 00 00 CB 00
00 00 47 FF FF FF FF FF FF FF FF FD 00 00 00 FF
FF FF FF 11 01 00 80 00 4D 79 43 6C 61 73 73 00
00 44 65 73 63 72 69 70 74 69 6F 6E 00 00 4D 79
43 6C 61 73 73 20 45 78 61 6D 70 6C 65 00 00 41
72 72 61 79 00 13 20 00 00 03 00 0C 00 00 00 01
00 00 00 11 00 00 00 0A 00 00 80 03 08 00 00 00
4D 00 00 00 00 75 69 6E 74 33 32 00 00 44 61 74
61 31 00 08 00 00 00 01 00 04 00 00 00 01 00 00
00 27 00 00 00 0A 00 00 80 03 08 00 00 00 91 00
00 00 03 00 00 80 00 0B 00 00 00 FF FF 04 00 00
80 00 0B 00 00 00 FF FF 00 73 74 72 69 6E 67 00
00 44 61 74 61 32 00 08 00 00 00 02 00 08 00 00
00 01 00 00 00 11 00 00 00 0A 00 00 80 03 08 00
00 00 BF 00 00 00 00 73 74 72 69 6E 67 00 00 49
64 00 03 40 00 00 00 00 00 00 00 00 00 00 00 00
1C 00 00 00 0A 00 00 80 23 08 00 00 00 F5 00 00
00 01 00 00 80 33 0B 00 00 00 FF FF 00 73 69 6E
74 33 32 00 00 64 65 66 61 75 6C 74 56 61 6C 75
65 00 00 00 00 00 00 00 0C 00 00 00 00 00 00 73
00 00 00 80 32 00 00 64 65 66 61 75 6C 74 56 61
6C 75 65 00 00 00 00 00 00 00 00 00 00 00 00 00
```

```
00 00 00 00 00 80 00 00 00 00
```

This encoding is decoded using the ABNF in the following table. Note that the ParentClass part of this encoding is the same as the encoding of the CurrentClass part of the Base CIM class encoding.

Relevant offset	Octet values	Comments
		EncodingUnit.
	78 56 32 12	Signature.
	23 02 00 00	EncodingUnit::EncodingLength UINT32 length of entire CIM class encoding (0x223 octets).
		ObjectBlock.
	05	Decoration. Bit 0 set == this is a CIM class definition. Bit 2 set == this CIM object is decorated with a server and CIM namespace name.
		Decoration.
	00 44 50 52 41 56 41 54 2D 44 56 00	DecServerName. This is an Encoded-String containing the name of the server that transmitted the CIM object DPRAVAT-DEV.
	00 52 4F 4F 54 00	DecNamespace. This is an Encoded-String containing the CIM namespace. The CIM object was created from ROOT.
		Encoding for derived CIM class. Base is appended at this location. This is the ClassType::ParentClass block.
	66 00 00 00	ParentClass::ClassPart::ClassHeader::EncodingLength. UINT32 length of ClassPart for the Base class, 0x66 octets. A separate binary chunk for MyClass follows, appended to this CIM class definition.
	00	ReservedOctet.
	00 00 00 00	ClassNameRef. Offset to CIM class name in the ClassHeap.
	05 00 00 00	ValueTableLength. Length of ValueTable for properties (five octets).

Relevant offset	Octet values	Comments
	04 00 00 00	DerivationList length, including the length of this UINT32. Because this is four octets and all four are used completely with this value, there is no derivation list.
	04 00 00 00	ClassQualifierSet::EncodingLength, including the length of this UINT32. Because all four octets are used completely with this value, there is no QualifierSet.
	01 00 00 00	PropertyLookupTable::PropertyCount. This is the CIM property count, not the length in octets.
	06 00 00 00	PropertyLookup::PropertyNameRef. Name of CIM property [0] in the form of an Encoded-String. The 0x0006 offset is from the beginning of the ClassHeap, not from the beginning of this packet.
	0A 00 00 00	PropertyLookup::PropertyInfoRef. Offset for PropertyInfo for Property[0], including any qualifiers. This is the offset from the beginning of the ClassHeap, not the beginning of this packet.
		NdTable.
	05	This has the binary value 00000101. Note that because there is only one property, only the two least-significant bits are valid, and the other bits may be any value. In this case, the bit value 01 indicates the property has a default value of NULL, but the default is not inherited from a superclass.
		ValueTable.
	FF FF FF FF	ValueTable::EncodedValue. There is only one CIM property, and this reserved value indicates there is no default value.
	3C 00 00 80	ClassHeap::HeapLength. Length of heap is 0x3C octets. MS bit is set to 1 for all encodings.
0x0	00 42 61 73 65 00	Encoded-String Base, which is the name of the class.
0x6	00 49 64 00	Encoded-String Property[0] name, which is Id.
0x10	03 00 00 00	PropertyInfo::PropertyType. CIM-TYPE-SINT32 == 3.
0x14	00 00	PropertyInfo::DeclarationOrder. This is the 0th property.
0x16	00 00 00 00	PropertyInfo::ValueTableOffset.

Relevant offset	Octet values	Comments
0x1A	00 00 00 00	PropertyInfo::ClassOfOrigin. CIM class of origin in DerivationList, or the 0th CIM class (this class).
	1C 00 00 00	PropertyQualifierSet::EncodingLength (0x1C octets in length, including itself).
	0A 00 00 80	Qualifier::QualifierName. CimType dictionary entry encoding, signaled by MS bit and the 0xA dictionary entry. This indicates that the current CIM qualifier is the CIMTYPE qualifier, which must be attached to every property.
	03	QualifierSet::QualifierFlavor. Here the octet is 03. This means that the bit 0 and bit 1 in the octet are set. Bit 0 set == Propagate to instances. Bit 1 set == Propagate to Derived Classes.
	08 00 00 00	Qualifier::QualifierType. CimType of CIM qualifier value 0x8 == CIM-TYPE-STRING.
	34 00 00 00	Qualifier::QualifierValue. HeapRef to the string of the CIM qualifier value sint32.
	01 00 00 80	Qualifier::QualifierName. This is a DictionaryReference instead of a plain HeapRef because the MS bit is set (0x80). The dictionary reference is key.
	13	QualifierFlavor.
	0B 00 00 00	QualifierType. CIM-TYPE-BOOLEAN.
	FF FF	EncodedValue for the CIM qualifier, which is BOOL. The value in this case is logical TRUE (all bits set).
0x34	00 73 69 6E 74 33 32 00	Encoded-String sint32.
	0C 00 00 00	Length of MethodsPart, including itself.
	00 00	MethodCount (zero methods).
	34 00	Two octets of MethodPadding; any values are legal.

Relevant offset	Octet values	Comments
	00 00 00 80	MethodHeap::Heap::HeapLength. This is the length of Heap. Zero, with MS bit set, as for all heaps.
		Encoding for derived CIM class MyClass is appended at this location. This is the ClassType::CurrentClass block.
	76 01 00 00	ClassHeader::EncodingLength. Length (0x176 octets).
	00	Reserved. Must be zero.
	00 00 00 00	ClassNameRef to CIM class name. This is relative to the upcoming heap for this class, not the previous heap for Base.
	11 00 00 00	ValueTableLength.
	0E 00 00 00	DerivationList length in octets, including itself.
	00 42 61 73 65 00	Encoded-String Base, which is the superclass to this class.
	06 00 00 00	EncodingLength of previous string, or six octets (includes both leading flag and trailing NULL).
	11 00 00 00	ClassQualifierSet::EncodingLength length in octets, including itself.
	09 00 00 00	QualifierName, Heap offset to Encoded-String.
	00	QualifierFlavor. 0 == Local.
	08 00 00 00	QualifierType. 0x8 == CIM-TYPE-STRING.
	16 00 00 00	QualifierValue. HeapRef to Encoded-String that is the value of the qualifier.
	04 00 00 00	PropertyLookupTable::PropertyCount. There are four properties in this class. The properties are sorted within this table, regardless of the order they appear in the current CIM class and any of its superclasses. This enables a binary search to be performed while locating properties by name.
	27 00 00 00	PropertyInfo::PropertyNameRef. Offset in heap to CIM property name. Points to the Encoded-String array.

Relevant offset	Octet values	Comments
	2E 00 00 00	PropertyInfo::PropertyInfoRef. Offset in heap of PropertyInfo table and any associated qualifiers for the property.
	55 00 00 00	PropertyInfo::PropertyNameRef. Offset in heap to CIM property name. Points to the Encoded-String Data1.
	5C 00 00 00	PropertyInfo::PropertyInfoRef. Offset in heap of PropertyInfo for Data1 and any associated qualifiers.
	99 00 00 00	PropertyInfo::PropertyNameRef. Offset in heap of CIM property name. Points to Data2.
	A0 00 00 00	PropertyInfo::PropertyInfoRef. Offset in heap of PropertyInfo for Data2 and any associated qualifiers.
	C7 00 00 00	PropertyInfo::PropertyNameRef. Offset in heap of CIM property name. Points to Id. All properties inherited from base classes are repeated within the PropertyLookupTable for each derived class.
	CB 00 00 00	PropertyInfo::PropertyInfoRef. Offset in heap of PropertyInfo for Id and any associated CIM qualifier sets.
	47	NdTable. 01 00 01 11b. Property 0 == 11b NULL, inherits DEFAULT. Property 1 == 01 NULL, no inherited default. Property 2 == 00 Not NULL, no inheritance. Property 3 == 01 Null, no inherited default. The indexes do not refer to the ordinal position in PropertyLookup, but refer to the propertyIndex field for the CIM property in the PropertyInfo table for that property.
		ValueTable.
0x0	FF FF FF FF	No default value.
0x4	FF FF FF FF	No default value.
0x8	FD 00 00 00	HeapRef to default value.
0xC	FF FF FF FF	No default value.
		ClassHeap::HeapLength. The length is 0x111 octets, and the MS bit is always set.
0x0	00 4D 79 43 6C 61	Encoded-String MyClass.

Relevant offset	Octet values	Comments
	73 73 00	
0x9	00 44 65 73 63 72 69 70 74 69 6F 6E 00	Encoded-String Description.
0x16	00 4D 79 43 6C 61 73 73 20 45 78 61 6D 70 6C 65 00	Encoded-String MyClass Example.
0x27	00 41 72 72 61 79 00	Encoded-String Array.
0x2E		PropertyInfo for Array property.
	13 20 00 00	PropertyType CIM-TYPE-UINT32 and CimArrayFlag.
	03 00	DeclarationOrder (starting with 0). Array was the 3rd CIM property after Id, Data1, and Data2. This is the value used in NdTable.
	0C 00 00 00	ValueTableOffset Offset into ValueTable for default CIM property value. In this case, the offset points to 0xFFFFFFFF, which means there is no default value.
	01 00 00 00	ClassOfOrigin. Class 1 in DerivationList array.
	11 00 00 00	PropertyQualifierSet::EncodingLength, including itself.
	0A 00 00 80	Qualifier::QualifierName. This is a DictionaryReference instead of a plain HeapRef because the MS bit is set (0x80). The dictionary reference is cimtype.
	03	QualifierFlavor.
	08 00 00 00	QualifierType, which is CIM-TYPE-STRING.
	4D 00 00 00	QualifierValue. Because the type is string, the value is a HeapRef.

Relevant offset	Octet values	Comments
0x4D	00 75 69 6E 74 33 32 00	Encoded-String uint32.
0x55	00 44 61 74 61 31 00	Encoded-String Data1.
0x5C		PropertyInfo.
	08 00 00 00	PropertyInfo::PropertyType, CIM-TYPE-STRING == 0x8.
	01 00	PropertyInfo::DeclarationOrder, zero-origin. Original order was {Id, Data1, Data2, Array}, so this is Property[1].
	04 00 00 00	PropertyInfo::ValueTableOffset. In this case, at that offset is the value 0xFFFFFFFF, which means there is no default.
	01 00 00 00	PropertyInfo::ClassOfOrigin. 1 == Current class.
0x6A		PropertyQualifierSet for Data1.
	27 00 00 00	EncodingLength of CIM qualifier set in octets, including itself.
	0A 00 00 80	Qualifier::QualifierName. This is a DictionaryReference instead of a plain HeapRef because the MS bit is set (0x80). The dictionary reference is cimtype.
	03	QualifierFlavor.
	08 00 00 00	QualifierType or CIM-TYPE-STRING.
	91 00 00 00	QualifierValue::EncodedValue, offset to value in current Heap.
		Another CIM qualifier for the current PropertyInfo encoding.
	03 00 00 80	Qualifier::QualifierName. This is a DictionaryReference instead of a plain HeapRef because the MS bit is set (0x80). The dictionary reference is read.

Relevant offset	Octet values	Comments
	00	QualifierFlavor.
	0B 00 00 00	QualifierType is CIM-TYPE-BOOLEAN.
	FF FF	QualifierValue::EncodedValue. This is the encoding for logical TRUE when type is CIM-TYPE-BOOLEAN. 0x0000 is FALSE.
	04 00 00 80	Qualifier::QualifierName. This is a DictionaryReference instead of a plain HeapRef because the MS bit is set (0x80). The dictionary reference is write.
	00	QualifierFlavor. No propagate.
	0B 00 00 00	QualifierType is CIM-TYPE-BOOLEAN.
	FF FF	QualifierValue::EncodedValue. This is the encoding for logical TRUE when type is CIM-TYPE-BOOLEAN. 0x0000 is FALSE.
0x91	00 73 74 72 69 6E 67 00	Encoded-String string.
0x99	00 44 61 74 61 32 00	Encoded-String Data2.
		PropertyInfo.
0xA0	08 00 00 00	PropertyInfo::PropertyType Type is CIM-TYPE-STRING.
	02 00	PropertyInfo::DeclarationOrder is 2 out of {0, 1, 2, 3}.
	08 00 00 00	PropertyInfo::ValueTableOffset for default value. This points to a HeapRef of 0xFD, which in turn points to DefaultValue.
	01 00 00 00	PropertyInfo::ClassOfOrigin points to class[1] in the derivation chain.
		PropertyQualifierSet.

Relevant offset	Octet values	Comments
	11 00 00 00	QualifierSet::EncodingLength. CIM qualifier set is 0x11 octets in length, including itself.
	0A 00 00 80	Qualifier::QualifierName. This is a DictionaryReference instead of a plain HeapRef because the MS bit is set (0x80). The dictionary reference is cimtype.
	03	Qualifier::Flavor, propagate to subclass and instance.
	08 00 00 00	QualifierType is CIM-TYPE-STRING.
	BF 00 00 00	QualifierValue::EncodedValue HeapRef to value.
0xBF	00 73 74 72 69 6E 67 00	Encoded-String of string.
0xC7	00 49 64 00	Encoded-String of Id.
		PropertyInfo.
0xCB	03 40 00 00	PropertyInfo::PropertyType. CIM-TYPE-SINT32 + INHERITED.
	00 00	PropertyInfo::DeclarationOrder, CIM property number 0.
	00 00 00 00	PropertyInfo::ValueTableOffset Default value in ValueTable.
	00 00 00 00	PropertyInfo::ClassOfOrigin. CIM class 0 in DerivationList.
		PropertyInfo::PropertyQualifierSet.
	1C 00 00 00	QualifierSet::EncodingLength in octets, including itself.
	0A 00 00 80	Qualifier::QualifierName. This is a DictionaryReference instead of a plain HeapRef because the MS bit is set (0x80). The dictionary reference is cimtype.

Relevant offset	Octet values	Comments
	23	Qualifier::Flavor, inherited and toclass+toinstance.
	08 00 00 00	Qualifier::QualifierType is CIM-TYPE-STRING.
	F5 00 00 00	Qualifier::QualifierValue::Encoded-Value, a HeapRef to value.
	01 00 00 80	Qualifier::QualifierName. This is a DictionaryReference instead of a plain HeapRef because the MS bit is set (0x80). The dictionary reference is key.
	33	Qualifier::QualifierFlavor = Not overridable/propagated/toclass/toinst.
	0B 00 00 00	Qualifier::QualifierType is CIM-TYPE-BOOLEAN.
	FF FF	Qualifier::QualifierValue::EncodedValue Default value.
0xF5	00 73 69 6E 74 33 32 00	Encoded-String sint32.
0xFD	00 64 65 66 61 75 6C 74 56 61 6C 75 65 00	Encoded-String defaultValue.
0x10B	00 00 00 00 00 00	MethodsPart.
	0C 00 00 00	MethodsPart::EncodingLength, including itself.
	00 00	MethodsPart::MethodCount (no methods in this case).
	00 73	MethodsPart::Padding. Two octets of padding; can be any value.
	00 00 00 80	MethodsPart::MethodHeap::HeapLength zero length method heap.
		END OF OBJECT

4.1 Instance Encoding

Using the above CIM class as a basis, the following CIM instance encoding is presented.

```
instance of MyClass
{
   Id = 123;
   Data1 = "StringField";
   Array = { 1, 2, 3 }
}
```

The raw hexadecimal encoding of this CIM instance follows.

```
78 56 34 12 D3 01 00 00 06 00 44 50
52 41 56 41 54 2D 44 45 56 00 00 52 4F 4F 54 00
76 01 00 00 00 00 00 00 00 11 00 00 00 0E 00 00
00 00 42 61 73 65 00 06 00 00 00 11 00 00 00 09
00 00 00 00 08 00 00 00 16 00 00 00 04 00 00 00
27 00 00 00 2E 00 00 00 55 00 00 00 5C 00 00 00
99 00 00 00 A0 00 00 00 C7 00 00 00 CB 00 00 00
47 FF FF FF FF FF FF FF FF FD 00 00 00 FF FF FF
FF 11 01 00 80 00 4D 79 43 6C 61 73 73 00 00 44
65 73 63 72 69 70 74 69 6F 6E 00 00 4D 79 43 6C
61 73 73 20 45 78 61 6D 70 6C 65 00 00 41 72 72
61 79 00 13 20 00 00 03 00 0C 00 00 00 01 00 00
00 11 00 00 00 0A 00 00 80 03 08 00 00 00 4D 00
00 00 00 75 69 6E 74 33 32 00 00 44 61 74 61 31
00 08 00 00 00 01 00 04 00 00 00 01 00 00 00 27
00 00 00 0A 00 00 80 03 08 00 00 00 91 00 00 00
03 00 00 80 00 0B 00 00 00 FF FF 04 00 00 80 00
0B 00 00 00 FF FF 00 73 74 72 69 6E 67 00 00 44
61 74 61 32 00 08 00 00 00 02 00 08 00 00 00 01
00 00 00 11 00 00 00 0A 00 00 80 03 08 00 00 00
BF 00 00 00 00 73 74 72 69 6E 67 00 00 49 64 00
03 40 00 00 00 00 00 00 00 00 00 00 00 00 1C 00
00 00 0A 00 00 80 23 08 00 00 00 F5 00 00 00 01
00 00 80 33 0B 00 00 00 FF FF 00 73 69 6E 74 33
32 00 00 64 65 66 61 75 6C 74 56 61 6C 75 65 00
00 00 00 00 00 00 49 00 00 00 00 00 00 00 00 20
7B 00 00 00 19 00 00 00 00 00 00 00 09 00 00 00
04 00 00 00 01 26 00 00 80 00 4D 79 43 6C 61 73
73 00 03 00 00 00 01 00 00 00 02 00 00 00
03 00 00 00 00 53 74 72 69 6E 67 46 69
65 6C 64 00
```

The following table breaks apart this encoding using the ABNF. Note that the shaded part is the ClassPart encoding of the CIM instance and is identical to the table above for MyClass. Encoded instances always contain the CIM class definition encoding as the first part of the block. This allows a CIM instance to be decoded in its entirety without retrieving or cross-referencing with a CIM class definition.

The part of the encoding that differs from the MyClass encoding (and that is specific to the CIM instance) is not shaded, and is covered in the table below.

Relevant offset	Octet values	Comments
		EncodingUnit.
	78 56 32 12	Standard CIM object Signature.
	D3 01 00 00	UINT32 length of entire CIM class encoding (0x1D3 octets).
		ObjectBlock.
	06	ObjectBlock::decoration. Bit 1 set == this is a CIM instance definition. Bit 2 set == this CIM object is decorated with a server and CIM namespace name.
		Decoration.
	00 44 50 52 41 56 41 54 2D 44 56 00	Encoded-String containing the name of the server that transmitted the CIM object DPRAVAT-DEV.
	00 52 4F 4F 54 00	Encoded-String containing the CIM namespace. The CIM object was created from ROOT.
	All shaded octets	InstanceType::CurrentClass. This is a direct copy of the CIM class encoding for MyClass in the CurrentClass block.
	49 00 00 00	InstanceType::EncodingLength. 0x49 octets (73 decimal).
	00 00 00 00	InstanceType::InstanceClassName. Points to CIM class name in heap.
	00	InstanceType::Flags.
	20	InstanceType::NdTabl. 00100000. Indicates third CIM property has its default value.
	7B 00 00 00	InstanceType::InstanceData::ValueTable. Value for CIM property 0.
	19 00 00 00	InstanceType::InstanceData::ValueTable. Value for Data1.

Relevant offset	Octet values	Comments
	00 00 00 00	InstanceType::InstanceData::ValueTable. Data 2 has default value still.
	09 00 00 00	InstanceType::InstanceData::ValueTable. Location of array for Array.
	04 00 00 00	InstanceType::InstanceQualifierSet::EncodingLength. This indicates that there is no CIM qualifier set data because the EncodingLength only includes its own size.
*** See notes table below	01	InstPropQualSetFlag. There are no property-level qualifiers.
	26 00 00 80	InstanceHeap::HeapLength The Heap is 0x26 octets in length, and the MS bit is set, for all HeapLength values.
Heap offset 0	00 4D 79 43 6C 61 73 73 00	Encoded-String MyClass.
Heap offset 9	03 00 00 00	Encoded-Array::ArrayCount. There are three elements in the array.
	01 00 00 00	UINT32 [0].
	02 00 00 00	UINT32 [1].
	03 00 00 00	UINT32 [2].
HeapOffset 0x19	00 53 73 72 69 6E 67 46 69 65 6C 64 00	Encoded-String of value StringField.

***In the left column above, a special case can occur with instances that have qualifiers (see the InstancePropQualifierSet rule):

```
instance of MyClass
{
 Array = {1, 2, 3};
 [test] Data1 = "StringField";
 Id = 123;
};
```

The encoding for the above CIM instance must take into account that there is a property-level CIM qualifier appearing within the instance. Once any CIM qualifier appears on any CIM property at the CIM instance level, there must be an array of QualifierSet elements, one for each CIM property in the CIM class that the CIM instance belongs to, even if they are not used. The binary encoding for the above CIM instance differs from the previous example (starting at the row in the table above where the *** occurs in the left column).

	Octet values	Comments
***	02	InstPropQualSetFlag. If two, there is one QualifierSet per property encoded at this location prior to the InstanceHeap.
	04 00 00 00	QualifierSet: No qualifiers for CIM property 0, as the EncodingLength is four octets, the length of the EncodingLength value itself.
	0F 00 00 00	QualifierSet CIM property 1, there are 15 octets for this QualifierSet. This is for CIM property Data1.
	26 00 00 00	QualifierName: Heap reference to CIM qualifier name test.
	0B 00 00 00	QualifierType: CIM-TYPE-BOOL.
	FF FF	Logical TRUE.
	04 00 00 00	QualifierSet, none for CIM property 2 (Data2).
	04 00 00 00	QualifierSet, none for CIM property 3 (Id).
	2C 00 00 80	InstanceHeap::HeapLength. Heap is longer to accommodate the name of the CIM qualifier test.
	...	Rest of the Heap.

4.2 Class Encoding with Methods

Classes that contain methods have an extra encoding block called MethodsPart in the ABNF encoding. The MethodsPart only applies to CIM objects that are encoded as CIM class definitions, and are not part of a CIM instance encoding, even though instances do contain parts of the CIM class encoding related to CIM property definitions.

The example CIM class contains one method, Restart, which has input parameters, output parameters, and a uint32 return value. The CIM class is derived from MyClass (as specified in section 4.1).

```
class MyClass2  :  MyClass
```

```
{
  [execute, performance={"fast", "sideffects"}]
   uint32 Restart([in] string ServiceName, [out] sint32 Status);
}
```

The raw hexadecimal encoding of the above CIM class definition is shown below. The shaded block is the encoding of the parent CIM class MyClass, which is the same as shown in previous sections.

```
78 56 34 12 BE 08 00 00
05 00 44 50 52 41 56 41 54 2D 44 45 56 00
00 52 4F 4F 54 00
76 01 00 00 00 00 00 00 00 11 00 00 00 0E 00 00
00 00 42 61 73 65 00 06 00 00 00 11 00 00 00 09
00 00 00 00 08 00 00 00 16 00 00 00 04 00 00 00
27 00 00 00 2E 00 00 00 55 00 00 00 5C 00 00 00
99 00 00 00 A0 00 00 00 C7 00 00 00 CB 00 00 00
47 FF FF FF FF FF FF FF FF FD 00 00 00 FF FF FF
FF 11 01 00 80 00 4D 79 43 6C 61 73 73 00 00 44
65 73 63 72 69 70 74 69 6F 6E 00 00 4D 79 43 6C
61 73 73 20 45 78 61 6D 70 6C 65 00 00 41 72 72
61 79 00 13 20 00 00 03 00 0C 00 00 00 01 00 00
00 11 00 00 00 0A 00 00 80 03 08 00 00 00 4D 00
00 00 00 75 69 6E 74 33 32 00 00 44 61 74 61 31
00 08 00 00 00 01 00 04 00 00 00 01 00 00 00 27
00 00 00 0A 00 00 80 03 08 00 00 00 91 00 00 00
03 00 00 80 00 0B 00 00 00 FF FF 04 00 00 80 00
0B 00 00 00 FF FF 00 73 74 72 69 6E 67 00 00 44
61 74 61 32 00 08 00 00 00 02 00 08 00 00 00 01
00 00 00 11 00 00 00 0A 00 00 80 03 08 00 00 00
BF 00 00 00 00 73 74 72 69 6E 67 00 00 49 64 00
03 40 00 00 00 00 00 00 00 00 00 00 00 00 1C 00
00 00 0A 00 00 80 23 08 00 00 00 F5 00 00 00 01
00 00 80 33 0B 00 00 00 FF FF 00 73 69 6E 74 33
32 00 00 64 65 66 61 75 6C 74 56 61 6C 75 65 00
00 00 00 00 00 00 0C 00 00 00 00 00 00 73 00 00
00 80
80 01 00 00 00 00 00 00 00 11 00 00 00 1b 00 00
00 00 4d 79 43 6c 61 73 73 00 09 00 00 00 00 42
61 73 65 00 06 00 00 00 04 00 00 00 04 00 00 00
0a 00 00 00 11 00 00 00 38 00 00 00 3f 00 00 00
66 00 00 00 6d 00 00 00 94 00 00 00 98 00 00 00
ef ff ff ff ff ff ff ff ff ca 00 00 00 ff ff ff
ff 1b 01 00 80 00 4d 79 43 6c 61 73 73 32 00 00
41 72 72 61 79 00 13 60 00 00 03 00 0c 00 00 00
01 00 00 00 11 00 00 00 0a 00 00 80 23 08 00 00
00 30 00 00 00 00 75 69 6e 74 33 32 00 00 44 61
74 61 31 00 08 40 00 00 01 00 04 00 00 00 01 00
00 00 11 00 00 00 0a 00 00 80 23 08 00 00 00 5e
00 00 00 00 73 74 72 69 6e 67 00 00 44 61 74 61
32 00 08 40 00 00 02 00 08 00 00 00 01 00 00 00
11 00 00 00 0a 00 00 80 23 08 00 00 00 8c 00 00
00 00 73 74 72 69 6e 67 00 00 49 64 00 03 40 00
00 00 00 00 00 00 00 00 00 00 00 1c 00 00 00 0a
00 00 80 23 08 00 00 00 c2 00 00 00 01 00 00 80
33 0b 00 00 00 ff ff 00 73 69 6e 74 33 32 00 00
64 65 66 61 75 6c 74 56 61 6c 75 65 00 00 00 00
```

```
00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
6b 05 00 00 01 00 00 00
00 00 00 00 00 00 00 00
02 00 00 00 f7 04 00 00
09 00 00 00 09 02 00 00
47 05 00 80 00 52 65 73 74 61 72 74 00
fc 01 00 00
05 00 44 50 2d 4d 00 00 52 4f 4f 54 5c 64 65
66 61 75 6c 74 00 1d 00 00 00 00
ff ff ff ff
00 00 00 00 04 00 00 00 04 00 00 00
00 00 00 00 00 00 00 80 0c 00 00 00 00 00 00 00
00 00 00 80
b2 01 00 0000 00 00 00 00
05 00 00 00 04 00 00 00
0F 00 00 000e 00 00 0000
0b 00 00 00
ff ff 01 00 00 00
2a 00 00 00 90 00 00 00 19 ff ff ff ff
cf 00 00 80
      00 5f 5f 50 41 52 41 4d 45 54 45 52 53 00
00 61 62 73 74 72 61 63 74 00
08 00 00 00 00 00 00 00 00 00 00 00 00 00
      04 00 00 00 00 53 65 72 76 69 63 65 4e 61 6d 65 00
00 73 74 72 69 6e 67 00
08 00 00 00 00 00 00 00 00 00 00 00 00 00
      11 00 00 00 0a 00 00 80 03
      08 00 00 00 37 00 00 00
00 69 6e 00 08 00 00 00 00 00 00 00 00 00
      00 00 00 00 1c 00 00 00 0a 00 00 80 03 08 00 00
00 37 00 00 00 5e 00 00 00 00 0b 00 00 00
ff ff 00 49 44 00 08 00 00 00 00 00
00 00 00 00 00 00 00 00 29 00 00 00
0a 00 00 80 03 08 00 00 00 c7 00 00 00
5e 00 00 00 00 0b 00 00 00 ff ff
8c 00 00 00 11 03 00 00 00 00 00 00 00
00 73 74 72 69 6e 67 00
00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
00 00 00 00 00 00 00 00 00 00 00 00 00 00
0c 00 00 00 00 00 5f 5f 00 00 00 80
ea 02 00 00
05 00 44 50 2d 4d 00
00 52 4f 4f 54 5c 64 65 66 61 75 6c 74 00 1d
00 00 00 00 ff ff ff ff 00 00 00 00
04 00 00 00 04 00 00 00 00 00 00 00 00
00 00 80 0c 00 00 00 00 00 00 00 00 00 80 a0
```

```
02 00 00 00 00 00 00 00 09 00 00 00 04 00 00 00
0f 00 00 00 0e 00 00 00 00 0b 00 00 00 ff ff 02
00 00 00 e1 00 00 00 22 01 00 00 2a 00 00 00 8c
00 00 00 15 ff ff ff ff ff ff ff ff 4c 01 00 80
00 5f 5f 50 41 52 41 4d 45 54 45 52 53 00 00 61
62 73 74 72 61 63 74 00 0d 00 00 00 00 00 00 00
00 00 00 00 00 00 04 00 00 00 00 53 74 61 74 75
73 00 00 6f 62 6a 65 63 74 00 0d 00 00 00 00 00
00 00 00 00 00 00 00 00 11 00 00 00 0a 00 00 80
03 08 00 00 00 32 00 00 00 00 6f 75 74 00 0d 00
00 00 00 00 00 00 00 00 00 00 00 00 1c 00 00 00
0a 00 00 80 03 08 00 00 00 32 00 00 00 59 00 00
00 00 0b 00 00 00 ff ff 00 49 44 00 0d 00 00 00
00 00 00 00 00 00 00 00 00 00 29 00 00 00 0a 00
00 80 03 08 00 00 00 c3 00 00 00 59 00 00 00 00
0b 00 00 00 ff ff 88 00 00 00 11 03 00 00 00 01
00 00 00 00 6f 62 6a 65 63 74 3a 69 6e 74 00 13
00 00 00 01 00 04 00 00 00 00 00 00 00 04 00 00
00 00 52 65 74 75 72 6e 56 61 6c 75 65 00 00 75
69 6e 74 33 32 00 13 00 00 00 01 00 04 00 00 00
00 00 00 00 11 00 00 00 0a 00 00 80 03 08 00 00
00 15 01 00 00 00 75 69 6e 74 33 32 00 00 6f 75
74 00 13 00 00 00 01 00 04 00 00 00 00 00 00 00
1c 00 00 00 0a 00 00 80 03 08 00 00 00 15 01 00
00 1d 01 00 00 00 0b 00 00 00 ff ff 00 00 00 00
00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
0c 00 00 00 00 00 5f 5f 00 00 00 80
1c 00 00 00 13 05 00 00 00 0b 00 00 00 ff ff
1c 05 00 00 00 08 20 00 00 29 05 00 00
00 65 78 65 63 75 74 65 00
00 70 65 72 66 6f 72 6d 61 6e 63 65 00
02 00 00 00 35 05 00 00 3b 05 00 00
00 66 61 73 74 00 00 73 69 64 65 66 66 65 63 74 73 00
20 00 00 29 05 00 00 00 65 78 65 63 75 74 65 00 00
70 65 72 66 6f 72 6d 61 6e 63 65 00 02 00 00 00 35
05 00 00 3b 05 00 00 00 66 61 73 74 00 00 73 69 64
65 66 66 65 63 74 73 00 00 00 00 00
```

Relevant offset	Octet values	Comments
		EncodingUnit.
	78 56 32 12	Standard CIM object signature.
	D3 01 00 00	UINT32 length of entire CIM class encoding (0x1D3 octets).
		ObjectBlock.
	Shaded Bold Portion 76 01 00 00... ...73 00 00 00 80	MyClass encoding.
	80 01 00 00	EncodingLength of MyClass2, or 0x180 octets.
	00	Reserved.
	00 00 00 00	Offset of CIM class name in CIM class Heap.
	11 00 00 00	ValueTableLength.
	1B 00 00 00	DerivationList length in octets.
	00 4D 79 43 6C 61 73 73 00	Encoded-String MyClass, which is the superclass of the current class.
	09 00 00 00	Length in octets of previous Encoded-String.
	00 42 61 73 65 00	Encoded-String Base, which is the superclass of MyClass.
	06 00 00 00	Length of octets in previous Encoded-String.
	04 00 00 00	ClassQualifierSet::EncodingLength, including itself. There are no qualifiers, so the octet count is four, which is just enough to encode its own length.

Relevant offset	Octet values	Comments
	04 00 00 00	PropertyLookupTable::PropertyCount.
	0A 00 00 00	PropertyNameRef, CIM property 1.
	11 00 00 00	PropertyInfoRef.
	38 00 00 00	PropertyNameRef, CIM property 2.
	3F 00 00 00	PropertyInfoRef.
	66 00 00 00	PropertyNameRef, CIM property 3.
	6d 00 00 00	PropertyInfoRef.
	94 00 00 00	PropertyNameRef, CIM property 4.
	98 00 00 00	PropertyInfoRef, CIM property 4.
	EF	NdTable.
	FF FF FF FF	Value Table, CIM property 1.
	FF FF FF FF	Value Table, CIM property 2.
	CA 00 00 00	Value Table, CIM property 3.
	FF FF FF FF	Value Table, CIM property 4.
	1B 01 00 80	ClassHeap::HeapLength (0x11B octets, 283 decimal).
	283 octets in shaded italics 00 4d 79... ...00 00 00	ClassHeap. This is essentially the same encoding as for MyClass because MyClass2 only adds a method and no properties.

Relevant offset	Octet values	Comments
	6b 05 00 00	MethodsPart::EncodingLength. (0x56B octets).
	01 00 00 00	MethodCount (only one method).
	00 00 00 00	MethodDescription::MethodName. Reference to Encoded-String in Method Heap; this points to Restart.
	00	MethodFlags (reserved octet; must be 0).
	00 00 00	MethodPadding.
	02 00 00 00	MethodOrigin (Class[2] in DerivationList) or MyClass2.
	F7 04 00 00	MethodQualifiers (HeapRef to QualifierSet).
	09 00 00 00	InputSignature (HeapRef to MethodSignature) for input parameters.
	09 02 00 00	OutputSignature (HeapRef to MethodSignature) for output parameters and return value.
	47 05 00 80	MethodHeap::HeapLength, 0x547 octets, with MS bit set.
0000	00 52 65 73 74 61 72 74 00	Encoded-String Restart.
0009	FC 01 00 00	MethodSignatureBlock::EncodingLength. The ObjectBlock is an embedded ClassPart that represents CIM class __PARAMETERS for the InputSignature. This block is shown indented in the hex dump above.
	05	ObjectFlags (Class + Decoration).
	00 44 50 2d 4d 00	DecServerName: Encoded-String of DP-M.
	00 52 4F 4F 54 5C 64 65 66 61 75 6C 74 00	DecNamespaceName: Encoded-String ROOT\default.

Relevant offset	Octet values	Comments
		Here begins the CIM class encoding for __PARAMETERS. There are ParentClass and CurrentClass parts, in succession. The ParentClass is more or less empty.
	1d 00 00 00	ClassHeader::EncodingLength. (ParentClass part of ClassType.)
	00	ReservedOctet.
	ff ff ff ff	ClassNameRef. This is a dummy heap reference, indicating no CIM class name. This is because the parent CIM class of __PARAMETERS is currently being encoded, and does not exist.
	00 00 00 00	ValueTableLength.
	04 00 00 00	DerivationList length in octets, including itself (no Derivation).
	04 00 00 00	ClassQualifierSet::EncodingLength, including itself. (No CIM qualifier set.)
	00 00 00 00	PropertyCount.
	00 00 00 80	HeapLength (zero, MS bit set).
	0C 00 00 00	MethodsPart::EncodingLength.
	00 00	MethodsPart::MethodCount.
	00 00	MethodsPart::MethodCountPadding.
	00 00 00 80	MethodsPart::MethodHeap, zero-length heap.
	b2 01 00 00	ClassPart::EncodingLength (0x1b2, 434 decimal octets). This is the CurrentClass part of the ClassType.
	00	ReservedOctet.
	00 00 00 00	ClassNameRef (points to __PARAMETERS).
	05 00 00 00	ValueTableLength.

Relevant offset	Octet values	Comments
	04 00 00 00	DerivationList (only itself, so no derivation).
	0F 00 00 00	ClassQualifierSet::EncodingLength.
	0E 00 00 00	QualifierName (reference), points to abstract, a requirement for all classes of type __PARAMETERS.
	00	QualifierFlavor.
	0B 00 00 00	QualifierType (CIM-TYPE-BOOLEAN).
	FF FF	TRUE.
	01 00 00 00	PropertyLookupTable::PropertyCount (one in parameter acting as a property).
	2a 00 00 00	PropertyNameRef (points to ServiceName).
	90 00 00 00	PropertyInfoRef.
	19	NdTable.
	FF FF FF FF	ValueTable, no default value for the parameter ServiceName.
Heap Offsets in this column	CF 00 00 80	ClassHeap::HeapLength (0xCF or 207 decimal octets).
00	00 5f 5f 50 41 52 41 4d 45 54 45 52 53 00	Encoded-String __PARAMETERS.
0E	00 61 62 73 74 72 61 63 74 00	Encoded-String abstract.
18	08 00 00 00 00 00 00 00 00 00 00 00 00	Unused; this is part of heap fragmentation and can be removed if the heap offsets for all subsequent HeapItems adjusted.

Relevant offset	Octet values	Comments
	00 04 00 00 00	
2A	00 53 65 72 76 69 63 65 4e 61 6d 65 00	Encoded-String ServiceName.
37	00 73 74 72 69 6e 67 00	Encoded-String string.
3F	08 00 00 00 00 00 00 00 00 00 00 00 00 00 11 00 00 00	PropertyInfo: PropertyType = CIM-TYPE-STRING. DeclarationOrder (0). ValueTableOffset (0). PropertyQualifierSet length 0x11 octets.
51	0a 00 00 80	QualifierName. DictionaryReference CIMTYPE.
55	03	QualifierFlavor.
	08 00 00 00	QualifierType (CIM-TYPE-STRING).
5A	37 00 00 00	QualifierValue (HeapRef to offset 0x37, or string).
5E	00 69 6E 00	Encoded-String in QualifierName (referenced later in the heap).
62	08 00 00 00 00 00 00 00 00 00 00 00 00 00 1C 00 00 00 0a 00 00 80 03 08 00 00 00 37 00 00 00	Unreferenced Heap fragment lost from previous editing or updates.

Relevant offset	Octet values	Comments
	5e 00 00 00 00 0b 00 00 00 ff ff	
8C	00 49 44 00	Encoded-String ID.
90	08 00 00 00 00 00 00 00 00 00 00 00 00 00 29 00 00 00	PropertyInfo: PropertyType = CIM-TYPE-STRING. DeclarationOrder (0). ValueTableOffset (0). PropertyQualifierSet length 0x29 octets.
	0a 00 00 80	QualifierName (DictionaryReference to CIMTYPE).
	03	QualifierFlavor.
	08 00 00 00 C7 00 00 00	QualifierType (CIM-TYPE-STRING).
10F	5E 00 00 00	QualifierValue (Offset 0xC7).
	00	QualifierName (Offset 5E points to in).
	00	Flavor.
	0B 00 00 00	QualifierType (CIM-TYPE-BOOL).
	FF FF	Logical TRUE.
	8C 00 00 00	QualifierName (points to ID, an attribute added to all properties acting as parameters in a method).
	11	QualifierFlavor.
11F	03 00 00 00	QualifierType (CIM-TYPE-SINT32).

Relevant offset	Octet values	Comments
	00 00 00 00	QualifierValue (zero) (meaning this is the 0th parameter in the signature).
127	00 73 74 72 69 6e 67 00	Encoded-String string.
	00..00	Unreferenced heap fragment lost from previous editing or updates. Heap Space: 174 octets of zero octets.
	0c 00 00 00 00 00 5f 5f 00 00 00 80	MethodsPart with total EncodingLength 0xC octets. MethodCount is zero, two octets of MethodCountPadding with random values. The MethodHeap is zero length with MS bit set.
	ea 02 00 00 … ...5f 5f 00 00 00 80	MethodSignatureBlock: :EncodingLength. (0x2EA, 746 octets). This is the Encoding block for __PARAMETERS for the output parameters and return value. It is decoded as in the previous example for __PARAMETERS for the input parameters.
	1C 00 00 00	QualifierSet::EncodingLength.
	13 05 00 00	QualifierName.
	00	QualifierFlavor.
	0B 00 00 00	QualifierType (CIM-TYPE-BOOL).
	FF FF	QualifierValue (TRUE).
	1C 05 00 00	Qualifier::Name.
	00	Flavor.
	08 20 00 00	Type (CIM-TYPE-STRING and CimArrayFlag), an array of strings.
	29 05 00 00	Location of string array in Heap.

Relevant offset	Octet values	Comments
	00 65 78 65 63 75 74 65 00	Encoded-String execute.
	00 70 65 72 66 6f 72 6d 61 6e 63 65 00	Encoded-String performance.
	02 00 00 00	Encoded-Array (two items).
	35 05 00 00	Location of first string in array (fast).
	3B 05 00 00	Location of second string in array (side effects).
	00 66 61 73 74 00	Encoded-String fast.
	00 73 69 64 65 66 66 65 63 74 73 00	Encoded-String side effects.
	20 00 00 29 05... ...74 73 00 00 00 00 00	Remaining octets in italics. Remainder of packet consists of no information and fills out the entire encoding length. These octets can be removed if the encoding length at the beginning of the encoding is adjusted.

5 Security

Because this specification only specifies an encoding, there are no security-specific considerations. There are no fields within the encoding associated with security.

6 Appendix A: Windows Behavior

The information in this specification is applicable to the following versions of Windows:

- Windows Server 2003 SP2
- Windows Server 2003
- Windows Vista
- Windows XP
- Windows XP SP1
- Windows XP 64-Bit Edition
- Windows 2000

Exceptions, if any, are noted below. Unless otherwise specified, any statement of optional behavior in this specification prescribed using the terms SHOULD or SHOULD NOT implies Windows behavior in accordance with the SHOULD or SHOULD NOT prescription. Unless otherwise specified, the term MAY implies that Windows does not follow the prescription.

<1> Section 2.2.59: Windows ignores other bit values.

7 Appendix B: ABNF Encoding Definition

```
;
;--- Main block
;
EncodingUnit = Signature ObjectEncodingLength ObjectBlock
ObjectEncodingLength = UINT32
ObjectBlock = ObjectFlags [Decoration] Encoding
ObjectFlags = OCTET
; Bit 0 = Class, Bit 1 = Instance,
; Bit 2 = DecorationBlock is present
Decoration = DecServerName DecNamespaceName
DecServerName = Encoded-String
DecNamespaceName = Encoded-String
Encoding = InstanceType / ClassType
;
;------ CIM class Encoding ----
;
ClassType = ParentClass CurrentClass
ParentClass = ClassAndMethodsPart
CurrentClass = ClassAndMethodsPart
ClassAndMethodsPart = ClassPart [MethodsPart]
 ; [MethodsPart] is always present if ObjectFlags indicates
 ; the CIM object is a CIM class definition, and always absent
 ; if the current CIM object is a CIM instance definition
ClassPart = ClassHeader DerivationList ClassQualifierSet
      PropertyLookupTable [NdTable ValueTable] ClassHeap
ClassHeader = EncodingLength ReservedOctet
      ClassNameRef ValueTableLength
DerivationList = EncodingLength *ClassNameEncoding
ClassNameEncoding = Encoded-String EncodingLength
ClassNameRef = HeapStringRef
ClassQualifierSet = QualifierSet
; --------------------
PropertyLookupTable = PropertyCount *PropertyLookup
 ; PropertyLookup entries are sorted by
 ;   CIM Property name so that binary
 ;   searches are possible.
PropertyCount = UINT32
PropertyLookup = PropertyNameRef PropertyInfoRef
PropertyNameRef = HeapStringRef
PropertyInfoRef = HeapRef
;-------------------
NdTable = *NullAndDefaultFlag
NullAndDefaultFlag = 2*BIT
  ;nullness = bit 0, inheritedDefault = bit1
ValueTableLength = UINT32
ValueTable = *EncodedValue
PropertyInfo = PropertyType DeclarationOrder
    ValueTableOffset ClassOfOrigin PropertyQualifierSet
PropertyType = CimType
DeclarationOrder = UINT16
ValueTableOffset = UINT32
ClassOfOrigin = UINT32
; Which CIM class in the DerivationList this
;   CIM Property comes from %x0 == the current class.
PropertyQualifierSet = QualifierSet
ClassHeap = Heap
```

```
;--- Method Encoding ---
MethodsPart = EncodingLength MethodCount
    MethodCountPadding *MethodDescription MethodHeap
MethodCount = UINT16
MethodCountPadding = 2OCTET
MethodHeap = Heap
MethodDescription = MethodName MethodFlags MethodPadding
    MethodOrigin MethodQualifiers InputSignature OutputSignature
MethodName = HeapStringRef
MethodFlags = OCTET
MethodPadding = 3OCTET
MethodOrigin = UINT32         ; CIM class in DerivationList
MethodQualifiers = HeapQualifierSetRef
InputSignature = MethodSignature
OutputSignature = MethodSignature
MethodSignature = HeapMethodSignatureBlockRef
; --- CIM instance encoding
InstanceType = CurrentClass EncodingLength InstanceFlags
   InstanceClassName NdTable InstanceData InstanceQualifierSet
   InstanceHeap
InstanceFlags = OCTET
InstanceClassName = HeapStringRef
InstanceData = ValueTable
InstanceQualifierSet = QualifierSet InstancePropQualifierSet
InstanceHeap = Heap
;--- CIM Qualifier Sets ---
QualifierSet = EncodingLength *Qualifier
Qualifier = QualifierName QualifierFlavor
   QualifierType QualifierValue
QualifierName = HeapStringRef
QualifierFlavor = OCTET
QualifierType = CimType
QualifierValue = EncodedValue
InstancePropQualifierSet = InstPropQualSetFlag *QualifierSet
InstPropQualSetFlag = %x1 / %x2
   ; One OCTET.  If 1, there is no list of Qualifiers in
   ; InstanceQualifierSet.  If 2, there is a list of Qualifiers.
   ; The number of qualifiers is equivalent to the number of
   ; properties in the CIM class definition for the instance.
   ; The CIM Qualifier sets are in the lexical order for the
   ; properties, as in the PropertyLookupTable.
;--- Heap ---
Heap = HeapLength *HeapItem
HeapStringRef = HeapRef
HeapQualifierSetRef = HeapRef
ClassPartRef = HeapRef
HeapMethodSignatureBlockRef = HeapRef
HeapRef = UINT32 / DictionaryReference
   ; The DictionaryReference choice is taken if the
   ; reference value has the MS bit set.  Therefore
   ; only 31 bits are used for an offset into the heap
   ; when the DictionaryReference is not being used.

HeapItem = PropertyInfo / Encoded-String / Encoded-Array /
   QualifierSet / ObjectBlock / MethodSignatureBlock
MethodSignatureBlock = EncodingLength [ObjectBlock]
EncodedValue = NumericValue / HeapRef / BOOL / NoValue
   ; Note that values are inline if they are
```

```
   ;  NumericValue, BOOL, or NoValue
   ; If the CimType of the encoded value is CIM-TYPE-STRING
   ;  or an array of any kind,
   ; then HeapRef points to the value in the heap.
;--- Simple types ----
HeapLength = UINT32
    ;*MS bit is always set, so length is expressed in lower 31 bits
EncodingLength = UINT32
NoValue = %xFF %xFF %xFF %xFF     ; 32 bits all set to 1
Encoded-String = Encoded-String-Flag *Character Null;
Encoded-String-Flag = OCTET
Character = AnsiCharacter / UnicodeCharacter
Null = Character
AnsiCharacter = OCTET
UnicodeCharacter = 2*OCTET
Encoded-Array = ArrayCount *EncodedValue
ArrayCount = UINT32
; The DictionaryReference is used where HeapRef may appear and the
;   EncodedValue type is Encoded-String
; These appear as 32 bit offsets into the Heap with the
;   MS bit set to 1 and the lower
; 31 bits set to one of the integer values below
DictionaryReference = %d0 / %d1 / %d2 / %d3 / %d4 / %d5 /
      %d6 / %d7 / %d8 / %d9 / %d10
  ; %d0   Encoded/Decoded as quote character
  ; %d1   Encoded/Decoded as "key"
  ; %d2   Encoded/Decoded as ""
  ; %d3   Encoded/Decoded as "read"
  ; %d4   Encoded/Decoded as "write"
  ; %d5   Encoded/Decoded as "volatile"
  ; %d6   Encoded/Decoded as "provider"
  ; %d7   Encoded/Decoded as "dynamic"
  ; %d8   Encoded/Decoded as "cimwin32"
  ; %d9   Encoded/Decoded as "DWORD"
  ; %d10  Encoded/Decoded as "CIMTYPE"
ReservedOctet = OCTET  ;*doc
Signature = UINT32  ;0x12345678 little-endian ;*doc
NumericValue = BYTE / SINT16 / UINT16 / SINT32 / UINT32 /
   SINT64 / UINT64 / REAL32 / REAL64 ;*doc
BYTE = OCTET
UINT32 = 4*OCTET
SINT32 = 4*OCTET
UINT64 = 8*OCTET
SINT64 = 8*OCTET
REAL32 = 4*OCTET             ; IEEE short floating-point format
REAL64 = 8*OCTET             ; IEEE format
UINT16 = 2*OCTET
SINT16 = 2*OCTET
BOOL = 2*OCTET                  ;*
OCTET = %x0-FF                  ;*
BIT = %x0 / %x1                 ;*doc
CimType = CimBaseType / CimArrayType
  ; 32 bit encoding, upper 16 bits not used.
CimArrayFlag = %x20 %x00           ; 0x2000 bit flag
CimBaseType = CIM-TYPE-EMPTY / CIM-TYPE-ILLEGAL /
   CIM-TYPE-SINT8 / CIM-TYPE-UINT8 /
   CIM-TYPE-SINT16 / CIM-TYPE-UINT16 /
   CIM-TYPE-SINT32 / CIM-TYPE-UINT32 /
```

```
    CIM-TYPE-SINT64 / CIM-TYPE-UINT64 /
    CIM-TYPE-REAL32 / CIM-TYPE-REAL64 /
    CIM-TYPE-BOOLEAN / CIM-TYPE-STRING /
    CIM-TYPE-DATETIME / CIM-TYPE-REFERENCE /
    CIM-TYPE-CHAR16 / CIM-TYPE-OBJECT

CimArrayType = CIM-ARRAY-SINT8 / CIM-ARRAY-UINT8 /
    CIM-ARRAY-SINT16 / CIM-ARRAY-UINT16 /
    CIM-ARRAY-SINT32 / CIM-ARRAY-UINT32 /
    CIM-ARRAY-SINT64 / CIM-ARRAY-UINT64 /
    CIM-ARRAY-REAL32 / CIM-ARRAY-REAL64 /
    CIM-ARRAY-BOOLEAN / CIM-ARRAY-STRING /
    CIM-ARRAY-DATETIME / CIM-ARRAY-REFERENCE /
    CIM-ARRAY-CHAR16 / CIM-ARRAY-OBJECT

CIM-TYPE-EMPTY = %d0
CIM-TYPE-ILLEGAL = %d4095
CIM-TYPE-SINT8 = %d16
CIM-TYPE-UINT8 = %d17
CIM-TYPE-SINT16 =%d2
CIM-TYPE-UINT16 =%d18
CIM-TYPE-SINT32 = %d3
CIM-TYPE-UINT32 = %d19
CIM-TYPE-SINT64 = %d20
CIM-TYPE-UINT64 = %d21
CIM-TYPE-REAL32 = %d4
CIM-TYPE-REAL64 = %d5
CIM-TYPE-BOOLEAN = %d11
CIM-TYPE-STRING = %d8
CIM-TYPE-DATETIME = %d101
CIM-TYPE-REFERENCE = %d102
CIM-TYPE-CHAR16 = %d103
CIM-TYPE-OBJECT = %d13

CIM-ARRAY-SINT8 = %d8208
CIM-ARRAY-UINT8 = %d8209
CIM-ARRAY-SINT16 =%d8194
CIM-ARRAY-UINT16 =%d8210
CIM-ARRAY-SINT32 = %d8195
CIM-ARRAY-UINT32 = %d8201
CIM-ARRAY-SINT64 = %d8202
CIM-ARRAY-UINT64 = %d8203
CIM-ARRAY-REAL32 = %d8196
CIM-ARRAY-REAL64 = %d8197
CIM-ARRAY-BOOLEAN = %d8203
CIM-ARRAY-STRING = %d8200
CIM-ARRAY-DATETIME = %d8293
CIM-ARRAY-REFERENCE = %d8294
CIM-ARRAY-CHAR16 = %d8295
CIM-ARRAY-OBJECT = %d8205
```

8 Index

O

P

Q

R

S

V

W