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

---

Intellectual Property Rights Notice for Open Specifications Documentation

- **Technical Documentation.** Microsoft publishes Open Specifications documentation for protocols, file formats, languages, standards as well as overviews of the interaction among each of these technologies.
- **Copyrights.** This documentation is covered by Microsoft copyrights. Regardless of any other terms that are contained in the terms of use for the Microsoft website that hosts this documentation, you may make copies of it in order to develop implementations of the technologies described in the Open Specifications and may distribute portions of it in your implementations using these technologies or your documentation as necessary to properly document the implementation. You may also distribute in your implementation, with or without modification, any schema, IDL's, or code samples that are included in the documentation. This permission also applies to any documents that are referenced in the Open Specifications.
- **No Trade Secrets.** Microsoft does not claim any trade secret rights in this documentation.
- **Patents.** Microsoft has patents that may cover your implementations of the technologies described in the Open Specifications. Neither this notice nor Microsoft's delivery of the documentation grants any licenses under those or any other Microsoft patents. However, a given Open Specification may be covered by Microsoft's Open Specification Promise (available here: http://www.microsoft.com/interop/osp) or the Community Promise (available here: http://www.microsoft.com/interop/cp/default.mspx). If you would prefer a written license, or if the technologies described in the Open Specifications are not covered by the Open Specifications Promise or Community Promise, as applicable, patent licenses are available by contacting iplg@microsoft.com.
- **Trademarks.** The names of companies and products contained in this documentation may be covered by trademarks or similar intellectual property rights. This notice does not grant any licenses under those rights.
- **Fictitious Names.** The example companies, organizations, products, domain names, e-mail addresses, logos, people, places, and events depicted in this documentation are fictitious. No association with any real company, organization, product, domain name, email address, logo, person, place, or event is intended or should be inferred.

Reservation of Rights. All other rights are reserved, and this notice does not grant any rights other than specifically described above, whether by implication, estoppel, or otherwise.

Tools. The Open Specifications do not require the use of Microsoft programming tools or programming environments in order for you to develop an implementation. If you have access to Microsoft programming tools and environments you are free to take advantage of them. Certain Open Specifications are intended for use in conjunction with publicly available standard specifications and network programming art, and assumes that the reader either is familiar with the aforementioned material or has immediate access to it.

Revision Summary

Date	Revision History	Revision Class	Comments
07/20/2007	0.1	Major	MCPP Milestone 5 Initial Availability
09/28/2007	0.2	Minor	Updated the technical content.
10/23/2007	0.3	Minor	Clarification of ABNF item.
11/30/2007	0.4	Minor	Updated the technical content.
01/25/2008	0.5	Editorial	Revised and edited the technical content.
03/14/2008	0.5.1	Editorial	Revised and edited the technical content.
05/16/2008	0.5.2	Editorial	Revised and edited the technical content.
06/20/2008	1.0	Major	Updated and revised the technical content.
07/25/2008	1.1	Minor	Updated the technical content.
08/29/2008	1.1.1	Editorial	Revised and edited the technical content.
10/24/2008	1.2	Minor	Updated the technical content.
12/05/2008	1.3	Minor	Updated the technical content.
01/16/2009	1.3.1	Editorial	Revised and edited the technical content.
02/27/2009	1.3.2	Editorial	Revised and edited the technical content.
04/10/2009	1.3.3	Editorial	Revised and edited the technical content.
05/22/2009	2.0	Major	Updated and revised the technical content.
07/02/2009	3.0	Major	Updated and revised the technical content.
08/14/2009	4.0	Major	Updated and revised the technical content.
09/25/2009	5.0	Major	Updated and revised the technical content.
11/06/2009	5.1	Minor	Updated the technical content.
12/18/2009	6.0	Major	Updated and revised the technical content.
01/29/2010	6.0.1	Editorial	Revised and edited the technical content.
03/12/2010	7.0	Major	Updated and revised the technical content.
04/23/2010	7.0.1	Editorial	Revised and edited the technical content.
06/04/2010	7.0.2	Editorial	Revised and edited the technical content.
07/16/2010	7.0.2	No change	No changes to the meaning, language, or formatting of the technical content.

Date	Revision History	Revision Class	Comments
08/27/2010	7.0.2	No change	No changes to the meaning, language, or formatting of the technical content.
10/08/2010	7.0.2	No change	No changes to the meaning, language, or formatting of the technical content.
11/19/2010	7.0.2	No change	No changes to the meaning, language, or formatting of the technical content.
01/07/2011	7.0.2	No change	No changes to the meaning, language, or formatting of the technical content.
02/11/2011	8.0	Major	Significantly changed the technical content.

Contents

1 Introduction

The Windows Management Instrumentation Encoding Version 1.0 Protocol specifies a binary data **encoding** format that is used by the Windows Management Instrumentation Remote Protocol for network communication.

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

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. This specification depends entirely on the metamodel and terminology specified in the DMTF specification set. The reader is referred to the CIM Infrastructure Specification for a description of the CIM metamodel. The **CIM objects** that are transferred by the Windows Management Instrumentation Remote Protocol are CIM objects encoded by using the technique specified in this specification.

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

1.1 Glossary

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

Augmented Backus-Naur Form (ABNF)
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
Distributed Component Object Model (DCOM)
Domain Name System (DNS)
encoding
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://ieeexplore.ieee.org/servlet/opac?punumber=2355

[MS-DCOM] Microsoft Corporation, "Distributed Component Object Model (DCOM) Remote Protocol Specification", March 2007.

[MS-WMI] Microsoft Corporation, "Windows Management Instrumentation Remote Protocol Specification", September 2007.

[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.

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

1.3 Overview

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 that is 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 WMI Remote Protocol is a management protocol for querying status and controlling the settings of real-world managed entities. These entities are modeled by 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 in the **CIM class**. CIM class definitions are thus similar to class definitions in other object-oriented database systems and programming systems.

In the WMI Remote Protocol, each managed entity is assigned 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 drive C, one for drive D, and one for drive E.

To query the status of the real-world CIM objects, the WMI Remote Protocol is used to retrieve these instances by using operations such as GetObject or ExecQuery. If updates are required, the WMI Remote Protocol is used to send the updated CIM instance over the wire with the new values. To perform an update, the WMI Remote Protocol writes the complete updated instance, even if only one value is changed. Thus, the WMI Remote Protocol 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 by using the MOF syntax. This is a text-only format for use by tools and in documentation and is not designed for use in 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 WMI Remote Protocol 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 PutInstance operation (as specified in [MS-WMI]) is used, and the updated CIM instance is encoded by using the format in this specification.

The WMI Remote Protocol 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 WMI Remote Protocol.

When retrieving CIM objects, the binary encoding that is transmitted over the WMI Remote Protocol must be decoded. The binary packet is parsed by using **Augmented Backus-Naur Form (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, ABNF shows that the first nonterminal 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. After 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 cannot detect 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 determine whether the CIM object being encoded is a CIM class or a CIM instance. Then the encoder emits the correct octet value using the ObjectFlags rule. Before this octet is emitted, the encoder must determine whether

the encoded CIM object will contain the server name of origin (the DecServerName rule) and the CIM namespace name (DecNamespaceName). After it is known whether 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 that use the ABNF grammar (as specified in [RFC4234]) should be thoroughly familiar with recursive-descent parsing, concepts of terminal and nonterminal productions, LL(1) grammar theory, and code emission techniques with syntax-directed translators. The techniques for the encoding and decoding are thus equivalent to the techniques used by high-level language compilers.

1.4 Relationship to Protocols and Other Structures

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 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].

1.6 Versioning and Localization

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

1.7 Vendor-Extensible Fields

The encoding format is not extensible.

2 Structures

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

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

2.1 Introduction

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

2.2 Annotated Object Block Encoding

CIM instance and CIM class definitions, as specified in [DMTF-DSP004] section 2.1, are encoded by using a binary data format. Qualifiers for instance and instance properties are Microsoft extensions to the CIM data model. Like qualifiers applied to a class, qualifiers applied to an instance are specific to the instance, and the qualifier need not be specified in the class from which the instance is derived. If the class defines the qualifier, then the instance can redefine the qualifier only if the class allows overriding the qualifier. 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 ABNF, as specified in section 6.

Because CIM classes and CIM instances have user-name properties and values, the data block can vary significantly, depending on the item that is 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 characters, such as UINT32, and have a binary encoding rule, as specified in section 2.3. All other productions of grammar are defined within ABNF. All the integer, unsigned integer, and floating point numbers that are encountered in the following encoding structures MUST be stored in little-endian format, unless explicitly stated otherwise.

In the definitions in this section, the use of the term offset always refers to an unsigned integer value that represents 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 object block is the root node block of the encoding that is used for encoding classes or instances if the object is encoded as specified in [MS-WMI] section 2.2.4.1. This block is contained within transmission encoding, as specified in [MS-DCOM] and [MS-WMI] respectively.

```
EncodingUnit = Signature ObjectEncodingLength ObjectBlock
```

The **Signature** field (as specified in section 2.2.77) acts as a verification signature for the EncodingUnit that is used to encode the CIM object that follows, according to the algorithm that is specified in this specification. Any other value MUST constitute an error.

ObjectEncodingLength (section 2.2.4) represents the size of the ObjectBlock (section 2.2.5) that contains the encoded CIM class or CIM instance.

2.2.2 EncodingUnitObjectBlock

EncodingUnitObjectBlock is the root node of the encoding that is used for encoding classes or instances if the object is encoded as specified in [MS-WMI] section 2.2.14.2 and section 2.2.14.3. During transmission, this block is contained within the ObjectArray structure, as specified in [MS-WMI] section 2.2.14.

```
EncodingUnitObjectBlock = ObjectBlock
```

ObjectBlock contains the binary encoding of the CIM object. The length of ObjectBlock MUST match the length specified in **dwSizeOfData** of WBEMOBJECT_CLASS (as specified in [MS-WMI] section 2.2.14.2) if EncodingUnitObjectBlock is contained within WBEMOBJECT_CLASS; or the length specified in **dwSizeOfData** of WBEMOBJECT_INSTANCE (as specified in [MS-WMI] section 2.2.14.3) if EncodingUnitObjectBlock is contained within WBEMOBJECT_INSTANCE.

2.2.3 EncodingUnitInstanceNoClass

The EncodingUnitInstanceNoClass is the root node of the encoding that is used for encoding instances if the CIM instance is encoded as specified in [MS-WMI] section 2.2.14.4. During transmission, this block is contained within the ObjectArray structure, as specified in [MS-WMI] section 2.2.14.

```
EncodingUnitInstanceNoClass = ObjectFlags [Decoration]
      EncodingLength InstanceFlags InstanceClassName NdTable
      InstanceData InstanceQualifierSet InstanceHeap
```

ObjectFlags (section 2.2.6) indicates whether the Decoration (section 2.2.7) block is present as specified in section 2.2.6. When used in EncodingUnitInstanceNoClass, the bit flag CIM class (0x01) MUST NOT be set in ObjectFlags. Other bit flags of ObjectFlags MUST follow the constraints specified in section 2.2.6.

The Decoration block contains the server and CIM namespace from which the object originates.

The EncodingLength field specifies the length, in octets, of itself and all the following fields.

InstanceFlags (section 2.2.54) is reserved and MUST be zero.

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

The values for the properties of an instance are stored in NdTable and InstanceData. The length of NdTable can be calculated as specified in section 2.2.26. Because default values from CIM class definitions can 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.

Any qualifier for the instance or for the properties of instance is stored in InstanceQualifierSet.

The values for any Heap-referenced items anywhere in the EncodingUnitInstanceNoClass encoding block MUST be contained in the InstanceHeap.

Note: EncodingUnitInstanceNoClass contains all the fields of InstanceType except CurrentClass. To minimize the amount of data transmitted, the ClassType is sent the first time using

WBEMOBJECT_INSTANCE, as specified in [MS-WMI] section 2.2.14.3. When instances of the same class need to be transmitted again, they are sent using WBEMOBJECT_INSTANCE_NOCLASS, as specified in [MS-WMI] section 2.2.14.4, which does not have CurrentClass. To encode or decode EncodingUnitInstanceNoClass, the CurrentClass associated with the WBEMOBJECT_INSTANCE_NOCLASS MUST be found as specified in [MS-WMI] section 2.2.14.4. This CurrentClass MUST be inserted into the data after Decoration, and all the data starting with the CurrentClass MUST be encoded or decoded exactly as InstanceType.

2.2.4 ObjectEncodingLength

ObjectEncodingLength is a 32-bit unsigned integer that specifies the length of the ObjectBlock (section 2.2.5).

```
ObjectEncodingLength = UINT32
```

2.2.5 ObjectBlock

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

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

ObjectFlags (section 2.2.6) indicates whether the Decoration (section 2.2.7) block is present and whether the CIM object is a CIM class definition or a CIM instance. The Encoding (section 2.2.10) block contains either a CIM class or CIM instance definition, depending on the value of ObjectFlags.

2.2.6 ObjectFlags

The ObjectFlags block is used to classify the currently encoded object.

```
ObjectFlags = OCTET
```

The octet MUST be a combination of one or more of the following values.

Value	Meaning
0x01	The object is a CIM class. This flag is mutually exclusive with 0x02. If this flag is set, the Encoding (section 2.2.10) block contains ClassType.
0x02	The object is a CIM instance. This flag is mutually exclusive with 0x01. If this flag is set, the Encoding (section 2.2.10) block contains InstanceType.
0x04	If this flag is set, the object has a Decoration block.
0x10	If this flag is set, the object is a prototype of the result object for the query, as specified in [MS-WMI] (section 2.2.4.1). This flag MUST be used only in combination with the 0x01 flag. This flag MUST NOT be used when returning IWbemClassObject, which is not represented as a Prototype Result Object.
0x40	If this flag is set, one or more key properties of the class are not present in the Prototype Result Object. This flag MUST be used only in combination with the 0x01 and 0x10 flags.

ObjectFlags MUST have either the 0x01 or the 0x02 bit set. They are mutually exclusive; both MUST NOT be set simultaneously.

2.2.7 Decoration

The Decoration block is used to optionally decorate the CIM object with information that indicates the server and CIM namespace from which the CIM object originates. This block MUST be present if ObjectFlags (section 2.2.6) has 0x04 set; otherwise, the Decoration block MUST be absent.

```
Decoration = DecServerName DecNamespaceName
```

In the encoded sequence, the strings DecServerName (section 2.2.8) and DecNamespaceName (section 2.2.9) MUST be placed inline. If either string has no value, an empty Encoded-String MUST be present. The two Encoded-String values MUST NOT be omitted, even if empty.

2.2.8 DecServerName

DecServerName is an Encoded-String that represents 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 **Domain Name System (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
```

2.2.9 DecNamespaceName

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

```
DecNamespaceName = Encoded-String
```

2.2.10 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
```

The InstanceType (section 2.2.53) block encodes the CIM instance, and the ClassType (section 2.2.11) block encodes the CIM class object.

2.2.11 ClassType

The ClassType block is used to define a CIM class. It consists of two sequential CIM class definitions. The first block consists of the definition of the **superclass** to the current CIM class. The second block is the actual CIM class definition that is 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 of 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 that is coded in CurrentClass has no superclass. In this case, the ParentClass block MUST be filled with the class name as NULL, zero class names in the derivation list, zero class qualifiers, zero properties, and zero HeapItems in ClassHeap.

2.2.12 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 2.2.14) specifies the properties and method signatures for the class.

2.2.13 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 and 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 in this grammar.

2.2.14 ClassAndMethodsPart

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

- ClassPart (section 2.2.15) contains the data declarations (properties).
- MethodsPart (section 2.2.38) contains the method table.

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

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

MethodsPart (section 2.2.38) MUST always be present if the ObjectFlags (section 2.2.6) value indicates that the outermost object being encoded is a ClassType (section 2.2.11) object. MethodsPart MUST NOT be present if the ObjectFlags indicates that the outermost object being encoded is an InstanceType (section 2.2.53).

2.2.15 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 2.2.16) contains information about the overall ClassPart block length and the length of various internal blocks. The DerivationList (section 2.2.17) 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 2.2.20) is the set of **CIM qualifiers** for the class.

The PropertyLookupTable (section 2.2.21) is a sorted dispatch table for looking up **CIM property** values and type information. The NdTable (section 2.2.26) indicates whether a particular CIM property has a default value that is locally defined in the current CIM class or whether the default is defined in a superclass. The ValueTable (section 2.2.29) contains values inline for simple numeric properties, or references to the values in the ClassHeap (section 2.2.37) for all other specified values in the HeapItem rule, such as arrays or strings.

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

2.2.16 ClassHeader

ClassHeader contains various details on the CIM class block.

```
ClassHeader = EncodingLength ReservedOctet ClassNameRef NdTableValueTableLength
```

The EncodingLength (section 2.2.73) field applies to the ClassPart as a whole, not just the ClassHeader. The ReservedOctet (section 2.2.76) octet is not used and MUST be zero. The ClassNameRef (section 2.2.19) contains a reference to the string that is the name of the current CIM class. The NdTableValueTableLength (section 2.2.28) is sum of the lengths, in octets, of the encoded "ClassPart::NdTable" and "ClassPart::ValueTable" blocks.

2.2.17 DerivationList

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

```
DerivationList = EncodingLength *ClassNameEncoding
```

EncodingLength (section 2.2.73) includes itself in the total. Therefore, an empty array still contains at least one UINT32 value of 0x4 hexadecimal, which is the length of the EncodingLength item. ClassNameEncoding (section 2.2.18) contains the names of the superclasses.

2.2.18 ClassNameEncoding

Each ClassNameEncoding is an Encoded-String that is suffixed by a 32-bit value that indicates the length, in character count, 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
```

2.2.19 ClassNameRef

ClassNameRef is a reference to the current CIM class name. It is a HeapStringRef (section 2.2.68) in the ClassHeap (section 2.2.37).

```
ClassNameRef = HeapStringRef
```

2.2.20 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.

2.2.21 PropertyLookupTable

PropertyLookupTable is a simple dispatching table for finding properties. The PropertyCount (section 2.2.22) indicates how many properties follow in the PropertyLookup (section 2.2.23) sequence.

```
PropertyLookupTable = PropertyCount *PropertyLookup
```

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

2.2.22 PropertyCount

PropertyCount is the total number of properties in the class. If zero, the optional NdTable (section 2.2.26) and ValueTable (section 2.2.29) blocks (as specified in section 2.2.15) MUST be absent.

```
PropertyCount = UINT32
```

2.2.23 PropertyLookup

The PropertyLookup structure represents a data block that allows a lookup of a specific named CIM property in a CIM class. The PropertyNameRef (section 2.2.24) 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 2.2.25) item is a heap reference to a PropertyInfo item that contains more information about the CIM property, such as 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.

2.2.24 PropertyNameRef

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

```
PropertyNameRef = HeapStringRef
```

2.2.25 PropertyInfoRef

PropertyInfoRef MUST be a heap reference to the PropertyInfo (section 2.2.30) item for the property.

```
PropertyInfoRef = HeapRef
```

2.2.26 NdTable

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

Values in the table are ordered similar to the order shown in the PropertyLookupTable.

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

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

In this example, both ValueX and ValueY have defaults; however, they are established in different classes. Because the derived class contains all the information from the base class, the effective class declaration is similar to the following.

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

```
}
```

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

Only 2 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 required octets is the following. When encoding or decoding NdTable under ClassPart, the PropertyCount specified in PropertyLookupTable in the ClassPart MUST be used for calculating length. When encoding or decoding NdTable under InstanceType, the PropertyCount specified in **InstanceType.CurrentClass.ClassPart.PropertyLookupTable** MUST be used.

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

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

2.2.27 NullAndDefaultFlag

NullAndDefaultFlag denotes how the default property value is set and whether that value is NULL.

```
NullAndDefaultFlag = 2BIT
```

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	The default property value is NULL, and it is inherited from a parent class.
SET	NOT SET	The default property value is NULL, and it is set by the current class.
NOT SET	SET	The default property value is NOT NULL, and it is inherited from a parent class.
NOT SET	NOT SET	The default property value is NOT NULL, and it is set by the current class.

For a specified property, if the preceding table shows either of the bit values as SET, the value of that property is predetermined as NULL or is propagated from the parent. In these cases, the value in the ValueTable for that property is ignored.

2.2.28 NdTableValueTableLength

NdTableValueTableLength is sum of the lengths, in octets, of the NdTable and ValueTable.

```
NdTableValueTableLength = UINT32
```

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

2.2.29 ValueTable

The ValueTable encodes the literal values of the properties or references to their values in the heap. However, for a specific property, the value here is relevant only if the corresponding NDTable bits for that property are both not set, that is, 0. Otherwise, the value in ValueTable for the property is ignored.

```
ValueTable = *EncodedValue
```

Depending on the type of the CIM property, each EncodedValue (section 2.2.71) 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 2.2.21) to its PropertyLookup (section 2.2.23), and from there get the PropertyInfoRef (section 2.2.25), which gives the PropertyInfo (section 2.2.30). From PropertyInfo, get the ValueTableOffset (section 2.2.34). Use this offset in the ValueTable (section 2.2.29) to discover the value.

If the value is numerical, the value MUST be directly located within this table. If the value is a string or an array type, the value table MUST contain a reference, HeapRef (section 2.2.69), into the Heap (section 2.2.66) to find the actual value.

ValueTable length can be calculated by NdTableValueTableLength minus the length of NdTable, as specified in section 2.2.26.

When encoding or decoding ValueTable under ClassPart, the NdTableValueTableLength specified in ClassHeader of the ClassPart MUST be used for calculating length. When encoding or decoding ValueTable under InstanceData of InstanceType, the NdTableValueTableLength specified in **InstanceType.CurrentClass.ClassPart.ClassHeader** MUST be used.

2.2.30 PropertyInfo

The PropertyInfo element exists in the heap and is referenced through a PropertyLookup block. It contains information about a property other than its value, such as its data type declaration order, the class in which 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
```

2.2.31 PropertyType

PropertyType encodes the data type of the property.

```
PropertyType = CimType | Inherited
```

PropertyType is obtained by doing a logical OR between CimType and Inherited.

CimType MUST have the form that is specified in section 2.2.82.

2.2.32 Inherited

Inherited indicates the origin of the property.

```
Inherited = 0x4000 / 0x0000
```

If the current property was originally defined in a parent class, Inherited is 0x4000; however, if the property was defined in the current class, Inherited is 0x0000.

2.2.33 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
```

2.2.34 ValueTableOffset

ValueTableOffset MUST be the offset in the ValueTable (section 2.2.29) 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 2.2.31) entry, which is a sibling of this ValueTableOffset in the larger PropertyInfo (section 2.2.30) encoding.

```
ValueTableOffset = UINT32
```

2.2.35 ClassOfOrigin

ClassOfOrigin defines from which CIM class in the DerivationList the CIM property comes, 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
```

2.2.36 PropertyQualifierSet

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

```
PropertyQualifierSet = QualifierSet
```

2.2.37 ClassHeap

ClassHeap is structured like any other heap except that the items that are contained in 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 in the ClassPart (section 2.2.15) block MUST be limited to references in the ClassHeap.

2.2.38 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 that has no methods MUST still contain the indicated fields. MethodCount MUST be zero, and there MUST be a zero-length MethodHeap that is encoded according to their respective rules.

2.2.39 MethodCount

MethodCount is the number of methods in the class.

```
MethodCount = UINT16
```

2.2.40 MethodCountPadding

MethodCountPadding is a two-octet sequence that is not used and SHOULD be set to zero. The recipient MUST ignore this field.<1>

```
MethodCountPadding = 2OCTET
```

2.2.41 MethodDescription

MethodDescription specifies one method.

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

2.2.42 MethodName

MethodName MUST be a simple HeapStringRef to the MethodHeap (section 2.2.52) for the method name.

```
MethodName = HeapStringRef
```

2.2.43 MethodFlags

The MethodFlags block defines the flags for the method.

```
MethodFlags = OCTET
```

The WBEM_FLAVOR_ORIGIN_PROPAGATED flag (0x20) MUST be set if the method is inherited from the parent class. The method origin is calculated for the current ClassAndMethodsPart (as defined in 2.2.14) that is being encoded and is not related to the ClassType being encoded.

The other bits MUST be set to 0.

2.2.44 MethodPadding

MethodPadding is reserved and SHOULD be zero.

```
MethodPadding = 3OCTET
```

Because the fields are not used, some implementations may place random values in these octets; therefore, values other than zero MUST be ignored.

2.2.45 MethodOrigin

MethodOrigin is a zero-origin array index to a CIM class name in the DerivationList that shows which 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.

2.2.46 MethodQualifiers

MethodQualifiers is a set of qualifiers that are applicable to the method.

```
MethodQualifiers = HeapQualifierSetRef
```

MethodQualifiers MUST be a HeapQualifierSetRef (section 2.2.47) in the MethodHeap (section 2.2.52). The QualifierSet (section 2.2.59) referred to by the HeapQualifierSetRef is the CIM qualifiers set that is applicable to the method. For example, in the following CIM class, the execute CIM qualifier and performance CIM qualifier are method-level qualifiers; however, in and out are parameter-level qualifiers.

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

2.2.47 HeapQualifierSetRef

HeapQualifierSetRef MUST be a HeapRef (section 2.2.69) to a single QualifierSet (section 2.2.59) in the current heap.

```
HeapQualifierSetRef = HeapRef
```

2.2.48 InputSignature

InputSignature specifies the input signature for the method.

```
InputSignature = MethodSignature
```

2.2.49 OutputSignature

OutputSignature specifies the output signature for the method.

```
OutputSignature = MethodSignature
```

2.2.50 MethodSignature

The InputSignature and OutputSignature fields MUST be a HeapRef to the MethodSignatureBlock (section 2.2.70) in the MethodHeap (section 2.2.52). 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 2.3.3, MUST be used. These rules do not affect the structure of the encoding; instead, they establish conventions for content, such as the name of the class and how to indicate in and out parameter flow by using qualifiers.

2.2.51 HeapMethodSignatureBlockRef

HeapMethodSignatureBlockRef MUST be a HeapRef to the MethodSignatureBlock (section 2.2.70) in the current Heap (section 2.2.66).

```
HeapMethodSignatureBlockRef = HeapRef
```

2.2.52 MethodHeap

MethodHeap contains information about all the methods, for example, their names, parameters, and types.

```
MethodHeap = Heap
```

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

2.2.53 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 field 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; 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 in the InstanceType encoding block MUST be contained in the InstanceHeap.

2.2.54 InstanceFlags

InstanceFlags is reserved and MUST be zero.

```
InstanceFlags = OCTET
```

2.2.55 InstanceClassName

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

```
InstanceClassName = HeapStringRef
```

2.2.56 InstanceData

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

```
InstanceData = ValueTable
```

2.2.57 InstanceQualifierSet

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

```
InstanceQualifierSet = QualifierSet InstancePropQualifierSet
```

2.2.58 InstanceHeap

InstanceHeap is the value heap for the current instance.

```
InstanceHeap = Heap
```

2.2.59 QualifierSet

QualifierSet represents a set of qualifiers. Qualifiers are applied to a CIM class; to a CIM instance; to properties within a CIM class; or to instances, 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 CIM qualifier value begins where the previous one ends. There are no delimiters between qualifiers; nor is there any indexing mechanism to access a specific qualifier.

Because each CIM qualifier block is 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.

2.2.60 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.

2.2.61 QualifierName

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

```
QualifierName = HeapStringRef
```

Class qualifiers MUST be located in the ClassHeap; CIM instance qualifiers MUST be located in the InstanceHeap; and method qualifiers MUST be located in the MethodHeap.

2.2.62 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.<2>

Qualifier flavor	Corresponding DMTF Qualifier name	Meaning	Bit values
WBEM_FLAVOR_FLAG_PROPAGATE_TO_INSTANCE	WMI specific flavor	If set, the qualifier is propagated to instances. If not set, the qualifier is not propagated to instances.	0x01
WBEM_FLAVOR_FLAG_PROPAGATE_TO_DERIVED_CLASS	ToSubclass	The qualifier is propagated to derived classes.	0x02
WBEM_FLAVOR_NOT_OVERRIDABLE	EnableOverride	The qualifier value cannot be overridden in a derived class or an instance.	0x10
WBEM_FLAVOR_ORIGIN_PROPAGATED	WMI specific flavor	If the qualifier is specified for the property of a class, this flavor means that the property was inherited from the parent class. If the qualifier is specified for the property of a class's instance, this flavor means that the property is inherited from the parent class, but has not been modified at the instance level.	0x20
WBEM_FLAVOR_ORIGIN_SYSTEM	WMI specific flavor	This indicates that the property is a WMI standard property.	0x40
WBEM_FLAVOR_AMENDED	Translatable	The qualifier is localized.	0x80

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

2.2.63 QualifierType

QualifierType is a CIM qualifier and MUST be any valid CimType (section 2.2.82).

```
QualifierType = CimType
```

2.2.64 QualifierValue

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

```
QualifierValue = EncodedValue
```

2.2.65 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 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.

If the flag value is set to 2, all the CIM qualifier sets for all 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 regardless whether there are any qualifiers for that property (there is at least an EncodingLength for that qualifier set).

For examples, see section 3.1.

2.2.66 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 in which the HeapRef occurs.

```
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.

The items appear in any order and do not need to be packed. Heaps MAY contain unused octets between HeapItems. 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.

2.2.67 HeapItem

HeapItem is one of the following data block types. Every HeapItem MUST have a corresponding HeapRef (section 2.2.69).

```
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).

2.2.68 HeapStringRef

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

```
HeapStringRef = HeapRef
```

2.2.69 HeapRef

HeapRef is a reference to any HeapItem and is expressed in 31 bits. If the HeapItem (section 2.2.67) referred to is a string, and the most significant bit of the 32-bit HeapStringRef (section 2.2.68) 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 value of HeapRef is 0xFFFFFFFF, then HeapItem is not present and MUST be considered NULL. If the most significant bit of the 32-bit value is clear, the reference is an offset to a HeapItem in the Heap.

2.2.70 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 2.3.3. MethodSignatureBlock contains ObjectEncodingLength followed by ObjectBlock. ObjectEncodingLength describes the size in octets of ObjectBlock. ObjectBlock encodes the CIM class describing either the input parameters or the output parameters of a method as described in section 2.3.3.

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

2.2.71 EncodedValue

EncodedValue is an encoded value that is used everywhere to represent numerical and string values.

If the value is of type 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 or CIM-TYPE-REAL64 as defined in CimType (section 2.2.82), the EncodedValue is inline as defined in NumericValue (section 2.2.72).

If the value is of type CIM-TYPE-CHAR16, the EncodedValue is a 16-bit value stored as a CIM-TYPE-SINT16.

If the value is of type CIM-TYPE-BOOLEAN, the encoded value is inline as defined in BOOL (section 2.2.75).

If the value is of type CIM-TYPE-STRING, CIM-TYPE-DATETIME, or CIM-TYPE-REFERENCE, the EncodedValue is a HeapRef (section 2.2.69) to that Encoded-String. The value of types CIM-TYPE-DATETIME and CIM-TYPE-REFERENCE are encoded as strings, as specified in 2.3.1 and 2.3.2 respectively.

If the value type is CimArrayType, the EncodedValue is a HeapRef to the Encoded-Array.

If the value type is CIM-TYPE-OBJECT, the EncodedValue is a HeapRef to the object encoded as an ObjectEncodingLength (section 2.2.4) followed by an ObjectBlock (section 2.2.5).

While encoding a class, if a property defined for the class does not have a default value in the class, the OCTETs reserved for the property in the ValueTable MUST be filled with NoValue (section 2.2.74). The number of OCTETs reserved for a property without a default value MUST be equal to the size required for the property type as defined above.

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

2.2.72 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	An 8-bit unsigned integer.
sint8	OCTET, BYTE	An 8-bit signed integer.
uint16	UINT16	A 16-bit unsigned integer.
sint16	SINT16	A 16-bit signed integer.

CIM type as specified in [DMTF-DSP004]	ABNF representation	Binary representation
uint32	UINT32	A 32-bit unsigned integer.
sint32	SINT32	A 32-bit signed integer.
real32	REAL32	A 4-byte floating-point format, as specified in [IEEE754].
real64	REAL64	An 8-byte floating-point format, as specified in [IEEE754].
sint64	SINT64	A 64-bit signed integer.
uint64	UINT64	A 64-bit unsigned integer.

- The binary representations MUST be used to encode the specified CIM data types.
- All signed and unsigned integer types that consist of more than one octet MUST be encoded as little-endian.
- The CIM Boolean type has its own encoding that is specified in the BOOL (section 2.2.75) encoding rule.
- Floating point values, as specified in [IEEE754], MUST be encoded as little-endian.

2.2.73 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
```

2.2.74 NoValue

NoValue is used when a default value does not occur in a CIM class definition for a specific CIM property, and a slot in the ValueTable must be filled for that property. In this, all the OCTETs reserved for the property MUST be set to %xFF, and the value is ignored.

2.2.75 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.

2.2.76 ReservedOctet

ReservedOctet is a reserved OCTET that MUST be set to 0 and is used in several places in the encoding.

```
ReservedOctet = OCTET
```

2.2.77 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 that is being processed conforms to this specification.

2.2.78 Encoded-String

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

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

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 Encoded-String-Flag is set to 0x01 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 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, Encoded-String-Flag 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 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 8 bits are all zero bits. If all the characters for a string have this quality, the string MUST be reduced to its 8-bit equivalent on a character-by-character basis.

This compression technique applies to characters in 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 which characters are present in the string.

2.2.79 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.

2.2.80 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.

2.2.81 BIT

BIT is a simple bit field that consists of 1 bit, either set or clear.

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

It is only used by the NdTable rule.

2.2.82 CimType

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

For any CimType given below, the corresponding values are encoded as specified in EncodedValue (section 2.2.71).

```
CimType = CimBaseType / CimArrayType

CimBaseType = 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 values for the individual types are constants specified in the following table. These values are mutually exclusive to each other.

```
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
```

CimArrayType can be defined in yet another way, as the following example shows.

```
CimArrayType = CimBaseType | CimArrayFlag
; Bitwise OR between a CimBaseType and CimArrayFlag gives
corresponding CimArrayType
```

2.3 Special Data Type Encodings

The various CIM data types have special binary encodings that are implied by the ABNF rules that are specified in sections 2.2.72 and 2.2.79. However, three special cases 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.

2.3.1 CIM DateTime Type

The CIM DateTime type is a string that has the specific format that is 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 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 type as a standard string.

2.3.2 CIM Reference Types

A CIM reference type is a string that contains the CIM object path to another CIM object, as specified in [DMTF-DSP004] section 5.3.2. The CIM reference type is essentially a pointer type that allows 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 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.

2.3.3 CIM Methods

The method signature (that is, the return type) input parameters and output parameters are encoded by 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 preceding CIM class example, 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 that is being encoded; so there is no immutable definition for the class. In this example, the two CIM class definitions 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 input parameters (regardless of where they appear in the method signature) and one definition that encodes all output 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. The ID attribute 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 valid method to reuse the encoding mechanism for classes. Because classes require names, __PARAMETERS is nothing more than that name.

Therefore, a method encoding contains two apparent CIM class definitions (in the InputSignature and OutputSignature rules in 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.

2.3.4 Heap Encoding

HeapItems in the Heap typically occur in any order as long as the Heap references to them (that is, any rules that reduce to HeapRef) are correct. For example, PropertyInfo blocks occur in an order that is different from the lexical order of the properties, and Encoded-String occurs at any location. When updates are being implemented, this implementation of the Heap is intended to allow for best-fit algorithms.

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 heap compaction and HeapRef updates for all heap items after each operation. 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.

The Heap process is important in decoding because code that processes the Heap and HeapItems MUST NOT fault if it encounters blocks that have no reference to them or encounters garbage octets at the end of the Heap.

The client MUST NOT alter a CIM class definition, including its Heap, after 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.

3 Structure 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].

Class base

```
{
      [key]
      sint32  Id;

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

      string   Data2 = "defaultValue";

uint32 Array[ ];
};
```

This example has 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. The individual fields are called properties and are identical to member variables, properties, or fields of object-oriented programming languages such as C++, C#, and Java.

The binary encoding is presented for both classes: first base, and then MyClass. Because each CIM class contains the encoding of itself and its base class, this encoding illustrates all the concepts that are 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 ObjectEncodingLength, see the note in section 2.2.1. The preceding sample ObjectEncodingLength value is 0xD0, which is larger than the actual required number of octets.

The following table decodes base using ABNF.

Relevant offset	Octet values	Comments
		EncodingUnit.
	78 56 34 12	Standard CIM object Signature (line 1).
	D0 00 00 00	EncodingUnit::ObjectEncodingLength 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	The DecServerName (Encoded-String) on lines 2–3 that is the server name decoration that indicates from which machine on the network this CIM object originated. 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	The DecNamespaceName (Encoded-String) that indicates from which CIM namespace the CIM object originates. 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::ClassAndMethodsPart::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 because base is the basest class.
	00 00 00 00	NdTableValueTableLength (italics line 4). This is zero, indicating that there is no NdTable or ValueTable 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 list consists only of the EncodingLength UINT32, it is four octets. The ClassNameEncoding list is empty.
	04 00 00 00	ClassQualifierSet::QualifierSet::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.

Relevant offset	Octet values	Comments
	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 most significant bit set. This indicates a zero-length Heap.
	0C 00 00 00	MethodsPart::EncodingLength. There are 12 octets in the encoding of the method table.
	00 00	MethodsPart::MethodCount. A 16-bit value that indicates how many methods are in the class, or zero.
	00 00	MethodsPart::MethodCountPadding. This is padding and can be zero or any random value.
	00 00 00 80	MethodsPart::MethodHeap. The heap length of the method heap, or zero. The most significant 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 nonexistent 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::NdTableValueTableLength. The NdTable and ValueTable is five octets in length.
	04 00 00 00	DerivationList::EncodingLength. There is no list of superclasses because base is the basest class. The encoding length is only four octets, which is the length of the EncodingLength.
	04 00 00 00	ClassQualifierSet::QualifierSet::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).

Relevant offset	Octet values	Comments
		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. 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 that the property has a default value of NULL; however, the default is not inherited from a superclass.
	FF FF FF FF	EncodedValue::NoValue. No value is defined for the CIM property by default in the CIM class definition.
	3C 00 00 80	ClassHeap::Heap::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. The offset in the ValueTable of the default value, or zero.
	00 00 00 00	PropertyInfo::ClassOfOrigin. A value of zero indicates the current class.
	1C 00 00 00	PropertyQualifierSet::QualifierSet::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 most significant 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).

Relevant offset	Octet values	Comments
		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 the octets in PropertyQualifierSet::QualifierSet::EncodingLength have not yet been completely used.
	01 00 00 80	Qualifier::QualifierName. This is a DictionaryReference instead of a plain HeapRef because the most significant 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	An 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.
	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 most significant 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
```

In the following table, this encoding is decoded by using ABNF. Note that the ParentClass part of this encoding is the same as the CurrentClass part of the base CIM class encoding.

Relevant offset	Octet values	Comments
		EncodingUnit.
	78 56 34 12	Signature.
	23 02 00 00	EncodingUnit::ObjectEncodingLength 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.

Relevant offset	Octet values	Comments
		Decoration.
	00 44 50 52 41 56 41 54 2D 44 56 00	DecServerName. This is an Encoded-String that contains the name of the server that transmitted the CIM object DPRAVAT-DEV.
	00 52 4F 4F 54 00	DecNamespaceName. This is an Encoded-String that contains the CIM namespace. The CIM object was created from ROOT.
		Encoding for the derived CIM class. Base is appended at this location. This is the ClassType::ParentClass block.
	66 00 00 00	ParentClass::ClassAndMethodsPart::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	NdTableValueTableLength. Length of NdTable and ValueTable for properties (five octets).
	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::QualifierSet::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. The name of the 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. The 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. Because there is only one property, only the two least significant bits are valid,

Relevant offset	Octet values	Comments
		and the other bits may be any value. In this case, the bit value 01 indicates that 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 that there is no default value.
	3C 00 00 80	ClassHeap::Heap::HeapLength. The length of heap is 0x3C octets. The most significant 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.
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::QualifierSet::EncodingLength (0x1C octets in length, including itself).
	0A 00 00 80	Qualifier::QualifierName. CimType dictionary entry encoding, signaled by the most significant 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::Qualifier::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. A 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 most

Relevant offset	Octet values	Comments
		significant bit is set (0x80). The dictionary reference is key.
	13	QualifierFlavor.
	0B 00 00 00	QualifierType. CIM-TYPE-BOOLEAN.
	FF FF	An 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	The length of MethodsPart, including itself.
	00 00	MethodCount (zero methods).
	34 00	Two octets of MethodPadding; any values are valid.
	00 00 00 80	MethodHeap::Heap::HeapLength. This is the length of Heap. Zero, with the most significant bit set, as for all heaps.
		Encoding for the 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	The 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	NdTableValueTableLength.
	0E 00 00 00	The DerivationList length, in octets, including itself.
	00 42 61 73 65 00	The Encoded-String base, which is the superclass to this class.
	06 00 00 00	The EncodingLength of the previous string, or six octets (includes both the leading flag and trailing NULL).
	11 00 00 00	The ClassQualifierSet::QualifierSet::EncodingLength length, in octets, including itself.
	09 00	QualifierName, Heap offset to Encoded-String.

Relevant offset	Octet values	Comments
	00 00	
	00	QualifierFlavor. 0 == Local.
	08 00 00 00	QualifierType. 0x8 == CIM-TYPE-STRING.
	16 00 00 00	QualifierValue. A HeapRef to the 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 in this table, regardless of the order in which 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 the CIM property name. Points to the Encoded-String array.
	2E 00 00 00	PropertyInfo::PropertyInfoRef. Offset in Heap of the PropertyInfo table and any associated qualifiers for the property.
	55 00 00 00	PropertyInfo::PropertyNameRef. Offset in Heap to the CIM property name. Points to the Encoded-String Data1.
	5C 00 00 00	PropertyInfo::PropertyInfoRef. Offset in Heap of the PropertyInfo for Data1 and any associated qualifiers.
	99 00 00 00	PropertyInfo::PropertyNameRef. Offset in Heap of the 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 the CIM property name. Points to Id. All properties that are inherited from base classes are repeated in 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; instead, they refer to the propertyIndex field for the CIM property in the PropertyInfo table for that property.
		ValueTable.
0x0	FF FF	No default value.

Relevant offset	Octet values	Comments
	FF FF	
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::Heap::HeapLength. The length is 0x111 octets, and the most significant bit is always set.
0x0	00 4D 79 43 6C 61 73 73 00	Encoded-String MyClass.
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 third 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.

Relevant offset	Octet values	Comments
	01 00 00 00	ClassOfOrigin. Class 1 in DerivationList array.
	11 00 00 00	PropertyQualifierSet::QualifierSet::EncodingLength, including itself.
	0A 00 00 80	Qualifier::QualifierName. This is a DictionaryReference instead of a plain HeapRef because the most significant 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.
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 most significant bit is set (0x80). The dictionary reference is cimtype.
	03	QualifierFlavor.

Relevant offset	Octet values	Comments
	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 most significant bit is set (0x80). The dictionary reference is read.
	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 most significant 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.

Relevant offset	Octet values	Comments
	01 00 00 00	PropertyInfo::ClassOfOrigin points to class[1] in the derivation chain.
		PropertyQualifierSet.
	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 most significant bit is set (0x80). The dictionary reference is cimtype.
	03	Qualifier::QualifierFlavor, 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 most significant bit is set (0x80). The dictionary reference is cimtype.
	23	Qualifier::QualifierFlavor, inherited and toclass+toinstance.

Relevant offset	Octet values	Comments
	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 most significant bit is set (0x80). The dictionary reference is key.
	33	Qualifier::QualifierFlavor = Not overridable/propagated/toclass/toinstance.
	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

3.1 Instance Encoding

Using the CIM class example from section 3 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 ABNF. Note that the shaded part is the ClassPart encoding of the CIM instance and is identical to the preceding table 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 from, or cross-referencing to, a CIM class definition.

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

Relevant offset	Octet values	Comments
		EncodingUnit.

Relevant offset	Octet values	Comments
	78 56 34 12	Standard CIM object Signature.
	D3 01 00 00	UINT32 length of the 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	The Encoded-String that contains the name of the server that transmitted the CIM object DPRAVAT-DEV.
	00 52 4F 4F 54 00	The Encoded-String that contains 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 the CIM class name in heap.
	00	InstanceType::Flags.
	20	InstanceType::NdTabl. 00100000. Indicates the third CIM property has its default value.
	7B 00 00 00	InstanceType::InstanceData::ValueTable. The value for CIM property 0.
	19 00 00 00	InstanceType::InstanceData::ValueTable. The value for Data1.
	00 00 00 00	InstanceType::InstanceData::ValueTable. Data 2 still has the default value.
	09 00 00 00	InstanceType::InstanceData::ValueTable. The location of array for Array.
	04 00 00	InstanceType::InstanceQualifierSet::EncodingLength.

Relevant offset	Octet values	Comments
	00	This indicates that there is no CIM qualifier set data because the EncodingLength only includes its own size.
*** See notes following this table	01	InstPropQualSetFlag. There are no property-level qualifiers.
	26 00 00 80	InstanceHeap::HeapLength The Heap is 0x26 octets in length, and the most significant 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	The Encoded-String of the value StringField.

***In the left column of the previous table, a special case can occur with instances that have qualifiers (see the InstancePropQualifierSet rule), as follows.

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

The encoding for the preceding CIM instance must take into account that a property-level CIM qualifier appears within the instance. When 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 to which the CIM instance belongs, even if one or more of the CIM properties is not used.

The binary encoding for the preceding CIM instance, detailed in the following table, differs from the previous example, which starts in the preceding table at the row above the row that contains the three asterisks (***) in the left column.

	Octet values	Comments
***	02	InstPropQualSetFlag. If the octet value is 02, one QualifierSet per property is encoded at this location prior to the InstanceHeap.
	04 00 00 00	QualifierSet: No qualifiers for CIM property 0 because the EncodingLength is four octets, which is the length of the EncodingLength value itself.
	0F 00 00 00	QualifierSet CIM property 1 (Data1): There are 15 octets for this QualifierSet.
	26 00 00 00	QualifierName: The Heap reference to the 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. The Heap is longer in order to accommodate the name of the CIM qualifier test.
	...	The remainder of the Heap.

3.2 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 not part of a CIM instance encoding—even though instances do contain parts of the CIM class encoding that is 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 3.1.

```
class MyClass2  :  MyClass
{
  [execute, performance{"fast", "sideffects"}]
   uint32 Restart([in] string ServiceName, [out] sint32 Status);
};
```

The raw hexadecimal encoding of the preceding CIM class definition is shown as follows. The shaded block is the encoding of the parent CIM class MyClass, which is the same as is shown in the Encoded Examples topic.

```
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 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 80 0c 00 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 34 12	Standard CIM object signature.
	D3 01 00 00	UINT32 length of the entire CIM class encoding (0x1D3 octets).
		ObjectBlock.
	Shaded Bold	MyClass encoding.

Relevant offset	Octet values	Comments
	Portion 76 01 00 00... ...73 00 00 00 80	
	80 01 00 00	EncodingLength of MyClass2, or 0x180 octets.
	00	Reserved.
	00 00 00 00	The offset of the CIM class name in the CIM class Heap.
	11 00 00 00	NdTableValueTableLength.
	1B 00 00 00	The DerivationList length, in octets.
	00 4D 79 43 6C 61 73 73 00	The Encoded-String MyClass, which is the superclass of the current class.
	09 00 00 00	The length, in octets, of the previous Encoded-String.
	00 42 61 73 65 00	The Encoded-String base, which is the superclass of MyClass.
	06 00 00 00	The length, in octets, of the previous Encoded-String.
	04 00 00 00	ClassQualifierSet::QualifierSet::EncodingLength, including itself. There are no qualifiers; so the octet count is four, which is just enough to encode its own length.
	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.

Relevant offset	Octet values	Comments
	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::Heap::HeapLength (0x11B octets, 283 decimal).
	283 octets in shaded italics 00 4d 79... ...00 00 00	ClassHeap. This is essentially the same encoding that is used for MyClass because MyClass2 only adds a method and no properties.
	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 MethodHeap; 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 the most significant bit set.

Relevant offset	Octet values	Comments
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 previous raw hexadecimal encoding.
	05	ObjectFlags (Class + Decoration).
	00 44 50 2d 4d 00	DecServerName: The Encoded-String of DP-M.
	00 52 4F 4F 54 5C 64 65 66 61 75 6C 74 00	DecNamespaceName: Encoded-String ROOT\default.
		The CIM class encoding for __PARAMETERS begins here. There are ParentClass and CurrentClass parts, in succession. The ParentClass is almost empty.
	1d 00 00 00	ClassHeader::EncodingLength. (ParentClass part of ClassType.)
	00	ReservedOctet.
	ff ff ff ff	ClassNameRef. This is a simulation of a Heap reference. It indicates no CIM class name because the parent CIM class of __PARAMETERS is currently being encoded and does not exist.
	00 00 00 00	NdTableValueTableLength.
	04 00 00 00	DerivationList length in octets, including itself (no Derivation).
	04 00 00 00	ClassQualifierSet::QualifierSet::EncodingLength, including itself. (No CIM qualifier set.)
	00 00 00 00	PropertyCount.
	00 00 00 80	HeapLength (zero, most significant bit set).
	0C 00 00 00	MethodsPart::EncodingLength.
	00 00	MethodsPart::MethodCount.

Relevant offset	Octet values	Comments
	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	NdTableValueTableLength.
	04 00 00 00	DerivationList (only itself, so no derivation).
	0F 00 00 00	ClassQualifierSet::QualifierSet::EncodingLength.
	0E 00 00 00	The 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::Heap::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	Encoded-String abstract.

Relevant offset	Octet values	Comments
	61 63 74 00	
18	08 00 00 00 00 00 00 00 00 00 00 00 00 00 04 00 00 00	Unused. This is part of Heap fragmentation and can be removed if the Heap offsets for all subsequent HeapItems are adjusted.
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	Unreferenced Heap fragment lost from previous editing or updates.

Relevant offset	Octet values	Comments
	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	
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.

Relevant offset	Octet values	Comments
	8C 00 00 00	QualifierName (points to ID, an attribute added to all properties that are acting as parameters in a method).
	11	QualifierFlavor.
11F	03 00 00 00	QualifierType (CIM-TYPE-SINT32).
	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 the most significant 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 in the same way as the Encoding block __PARAMETERS for the input parameters (octet FC 01 00 00 above).
	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.

Relevant offset	Octet values	Comments
	29 05 00 00	Location of string array in Heap.
	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	The location of the first string in the array (fast).
	3B 05 00 00	The location of the second string in the 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 00 29 05... ...74 73 00 00 00 00 00	The remaining octets are in italics. The remainder of the 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.

4 Security Considerations

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

5 Appendix A: Product Behavior

The information in this specification is applicable to the following Microsoft products or supplemental software. References to product versions include released service packs:

- Microsoft Windows® 2000 operating system
- Windows® XP 64-Bit Edition operating system
- Windows® XP operating system
- Windows Vista® operating system
- Windows Server® 2003 operating system
- Windows Server® 2008 operating system
- Windows® 7 operating system
- Windows Server® 2008 R2 operating system

Exceptions, if any, are noted below. If a service pack or Quick Fix Engineering (QFE) number appears with the product version, behavior changed in that service pack or QFE. The new behavior also applies to subsequent service packs of the product unless otherwise specified. If a product edition appears with the product version, behavior is different in that product edition.

Unless otherwise specified, any statement of optional behavior in this specification that is prescribed using the terms SHOULD or SHOULD NOT implies product behavior in accordance with the SHOULD or SHOULD NOT prescription. Unless otherwise specified, the term MAY implies that the product does not follow the prescription.

<1> Section 2.2.40: Windows 2000, Windows XP, Windows Server 2003, Windows Vista, Windows Server 2008, Windows 7, and Windows Server 2008 R2 send random data in this field.

<2> Section 2.2.62: Windows ignores other bit values.

6 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 NdTableValueTableLength
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 = 2BIT
  ;nullness = bit 0, inheritedDefault = bit1
NdTableValueTableLength = 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
Encoded-String = Encoded-String-Flag *Character Null;
Encoded-String-Flag = OCTET
Character = AnsiCharacter / UnicodeCharacter
Null = Character
AnsiCharacter = OCTET
UnicodeCharacter = 2OCTET
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 = 4OCTET
SINT32 = 4OCTET
UINT64 = 8OCTET
SINT64 = 8OCTET
REAL32 = 4OCTET             ; IEEE short floating-point format
REAL64 = 8OCTET             ; IEEE format
UINT16 = 2OCTET
SINT16 = 2OCTET
BOOL = 2OCTET                  ;*
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-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-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
```

7 Change Tracking

This section identifies changes that were made to the [MS-WMIO] protocol document between the January 2011 and February 2011 releases. Changes are classified as New, Major, Minor, Editorial, or No change.

The revision class **New** means that a new document is being released.

The revision class **Major** means that the technical content in the document was significantly revised. Major changes affect protocol interoperability or implementation. Examples of major changes are:

- A document revision that incorporates changes to interoperability requirements or functionality.
- An extensive rewrite, addition, or deletion of major portions of content.
- The removal of a document from the documentation set.
- Changes made for template compliance.

The revision class **Minor** means that the meaning of the technical content was clarified. Minor changes do not affect protocol interoperability or implementation. Examples of minor changes are updates to clarify ambiguity at the sentence, paragraph, or table level.

The revision class **Editorial** means that the language and formatting in the technical content was changed. Editorial changes apply to grammatical, formatting, and style issues.

The revision class **No change** means that no new technical or language changes were introduced. The technical content of the document is identical to the last released version, but minor editorial and formatting changes, as well as updates to the header and footer information, and to the revision summary, may have been made.

Major and minor changes can be described further using the following change types:

- New content added.
- Content updated.
- Content removed.
- New product behavior note added.
- Product behavior note updated.
- Product behavior note removed.
- New protocol syntax added.
- Protocol syntax updated.
- Protocol syntax removed.
- New content added due to protocol revision.
- Content updated due to protocol revision.
- Content removed due to protocol revision.

- New protocol syntax added due to protocol revision.
- Protocol syntax updated due to protocol revision.
- Protocol syntax removed due to protocol revision.
- New content added for template compliance.
- Content updated for template compliance.
- Content removed for template compliance.
- Obsolete document removed.

Editorial changes are always classified with the change type **Editorially updated.**

Some important terms used in the change type descriptions are defined as follows:

- **Protocol syntax** refers to data elements (such as packets, structures, enumerations, and methods) as well as interfaces.
- **Protocol revision** refers to changes made to a protocol that affect the bits that are sent over the wire.

The changes made to this document are listed in the following table. For more information, please contact protocol@microsoft.com.

Section	Tracking number (if applicable) and description	Major change (Y or N)	Change type
2.2.78 Encoded-String	59078 Simplified requirement to reduce sixteen-bit characters to eight-bit when first eight bits are zero for all characters in the string.	Y	Content updated.

8 Index

P

Q

R

S

T

V