CIM Data Model and the OpenPegasus Common Library

Roger Kumpf

Hewlett-Packard

Module Content

CIM Standard Data Types

- **CIM Data Types**
- CIM Model Components

Copyright © 2003 Hewlett-Packard Development Company, L. P.

CIM Data Types

CIM Data Type	Interpretation	Pegasus Type
boolean	Boolean	Boolean
Uint8	Unsigned 8-bit integer	Uint8
sint8	Signed 8-bit integer	Sint8
uint16	Unsigned 16-bit integer	Uint16
sint16	Signed 16-bit integer	Sint16
uint32	Unsigned 32-bit integer	Uint32
sint32	Signed 32-bit integer	Sint32
uint64	Unsigned 64-bit integer	Uint64
sint64	Signed 64-bit integer	Sint64
real32	IEEE 4-byte floating-point	Real32
real64	IEEE 8-byte floating-point	Real64
char16	16-bit UCS-2 character	Char16
string	UCS-2 string	String
datetime	A date-time formatted string	CIMDateTime
<classname>ref	Strongly typed reference	CIMObjectPath

Copyright © 2003 Hewlett-Packard Development Company, L. P.

Char16

Intrinsic Data Type	Interpretation
char16	16-bit UCS-2 character

```
'A', '&', 'z',
'\x0032', 0x32,
50
```

Escape Sequence		Interpretation
\b	\x0008	Backspace
\t	\x0009	Horizontal tab
\n	\x000A	Linefeed
\f	\x000C	Form feed
\r	\x000D	Carriage return
\"	\x0022	Double quote
\'	\x0027	Single quote
\\	\x005C	Backslash
\x<hex>		<hex> is one to four hex digits
\X<hex>		<hex> is one to four hex digits

Copyright © 2003 Hewlett-Packard Development Company, L. P.

Char16 API and Examples

```
Char16();.
Char16(Uint16 x);
Char16(const Char16& x);
~Char16();
Char16& operator=(Uint16 x);
Char16& operator=(const Char16& x);
operator Uint16() const;

Boolean operator==(const Char16& x, const Char16& y);
Boolean operator==(const Char16& x, char y);
Boolean operator==(char x, const Char16& y);
Boolean operator!=(const Char16& x, const Char16& y);
Boolean operator!=(const Char16& x, char y);
Boolean operator!=(char x, const Char16& y);
```

```
Char16 char16Value1;
Char16 char16Value2 = 'a';

Uint16 uint16Value1 = 97;
Char16 char16Value3 = uint16Value1;

char charValue2 = 'a';
assert(char16Value2 == charValue2);
assert(char16Value2 == char16Value3);

char16Value1 = '&';
char16Value1 = '\x0032';
char16Value1 = 0x32;
```


Copyright © 2003 Hewlett-Packard Development Company, L. P.

String

Intrinsic Data Type	Interpretation
string	UCS-2 string

A string value is a sequence of zero or more UCS-2 characters enclosed in double-quotes (").

- A double-quote is allowed within the value as long as it is preceded immediately by a backslash (\).
- Escape sequences are permitted as with character values.
- Successive quoted strings are concatenated when only white space or a comment intervenes.

```
"this is a \"string\""
"this" " is a \"string\""
"this" /*comment*/ " is a \"string\""
```


Copyright © 2003 Hewlett-Packard Development Company, L. P.

String API

String Constructors

```
static const String EMPTY;
String();
String(const String& str);
String(const String& str, Uint32 n);
String(const Char16* str);
String(const Char16* str, Uint32 n);
String(const char* str);
String(const char* str, Uint32 n);
String& operator=(const String& str);
String& assign(const String& str);
String& assign(const Char16* str);
String& assign(const Char16* str, Uint32 n);
String& assign(const char* str);
String& assign(const char* str, Uint32 n);
```

String Destructor

```
~String();
```

```
void clear();
void reserveCapacity(Uint32 capacity);
Uint32 size() const;
```


Copyright © 2003 Hewlett-Packard Development Company, L. P.

String API

```
const Char16* getChar16Data() const;
CString getCString() const;
Char16& operator[](Uint32 index);
const Char16 operator[](Uint32 index) const;
```

```
String& append(const Char16& c);
String& append(const Char16* str, Uint32 n);
String& append(const String& str);
void remove(Uint32 index, Uint32 size = PEG_NOT_FOUND);
```

```
String subString(Uint32 index, Uint32 length = PEG_NOT_FOUND) const;
Uint32 find(Char16 c) const;
Uint32 find(Uint32 index, Char16 c) const;
Uint32 find(const String& s) const;
Uint32 reverseFind(Char16 c) const;
void toLower();
```

```
PEGASUS_STD(ostream)& operator<<(
   PEGASUS_STD(ostream)& os,
   const String& str);
String operator+(const String& str1, const String& str2);
```


Copyright © 2003 Hewlett-Packard Development Company, L. P.

String API

Comparison Operators

```
static int compare(const String& s1, const String& s2, Uint32 n);
static int compare(const String& s1, const String& s2);
static int compareNoCase(const String& s1, const String& s2);
static Boolean equal(const String& str1, const String& str2);
static Boolean equalNoCase(const String& str1, const String& str2);
Boolean operator==(
   const String& str1,
   const String& str2);
Boolean operator==(const String& str1, const char* str2);
Boolean operator==(const char* str1, const String& str2);
Boolean operator!=(
   const String& str1,
   const String& str2);
```

```
Boolean operator<(
   const String& str1,
   const String& str2);
Boolean operator<=(
   const String& str1,
   const String& str2);
Boolean operator>(
   const String& str1,
   const String& str2);
Boolean operator>=(
   const String& str1,
   const String& str2);
```


Copyright © 2003 Hewlett-Packard Development Company, L. P.

String Examples

```
static const String EMPTY;
String();
String(const String& str);
String(const String& str, Uint32 n);
String(const Char16* str);
String(const Char16* str, Uint32 n);
String(const char* str);
String(const char* str, Uint32 n);
String& operator=(const String& str);
```

```
String s1;
assert(s1 == String::EMPTY);
s1 = "";
assert(s1 == String::EMPTY);

String s2 = "Hello World";
String s3 = s2;
String s4(s3);
assert(String::equal(s2, s4));

String s5("123456", 3);
assert(String::equal(s5, "123"));

s2.clear();
assert(s1 == s2);
assert(s2.size() == 0);

s1 = "this is a \"string\"";
s2 = "this" " is a \"string\"";
s3 = "this " /*comment*/ "is a \"string\"";
assert(s1.size() == 18);
assert(s1 == s2);
assert(s2 == s3);
```


Copyright © 2003 Hewlett-Packard Development Company, L. P.

String Examples

```
String s1 = "String1";
String s2, s3;
String s4 = s1;
s2.assign(s1);
s3.assign(s1);

assert(s1 == s2);
assert(s2 == s3);

s1[0] = 's';
assert(!(s1 == "String1"));
assert(s1 == "string1");
assert(s2 == "String1");
assert(s3 == "String1");
assert(s4 == "String1");
```

```
String& assign(const String& str);
String& assign(const Char16* str);
String& assign(const Char16* str, Uint32 n);
String& assign(const char* str);
String& assign(const char* str, Uint32 n);
```


Copyright © 2003 Hewlett-Packard Development Company, L. P.
hp invent

String Examples

```
try
{
   String s1 = "abc";
   assert(s1.size() == 3);
   Boolean exceptionCaught = false;
   try
   {
      Char16 c = s1[10];
   }
   catch (IndexOutOfBoundsException&)
   {
      exceptionCaught = true;
   }
   assert(true);
}
catch(Exception& e)
{
    cerr << "test02 Error: " << e.getMessage() << endl;
    exit(1);
}
```


Copyright © 2003 Hewlett-Packard Development Company, L. P.

String Examples

```
PEGASUS_STD(ostream)& operator<<(
   PEGASUS_STD(ostream)& os,
   const String& str);
String operator+(const String& str1, const String& str2);
```

```
String s1 = "Hello";
s1.append(Char16(0x0000));
s1.append(Char16(0x1234));
s1.append(Char16(0x5678));
s1.append(Char16(0x9cde));
s1.append(Char16(0xffff));

ostrstream os;
os << s1;
os.put('\0');
const char EXPECTED[] = "Hello\\x0000\\x1234\\x5678\\x9CDE\\xFFFF";
char* tmp = os.str();
assert(strcmp(EXPECTED, tmp) == 0);
delete tmp;
```


Copyright © 2003 Hewlett-Packard Development Company, L. P.

String Examples

```
void remove(Uint32 index, Uint32 size = PEG_NOT_FOUND);
```

```
String s = "abcdefg";
s.remove(3, 3);
assert(String::equal(s, "abcg"));
assert(s.size() == 4);

s = "abcdefg";
s.remove(3, 4);
assert(String::equal(s, "abc"));
assert(s.size() == 3);

s = "abcdefg";
s.remove(3);
assert(String::equal(s, "abc"));
assert(s.size() == 3);

s = "abc";
s.remove(3);
assert(String::equal(s, "abc"));
assert(s.size() == 3);
```

```
s = "abc";
s.remove(0);
assert(String::equal(s, ""));
assert(s.size() == 0);

s = "abc";
s.remove(0, 1);
assert(String::equal(s, "bc"));
assert(s.size() == 2);
```


Copyright © 2003 Hewlett-Packard Development Company, L. P.

String Examples

```
Uint32 find(Char16 c) const;
Uint32 find(Uint32 index, Char16 c) const;
Uint32 find(const String& s) const;
```

```
String t1 = "abcdef";
String t2 = "cde";
String t3 = "xyz";
String t4 = "abc";
String t5 = "abd";
String t6 = "defg";
assert(t1.find('c') == 2);
assert(t1.find(t2)==2);
assert(t1.find(t3)==(Uint32)-1);
assert(t1.find(t4)==0);
assert(t1.find(t5)==(Uint32)-1);
assert(t1.find(t6)==(Uint32)-1);
assert(t1.find("cde")==2);
assert(t1.find("def")==3);
assert(t1.find("xyz")==(Uint32)-1);
assert(t1.find("a") ==0);
```

```
String s = "this is an apple";
assert(s.find("apple")==11);
assert(s.find("appld")==(Uint32)-1);
assert(s.find("this")==0);
assert(s.find("t")==0);
assert(s.find("e")==15);
s = "a";
assert(s.find("b")==(Uint32)-1);
assert(s.find("a")==0);
assert(s.find(s)==0);
s = "aaaapple";
assert(s.find("apple")==3);
```


Copyright © 2003 Hewlett-Packard Development Company, L. P.

CString API

```
CString();
CString(const CString& cstr);
~CString();
CString& operator=(const CString& cstr);
operator const char*() const;
```

```
const char STR0[] = "one two three four";
String s = STR0;
assert(strcmp(s.getCString(), STR0) == 0);
```


Copyright © 2003 Hewlett-Packard Development Company, L. P.

Datetime Value

Intrinsic Data Type	Interpretation
datetime	A date-time formatted string

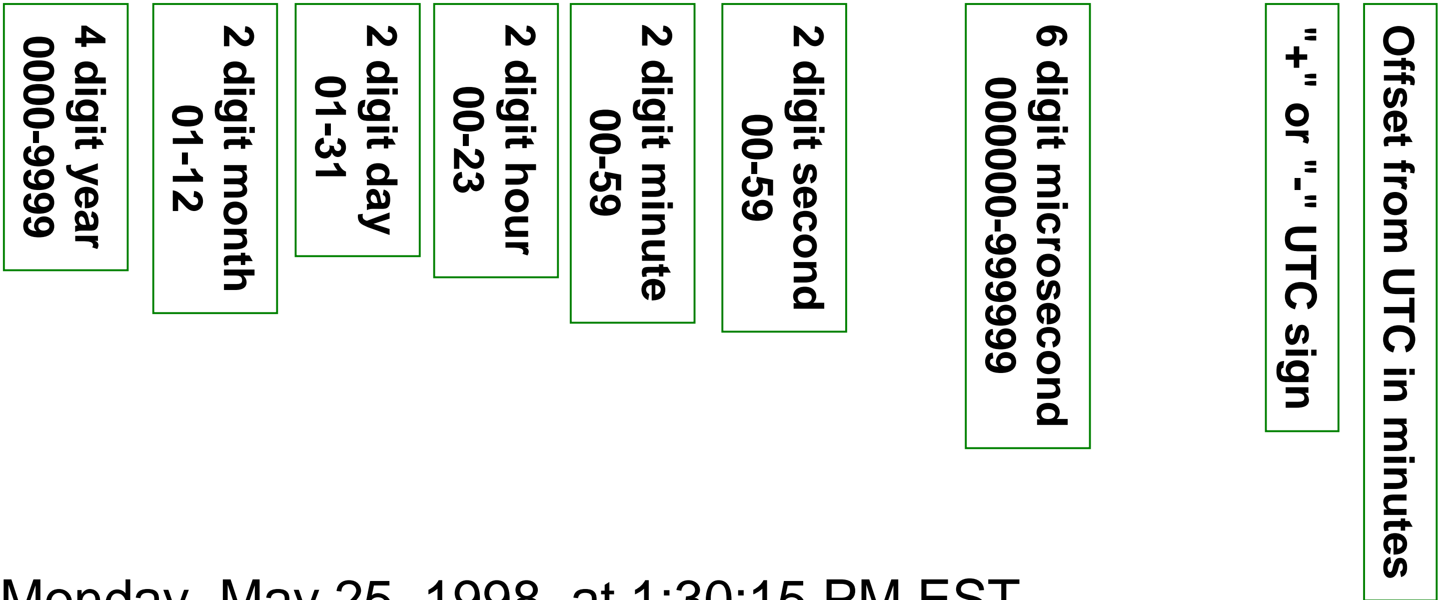
A Datetime value is represented as a formatted string. Values must be zero-padded so that the entire string is always the same 25-character length.

A Datetime may be used to represent an absolute time or a time interval.

Coordinated Universal Time (UTC) is the international time standard used to indicate the time zone for an absolute time. It is the current term for what was commonly referred to as Greenwich Meridian Time (GMT).

Copyright © 2003 Hewlett-Packard Development Company, L. P.

Datetime Value – Absolute Time



Monday, May 25, 1998, at 1:30:15 PM EST
19980525133015.000000-300

Copyright © 2003 Hewlett-Packard Development Company, L. P.
hp invent

Datetime Value – Time Interval

ddddddddhhmmss.mmmmmm:000

- 8 digit days
- 2 digit hours
- 2 digit minutes
- 2 digit seconds
- 6 digit microseconds
- Always 000

The interpretation of the fields in an interval is based on elapsed time.

42 days, 13 hours, 8 minutes, and 12 seconds
00000042130812.000000:000

Copyright © 2003 Hewlett-Packard Development Company, L. P.

CIMDateTime API

```
CIMDateTime();
CIMDateTime(const String & str);
CIMDateTime(const CIMDateTime& x);
CIMDateTime& operator=(const CIMDateTime& x);
```

```
~CIMDateTime();
```

```
String toString () const;
void set(const String & str);
void clear();
```

```
static CIMDateTime getCurrentDateTime();
static Sint64 getDifference(CIMDateTime startTime, CIMDateTime finishTime);
Boolean isInterval();
Boolean equal (const CIMDateTime & x) const;
Boolean operator==(const CIMDateTime& x, const CIMDateTime& y);
```


Copyright © 2003 Hewlett-Packard Development Company, L. P.

CIMDateTime Examples

```
CIMDateTime();
CIMDateTime(const String & str);
CIMDateTime(const CIMDateTime& x);
CIMDateTime& operator=(const CIMDateTime& x)

void set(const String & str);
void clear();
```

```
CIMDateTime dt1, dt2;
 dt1.set("19991224120000.000000+360");
 dt2 = dt1;
 assert(dt1.equal(dt2));

 dt1.clear();
 assert(dt1.equal(CIMDateTime("00000000000000.000000:000")));
```


Copyright © 2003 Hewlett-Packard Development Company, L. P.

CIMDateTime Examples

```
try
{
   Boolean exceptionCaught = false;
   try
   {
      CIMDateTime dt;

      dt.set("too short");
    }
    catch (InvalidDateTimeFormatException&)
    {
      exceptionCaught = true;
    }
    assert(exceptionCaught);
}
catch(Exception& e)
{
    cerr << "test02 Error: " << e.getMessage() << endl;
    exit(1);
}
```


Copyright © 2003 Hewlett-Packard Development Company, L. P.

hp invent

CIMDateTime Examples

```
try
{
   Boolean exceptionCaught = false;
   try
   {
      CIMDateTime dt;

      dt.set("19990132120000+360");
    }
    catch (InvalidDateTimeFormatException&)
    {
      exceptionCaught = true;
    }
    assert(exceptionCaught);
}
catch(Exception& e)
{
    cerr << "test02 Error: " << e.getMessage() << endl;
    exit(1);
}
```


Copyright © 2003 Hewlett-Packard Development Company, L. P.

CIMDateTime Examples

```
static Sint64 getDifference(CIMDateTime startTime, CIMDateTime finishTime);
```

```
CIMDateTime startTime, finishTime;

startTime.set("20020507170000.000000-480");
finishTime.set("20020507170000.000000-300");

Sint64 differenceInMicroseconds =
  CIMDateTime::getDifference(startTime, finishTime);

assert (differenceInMicroseconds ==
    -PEGASUS_SINT64_LITERAL(10800000000));
```


Copyright © 2003 Hewlett-Packard Development Company, L. P.

hp invent

CIMDateTime Examples

```
Boolean isInterval();
```

```
CIMDateTime     startInterval;
CIMDateTime     finishInterval;
Sint64          intervalDifferenceInMicroseconds;

startInterval.set("00000001010100.000000:000");
finishInterval.set("00000001010200.000000:000");

intervalDifferenceInMicroseconds = CIMDateTime::getDifference
                        (startInterval, finishInterval);

assert(startInterval.isInterval());
assert(intervalDifferenceInMicroseconds == 60000000 );
```


Copyright © 2003 Hewlett-Packard Development Company, L. P.

Reference Value

A Reference Value contains the name of a Class or Instance. This value is generally referred to as an Object Path.

Object Paths come in many forms. When the context is such that the Object Path is known to refer to the local Host or Namespace, those values may be omitted.

When an Object Path refers to a Class, no Key Bindings are specified.

Object Path = [[Host] + Namespace] +
Class Name +
zero or more Key Bindings

Copyright © 2003 Hewlett-Packard Development Company, L. P.

Reference Value – MOF

Object Path = [[Host] + Namespace] +
Class Name +
zero or more Key Bindings

Reference values are converted to a String format and enclosed in double-quotes when encoded in MOF:

"[[//<hostname>] /<namespace>:]
<classname> [.<keyname>=<value>
(,<keyname>=<value>)*]"

```
"/root/cimv2:MyClass"
"//myhost.com/root/cimv2:MyClass.Key=Instance1"
"MyClass.Key1=123,Key2=true"
```


Copyright © 2003 Hewlett-Packard Development Company, L. P.

CIMKeyBinding API

CIMKeyBinding Constructor

```
enum Type { BOOLEAN, STRING, NUMERIC, REFERENCE };
CIMKeyBinding();
CIMKeyBinding(const CIMKeyBinding& x);
CIMKeyBinding(const CIMName& name, const String& value, Type type);
CIMKeyBinding(const CIMName& name, const CIMValue& value);
```

CIMKeyBinding Destructor

```
~CIMKeyBinding();
```

```
CIMKeyBinding& operator=(const CIMKeyBinding& x);
const CIMName& getName() const;
void setName(const CIMName& name);
const String& getValue() const;
void setValue(const String& value);
Type getType() const;
void setType(Type type);
Boolean equal(CIMValue value);
Boolean operator==(
    const CIMKeyBinding& x,
    const CIMKeyBinding& y);
```


Copyright © 2003 Hewlett-Packard Development Company, L. P.

CIMObjectPath API

CIMObjectPath Constructor

```
CIMObjectPath();
CIMObjectPath(const CIMObjectPath& x);
CIMObjectPath(const String& objectName);
CIMObjectPath(
        const String& host,
        const CIMNamespaceName& nameSpace,
        const CIMName& className,
        const Array<CIMKeyBinding>& keyBindings = Array<CIMKeyBinding>());
```

CIMObjectPath Destructor

```
~CIMObjectPath();
```

```
CIMObjectPath& operator=(const CIMObjectPath& x);
void clear();
void set(
        const String& host,
        const CIMNamespaceName& nameSpace,
        const CIMName& className,
        const Array<CIMKeyBinding>& keyBindings = Array<CIMKeyBinding>());
void set(const String& objectName);
CIMObjectPath& operator=(const String& objectName);
```


Copyright © 2003 Hewlett-Packard Development Company, L. P.

CIMObjectPath API

```
const String& getHost() const;
void setHost(const String& host);
const CIMNamespaceName& getNameSpace() const;
void setNameSpace(const CIMNamespaceName& nameSpace);
const CIMName& getClassName() const;
void setClassName(const CIMName& className);
const Array<CIMKeyBinding>& getKeyBindings() const;
void setKeyBindings(const Array<CIMKeyBinding>& keyBindings);
String toString() const;
Boolean identical(const CIMObjectPath& x) const;
Uint32 makeHashCode() const;
```

```
Boolean operator==(
   const CIMObjectPath& x,
   const CIMObjectPath& y);
Boolean operator!=(
   const CIMObjectPath& x,
   const CIMObjectPath& y);
Boolean operator==(
   const CIMObjectPath& x,
   const CIMObjectPath& y);
Boolean operator!=(
   const CIMObjectPath& x,
   const CIMObjectPath& y);
```


Copyright © 2003 Hewlett-Packard Development Company, L. P.

Array

```
Array();
Array(const Array<PEGASUS_ARRAY_T>& x);
Array(Uint32 size);
Array(Uint32 size, const PEGASUS_ARRAY_T& x);
Array(const PEGASUS_ARRAY_T* items, Uint32 size);
Array<PEGASUS_ARRAY_T>& operator=(const Array<PEGASUS_ARRAY_T>& x);
```

```
~Array();
```

```
void append(const PEGASUS_ARRAY_T& x);
void append(const PEGASUS_ARRAY_T* x, Uint32 size);
void appendArray(const Array<PEGASUS_ARRAY_T>& x);
void prepend(const PEGASUS_ARRAY_T& x);
void prepend(const PEGASUS_ARRAY_T* x, Uint32 size);
void insert(Uint32 index, const PEGASUS_ARRAY_T& x);
void insert(Uint32 index, const PEGASUS_ARRAY_T* x, Uint32 size);
void remove(Uint32 index);
void remove(Uint32 index, Uint32 size);
```


Copyright © 2003 Hewlett-Packard Development Company, L. P.
hp invent

Array

```
void clear();
void reserveCapacity(Uint32 capacity);
void grow(Uint32 size, const PEGASUS_ARRAY_T& x);
void swap(Array<PEGASUS_ARRAY_T>& x);
```

```
Uint32 size() const;
Uint32 getCapacity() const;
const PEGASUS_ARRAY_T* getData() const;
PEGASUS_ARRAY_T& operator[](Uint32 index);
const PEGASUS_ARRAY_T& operator[](Uint32 index) const;
```


Copyright © 2003 Hewlett-Packard Development Company, L. P.

Module Content

CIM Standard Data Types

- CIM Data Types
- **CIM Model Components**

Copyright © 2003 Hewlett-Packard Development Company, L. P.

CIM Model Components

CIM Value

CIM Name

CIM Qualifier Scope

CIM Qualifier Flavor

CIM Qualifier

CIM Property

CIM Method

CIM Parameter

CIM Parameter Value

CIM Class

CIM Instance

Copyright © 2003 Hewlett-Packard Development Company, L. P.

CIM Value

A CIM value may represent any one of the intrinsic data types or an array of entries of a given intrinsic type. An array may have zero or more elements.

If no value is provided for a given CIM value, its value is implicitly NULL.

Copyright © 2003 Hewlett-Packard Development Company, L. P.

CIMValue API

CIMValue Constructors

```
CIMValue();
CIMValue(CIMType type, Boolean isArray, Uint32 arraySize = 0);
CIMValue(Boolean x);
CIMValue(Uint8 x);
CIMValue(Sint8 x);
CIMValue(Uint16 x);
CIMValue(Sint16 x);
CIMValue(Uint32 x);
CIMValue(Sint32 x);
CIMValue(Uint64 x);
CIMValue(Sint64 x);
CIMValue(Real32 x);
CIMValue(Real64 x);
CIMValue(const Char16& x);
CIMValue(const String& x);
CIMValue(const CIMDateTime& x);
CIMValue(const CIMObjectPath& x);
CIMValue(const CIMValue& x);
CIMValue& operator=(const CIMValue& x);
void assign(const CIMValue& x);
```

```
CIMValue(const Array<Boolean>& x);
CIMValue(const Array<Uint8>& x);
CIMValue(const Array<Sint8>& x);
CIMValue(const Array<Uint16>& x);
CIMValue(const Array<Sint16>& x);
CIMValue(const Array<Uint32>& x);
CIMValue(const Array<Sint32>& x);
CIMValue(const Array<Uint64>& x);
CIMValue(const Array<Sint64>& x);
CIMValue(const Array<Real32>& x);
CIMValue(const Array<Real64>& x);
CIMValue(const Array<Char16>& x);
CIMValue(const Array<String>& x);
CIMValue(const Array<CIMDateTime>& x);
CIMValue(const Array<CIMObjectPath>& x);
```

CIMValue Destructor

```
~CIMValue();
```


Copyright © 2003 Hewlett-Packard Development Company, L. P.
invent

CIMValue API

```
void clear();
Boolean typeCompatible(const CIMValue& x) const;
Boolean isArray() const;
Boolean isNull() const;
Uint32 getArraySize() const;
CIMType getType() const;
Boolean equal(const CIMValue& x) const;
String toString() const;


Boolean operator==(const CIMValue& x, const CIMValue& y);
Boolean operator!=(const CIMValue& x, const CIMValue& y
```


Copyright © 2003 Hewlett-Packard Development Company, L. P.

CIMValue API

```
void setNullValue(CIMType type, Boolean isArray, Uint32 arraySize = 0);
void set(Boolean x);
 void set(Uint8 x);
void set(Sint8 x);
void set(Uint16 x);
void set(Sint16 x);
void set(Uint32 x);
void set(Sint32 x);
void set(Uint64 x);
void set(Sint64 x);
void set(Real32 x);
void set(Real64 x);
void set(const Char16& x);
void set(const String& x);
void set(const CIMDateTime& x);
void set(const CIMObjectPath& x);
```

```
void set(const Array<Boolean>& x);
void set(const Array<Uint8>& x);
void set(const Array<Sint8>& x);
void set(const Array<Uint16>& x);
void set(const Array<Sint16>& x);
void set(const Array<Uint32>& x);
void set(const Array<Sint32>& x);
void set(const Array<Uint64>& x);
void set(const Array<Sint64>& x);
void set(const Array<Real32>& x);
void set(const Array<Real64>& x);
void set(const Array<Char16>& x);
void set(const Array<String>& x);
void set(const Array<CIMDateTime>& x);
void set(const Array<CIMObjectPath>& x);
```


Copyright © 2003 Hewlett-Packard Development Company, L. P.

CIMValue API

```
void get(Boolean& x) const;
void get(Uint8& x) const;
void get(Sint8& x) const;
void get(Uint16& x) const;
void get(Sint16& x) const;
void get(Uint32& x) const;
void get(Sint32& x) const;
void get(Uint64& x) const;
void get(Sint64& x) const;
void get(Real32& x) const;
void get(Real64& x) const;
void get(Char16& x) const;
void get(String& x) const;
void get(CIMDateTime& x) const;
void get(CIMObjectPath& x) const;
```

```
void get(Array<Boolean>& x) const;
void get(Array<Uint8>& x) const;
void get(Array<Sint8>& x) const;
void get(Array<Uint16>& x) const;
void get(Array<Sint16>& x) const;
void get(Array<Uint32>& x) const;
void get(Array<Sint32>& x) const;
void get(Array<Uint64>& x) const;
void get(Array<Sint64>& x) const;
void get(Array<Real32>& x) const;
void get(Array<Real64>& x) const;
void get(Array<Char16>& x) const;
void get(Array<String>& x) const;
void get(Array<CIMDateTime>& x) const;
void get(Array<CIMObjectPath>& x) const;
```


Copyright © 2003 Hewlett-Packard Development Company, L. P.

CIMValue Examples

```
CIMValue value1;
assert(value1.isNull());
value1.set(String());
assert(!value1.isNull());
value1.clear();
assert(value1.isNull());
value1.set("");
assert(!value1.isNull());

CIMValue value2 (String("String2"));
CIMValue value3 = CIMValue(String("String3"));
CIMValue value4 = value3;
CIMValue value5;
value5.assign(value3);
value3.set(String("NewString"));
```

```
CIMValue();
void clear();
Boolean isNull() const;
void set(const String& x);

CIMValue& operator=(const CIMValue& x);
void assign(const CIMValue& x);

void get(String& x) const;
```

```
String tmpString;
value3.get(tmpString);
assert(tmpString == String("NewString"));
value4.get(tmpString);
assert(tmpString == String("String3"));
value5.get(tmpString);
assert(tmpString == String("String3"));
```


Copyright © 2003 Hewlett-Packard Development Company, L. P.

CIMValue Examples

```
Boolean typeCompatible(const CIMValue& x) const;
CIMType getType() const;
```

```
CIMValue value1 (String("String1"));
CIMValue value2 (Sint32(-200));
CIMValue value3 (Sint32(200));
CIMValue value4 (Uint32(300));

assert(value1.getType() == CIMTYPE_STRING);
assert(value2.getType() == CIMTYPE_SINT32);
assert(value4.getType() == CIMTYPE_UINT32);

assert(value2.typeCompatible(value3));
assert(!value3.typeCompatible(value4));
assert(!value3.typeCompatible(value1));
```


Copyright © 2003 Hewlett-Packard Development Company, L. P.

CIMValue Examples

```
String toString() const;
```

```
CIMValue booleanValue (Boolean(true));
assert(booleanValue.toString() == "TRUE");

CIMValue uint32Value (Uint32(123));
assert(uint32Value.toString() == "123");

CIMValue dateValue (CIMDateTime("20020507170000.000000-480"));
assert(dateValue.toString() == "20020507170000.000000-480");
```


Copyright © 2003 Hewlett-Packard Development Company, L. P.

CIM Name

A CIM Name describes the name of a CIM Element (Class, Instance, Method, Property, Qualifier, or Parameter).

Its value MUST be a legal CIM element name:

- Initial identifier characters must be in set S1, where S1 = {U+005F, U+0041...U+005A, U+0061...U+007A, U+0080...U+FFEF} [This is alphabetic and underscore]
- All following characters must be in set S2 where S2 = S1 union {U+0030...U+0039} [This is alphabetic, underscore, and Arabic numerals 0 through 9.]

Name = alphabetic or '_' +
zero or more alphanumeric or '_'

Copyright © 2003 Hewlett-Packard Development Company, L. P.

CIMName API

```
CIMName();
CIMName(const String& name);
CIMName(const char* name);

CIMName& operator=(const CIMName& name);
CIMName& operator=(const String& name);
```

```
const String& getString() const;
Boolean isNull() const;
void clear();
Boolean equal(const CIMName& name) const;
static Boolean legal(const String& name);
```


Copyright © 2003 Hewlett-Packard Development Company, L. P.

CIMName Examples

```
CIMName();
CIMName& operator=(const CIMName& name);
CIMName& operator=(const String& name);
const String& getString() const;
Boolean isNull() const;
void clear();
Boolean equal(const CIMName& name) const;
static Boolean legal(const String& name);
```

```
CIMName value1;
assert(value1.isNull());

CIMName value2 = String("CIM_OperatingSystem");
CIMName value3 = "CIM_OperatingSystem";
assert(value2 == value3);
assert(!(value1 == value3));

assert(value2.getString() == String("CIM_OperatingSystem"));
assert(value2.getString() == "CIM_OperatingSystem");
assert(value2.equal(value3));
value2.clear();
assert(value2.isNull());
assert(value2 == value1);
assert(value2.equal(value1));

assert(CIMName::legal(String("CIM_OperatingSystem")));
assert(!CIMName::legal(String("127.0.0.1")));
assert(!CIMName::legal(String()));
```


Copyright © 2003 Hewlett-Packard Development Company, L. P.

CIM Qualifier Scope

Qualifier Scope indicates the kinds of Named Elements to which a given Qualifier may apply.

A Scope value may reflect any combination of “class”, “association”, “reference”, “property”, “method”, “indication”, and “parameter” (though defining a Qualifier with no Scope is not permitted). The value “any” is defined to indicate all scopes apply.

Copyright © 2003 Hewlett-Packard Development Company, L. P.

CIMScope API

```
CIMScope ();
CIMScope (const CIMScope & scope);
CIMScope & operator= (const CIMScope & scope);
Boolean hasScope (const CIMScope & scope) const;
void addScope (const CIMScope & scope);
Boolean equal (const CIMScope & scope) const;
CIMScope operator+ (const CIMScope & scope) const;
String toString () const;
static const CIMScope NONE;
static const CIMScope CLASS;
static const CIMScope ASSOCIATION;
static const CIMScope INDICATION;
static const CIMScope PROPERTY;
static const CIMScope REFERENCE;
static const CIMScope METHOD;
static const CIMScope PARAMETER;
static const CIMScope ANY;
```


Copyright © 2003 Hewlett-Packard Development Company, L. P.

CIM Qualifier Flavor

A set of behavior rules for a given Qualifier are encapsulated in its Qualifier Flavor.

Flavor	Interpretation	Default
EnableOverride	The Qualifier is overridable.	Yes
DisableOverride	The Qualifier cannot be overridden.	No
ToSubclass	The Qualifier is inherited by any subclass.	Yes
Restricted	The Qualifier applies only to the Class in which it is declared.	No
Translatable	Indicates the value of the Qualifier can be specified in multiple locales.	No

Copyright © 2003 Hewlett-Packard Development Company, L. P.

CIMFlavor API

```
CIMFlavor ();
CIMFlavor (const CIMFlavor & flavor);
CIMFlavor & operator= (const CIMFlavor & flavor);
void addFlavor (const CIMFlavor & flavor);
void removeFlavor (const CIMFlavor & flavor);
Boolean hasFlavor (const CIMFlavor & flavor) const;
Boolean equal (const CIMFlavor & flavor) const;
CIMFlavor operator+ (const CIMFlavor & flavor) const;
String toString () const;
static const CIMFlavor NONE;
static const CIMFlavor OVERRIDABLE;
static const CIMFlavor ENABLEOVERRIDE;
static const CIMFlavor DISABLEOVERRIDE;
static const CIMFlavor TOSUBCLASS;
static const CIMFlavor RESTRICTED;
static const CIMFlavor TOINSTANCE;
static const CIMFlavor TRANSLATABLE;
static const CIMFlavor DEFAULTS;
static const CIMFlavor TOSUBELEMENTS;
```


Copyright © 2003 Hewlett-Packard Development Company, L. P.

CIM Qualifier

A Qualifier is used to characterize a CIM Element (Class, Instance, Method, Property, or Parameter). For example, a Qualifier may define characteristics of a Property or a key of a Class.)

Qualifiers provide a mechanism that makes the meta schema extensible in a limited and controlled fashion. It is possible to add new types of Qualifiers by the introduction of a new Qualifier name, thereby providing new types of meta data to processes that manage and manipulate classes, properties and other elements of the meta schema.

Qualifier = Name + Type + Value + Flavor

Qualifier Declaration = Name + Type + Scope + Flavor + Default Value

Copyright © 2003 Hewlett-Packard Development Company, L. P.

CIM Qualifier Declaration – MOF

Qualifier Declaration = Name + Type + Scope + Flavor + Default Value

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

Qualifier Key : boolean = false,
    Scope(property, reference),
    Flavor(DisableOverride);

Qualifier Units : string = null,
    Scope(property, method, parameter),
    Flavor(Translatable);

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


Copyright © 2003 Hewlett-Packard Development Company, L. P.

CIMQualifierDecl API

```
CIMQualifierDecl();
CIMQualifierDecl(const CIMQualifierDecl& x);
CIMQualifierDecl(
        const CIMName& name,
        const CIMValue& value,
        const CIMScope & scope,
        const CIMFlavor & flavor = CIMFlavor (CIMFlavor::DEFAULTS),
        Uint32 arraySize = 0);
 ~CIMQualifierDecl();
CIMQualifierDecl& operator=(const CIMQualifierDecl& x);
const CIMName& getName() const;
void setName(const CIMName& name);
CIMType getType() const;
Boolean isArray() const;
const CIMValue& getValue() const;
void setValue(const CIMValue& value);
const CIMScope & getScope() const;
const CIMFlavor & getFlavor() const;
Uint32 getArraySize() const;
Boolean isUninitialized() const
Boolean identical(const CIMConstQualifierDecl& x) const;
CIMQualifierDecl clone() const;
```


Copyright © 2003 Hewlett-Packard Development Company, L. P.

CIM Qualifier – MOF

Qualifier = Name + Type + Value + Flavor

A Qualifier is specified in MOF with its value following the Name in parentheses. A boolean Qualifier specified without a value is implicitly set to “true”.

The Type and Flavor are omitted from MOF Qualifier specifications as they are assumed to be the same as the Qualifier Declaration.

```
[Version ("1.2.0"), Description ("Sample class definition") ]

[Key, Description ("Type of vehicle") ]

[IN, Description ("Desired speed") ]
```


Copyright © 2003 Hewlett-Packard Development Company, L. P.

CIMQualifier API

CIMQualifier Constructors

```
CIMQualifier();
CIMQualifier(const CIMQualifier& x);
CIMQualifier(
        const CIMName& name,
        const CIMValue& value,
        const CIMFlavor & flavor = CIMFlavor (CIMFlavor::NONE),
        Boolean propagated = false);
CIMQualifier& operator=(const CIMQualifier& x);
CIMQualifier clone() const;
```

CIMQualifier Destructor

```
~CIMQualifier();
```


Copyright © 2003 Hewlett-Packard Development Company, L. P.

CIMQualifier API

```
const CIMName& getName() const;
void setName(const CIMName& name);

CIMType getType() const;
Boolean isArray() const;

const CIMValue& getValue() const;
void setValue(const CIMValue& value);

void setFlavor(const CIMFlavor & flavor);
void unsetFlavor(const CIMFlavor & flavor);
const CIMFlavor & getFlavor() const;
const Uint32 getPropagated() const;
void setPropagated(Boolean propagated);
Boolean isUninitialized() const;
Boolean identical(const CIMConstQualifier& x) const;
```


Copyright © 2003 Hewlett-Packard Development Company, L. P.

CIMQualifier Example

```
CIMQualifier();
CIMQualifier(const CIMQualifier& x);
CIMQualifier(
        const CIMName& name,
        const CIMValue& value,
        const CIMFlavor & flavor = CIMFlavor (CIMFlavor::NONE),
        Boolean propagated = false);
const CIMName& getName() const;
CIMType getType() const;
const CIMFlavor & getFlavor() const;
const Uint32 getPropagated() const;
Boolean isUninitialized() const;
```

```
CIMQualifier qValue1;
assert(qValue1.isUninitialized());

CIMQualifier qValue2(CIMName("Abstract"), CIMValue(Boolean(false)));
assert(qValue2.getName() == String("Abstract"));
assert(qValue2.getType() == CIMTYPE_BOOLEAN);
assert(qValue2.getFlavor().hasFlavor(CIMFlavor::NONE));
assert(!qValue2.getPropagated());
```


Copyright © 2003 Hewlett-Packard Development Company, L. P.

CIMQualifier Example

```
CIMQualifier(const CIMQualifier& x);
CIMQualifier& operator=(const CIMQualifier& x);
CIMQualifier clone() const;

const CIMName& getName() const;
void setName(const CIMName& name);
```

```
CIMQualifier qValue1(CIMName("Abstract"), CIMValue(Boolean(false)));
CIMQualifier qValue2 = qValue1;
CIMQualifier qValue3(qValue1);
CIMQualifier qValue4 = qValue1.clone();

qValue1.setName(String("NewName"));
assert(qValue1.getName() == String("NewName"));
assert(qValue2.getName() == String("NewName"));
assert(qValue3.getName() == String("NewName"));
assert(qValue4.getName() == String("Abstract"));
```


Copyright © 2003 Hewlett-Packard Development Company, L. P.

CIM Property

A CIM Property is used to represent a data member of a class or instance.

**Property = Name + Type + Value +
zero or more Qualifiers**

Copyright © 2003 Hewlett-Packard Development Company, L. P.

CIM Property – MOF

Property = Name + Type + Value + zero or more Qualifiers

A property specified in a class definition without a (default) value is given a default value of “null”.

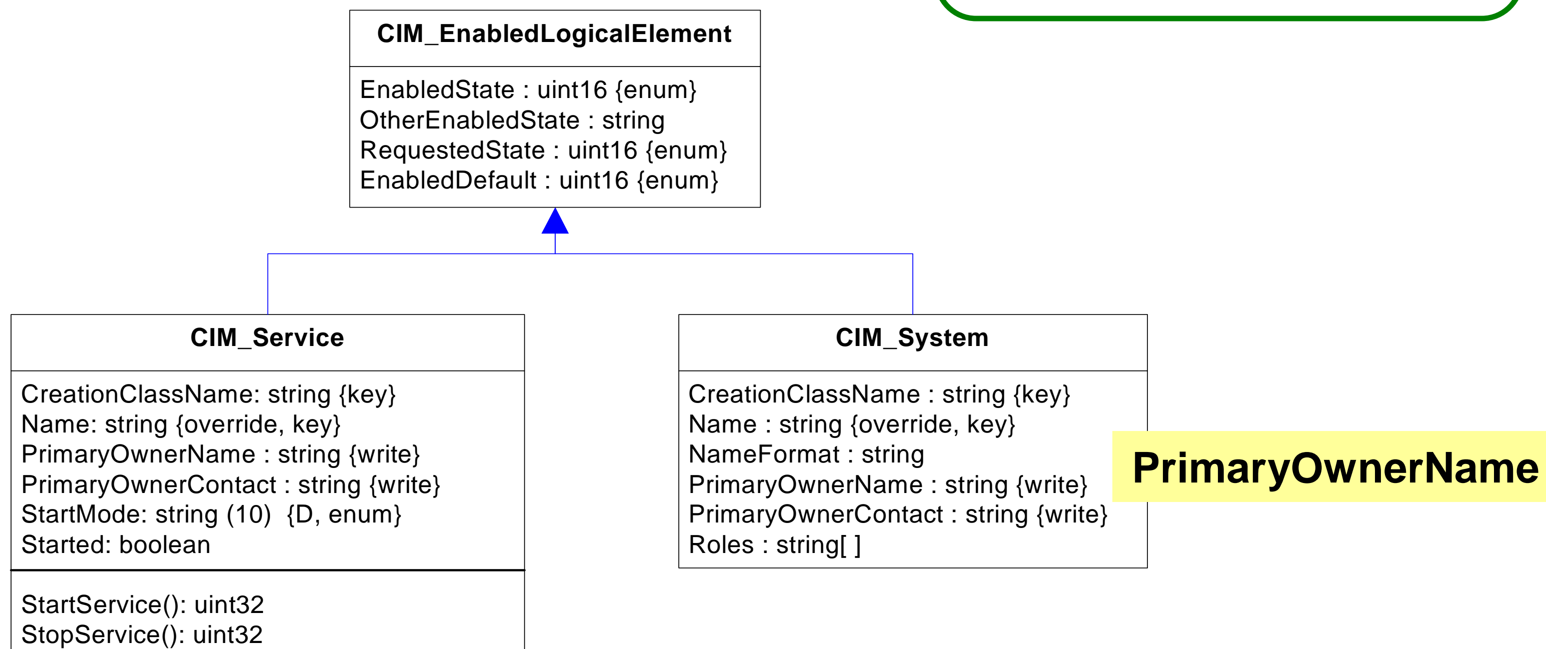
```
[Key, Description ("Type of vehicle") ]
string Model;

[Units ("km/h") ]
uint8 CurrentSpeed = 0;
```


Copyright © 2003 Hewlett-Packard Development Company, L. P.

CIMProperty

A **Property** is the definition of a value that describes a characteristic of the instances of a class.



Copyright © 2003 Hewlett-Packard Development Company, L. P.

CIMProperty API

```
CIMProperty();
CIMProperty(const CIMProperty& x);
 CIMProperty(
        const CIMName& name,
        const CIMValue& value,
        Uint32 arraySize = 0,
        const CIMName& referenceClassName = CIMName(),
        const CIMName& classOrigin = CIMName(),
        Boolean propagated = false);
CIMProperty& operator=(const CIMProperty& x);
CIMProperty clone() const;
```

CIMProperty Constructors

```
~CIMProperty();
```

CIMProperty Destructor

```
Boolean isUninitialized() const;
const CIMName& getName() const;
void setName(const CIMName& name);
Boolean identical(const CIMConstProperty& x) const;
```


Copyright © 2003 Hewlett-Packard Development Company, L. P.

CIMProperty API

```
const CIMValue& getValue() const;
CIMType getType() const;
void setValue(const CIMValue& value);
Boolean isArray() const;
Uint32 getArraySize() const;
```

```
const CIMName& getReferenceClassName() const;
const CIMName& getClassOrigin() const;
void setClassOrigin(const CIMName& classOrigin);
Boolean getPropagated() const;
void setPropagated(Boolean propagated);
```

```
CIMProperty& addQualifier(const CIMQualifier& x);
Uint32 findQualifier(const CIMName& name) const;
CIMQualifier getQualifier(Uint32 index);
CIMConstQualifier getQualifier(Uint32 index) const;
void removeQualifier(Uint32 index);
Uint32 getQualifierCount() const;
```


Copyright © 2003 Hewlett-Packard Development Company, L. P.

CIMProperty Examples

```
CIMProperty();
CIMProperty(
        const CIMName& name,
        const CIMValue& value,
        Uint32 arraySize = 0,
        const CIMName& referenceClassName = CIMName(),
        const CIMName& classOrigin = CIMName(),
        Boolean propagated = false);
Boolean isUninitialized() const; ;
const CIMName& getName() const;
```

```
CIMProperty testProperty1;
 assert(testProperty1.isUninitialized());

 CIMProperty testProperty2(CIMName("P2"), String("String2"));
 assert(!testProperty2.isUninitialized());
 assert(testProperty2.getName().equal(CIMName("P2")));

 testProperty1 = testProperty2;
 assert(!testProperty1.isUninitialized());
 assert(testProperty1.getName().equal(CIMName("P2")));
```


Copyright © 2003 Hewlett-Packard Development Company, L. P.

CIMProperty Examples

```
CIMProperty& operator=(const CIMProperty& x);
CIMProperty clone() const;
Boolean identical(const CIMConstProperty& x) const;
const CIMValue& getValue() const;
void setValue(const CIMValue& value);
```

```
CIMProperty testProperty3(CIMName("P3"), String("String3"));

CIMProperty testProperty1 = testProperty3;
CIMProperty testCloneProperty2 = testProperty3.clone();
assert(testProperty1.identical(testProperty3));
assert(testCloneProperty2.identical(testProperty3));

testProperty3.setValue(String("NewValue"));
assert(testProperty3.getValue().equal(String("NewValue")));
assert(testCloneProperty2.getValue().equal(String("String3")));
assert(testProperty1.getValue().equal(String("NewValue")));
```


Copyright © 2003 Hewlett-Packard Development Company, L. P.

CIMProperty Examples

```
CIMType getType() const;
```

```
CIMProperty testProperty1(CIMName("P3"), String("String3"));
CIMProperty testProperty2(CIMName("P4"), CIMValue(String("String4")));
CIMProperty testProperty4(CIMName("P5"), Sint32(100));
CIMProperty testProperty5(CIMName("P6"), Uint8(100));

assert(testProperty1.getType() == CIMTYPE_STRING);
assert(testProperty2.getType() == CIMTYPE_STRING);
assert(testProperty4.getType() == CIMTYPE_SINT32);
assert(testProperty5.getType() == CIMTYPE_UINT8);
```


Copyright © 2003 Hewlett-Packard Development Company, L. P.

CIMProperty Examples

```
CIMProperty(
        const CIMName& name,
        const CIMValue& value,
        Uint32 arraySize = 0,
        const CIMName& referenceClassName = CIMName(),
        const CIMName& classOrigin = CIMName(),
        Boolean propagated = false);
Boolean isArray() const;
Uint32 getArraySize() const;
```

```
CIMProperty testNonArrayProperty1(CIMName("P1"), String("String1"), 0);
Array<String> stringArray1(3, "stringValue");
CIMProperty testArrayProperty1(CIMName("P2"), stringArray1);
CIMProperty testArrayProperty2(CIMName("P3"), stringArray1, 3);

assert(!testNonArrayProperty1.isArray());
assert(testArrayProperty1.isArray());
assert(testArrayProperty1.getArraySize() == 0);
assert(testArrayProperty2.getArraySize() == 3);
```


Copyright © 2003 Hewlett-Packard Development Company, L. P.

CIMProperty Examples

```
Uint32 findQualifier(const CIMName& name) const;
CIMQualifier getQualifier(Uint32 index);
```

```
Boolean isKey(CIMProperty property)
{
    int index;

    Boolean keyQualifierValue = false;
    if ((index = property.findQualifier(CIMName("Key"))) != PEG_NOT_FOUND)
    {
       CIMValue value = property.getQualifier(index).getValue();
       if (!value.isNull ())
       {
          value.get(keyQualifierValue);
        }
     }
    return(keyQualifierValue);
}
```


Copyright © 2003 Hewlett-Packard Development Company, L. P.

CIMProperty Examples

```
Uint32 getQualifierCount() const;
CIMProperty& addQualifier(const CIMQualifier& x);
```

```
CIMProperty testProperty1(CIMName("P1"), String("UniqueIdentifier"));
 assert(testProperty1.getQualifierCount() == 0);
 testProperty1.addQualifier(CIMQualifier(CIMName("Description"),
                                          String("Comment")));
 assert(!isKey(testProperty1));
 assert(testProperty1.getQualifierCount() == 1);
 testProperty1.addQualifier(CIMQualifier(CIMName("Key"), true));
 assert(testProperty1.getQualifierCount() == 2);
 assert(isKey(testProperty1));
```


Copyright © 2003 Hewlett-Packard Development Company, L. P.

CIMPropertyList API

CIMPropertyList Constructors

```
CIMPropertyList();
CIMPropertyList(const CIMPropertyList& x);
CIMPropertyList(const Array<CIMName>& propertyNames);
CIMPropertyList& operator=(const CIMPropertyList& x);
```

CIMPropertyList Destructor

```
~CIMPropertyList();
```

```
const CIMName& operator[](Uint32 index) const;
Array<CIMName> getPropertyNameArray() const;
```

```
void set(const Array<CIMName>& propertyNames);
void clear();
Boolean isNull() const;
Uint32 size() const;
```


Copyright © 2003 Hewlett-Packard Development Company, L. P.

CIM Method

A CIM Method is used to represent a method signature of a class or instance.

Method = Name + Return Type +
zero or more Parameters +
zero or more Qualifiers

A CIM Parameter describes a parameter to a CIM Method.

Parameter = Name + Type +
zero or more Qualifiers

Copyright © 2003 Hewlett-Packard Development Company, L. P.

CIM Parameter – MOF

**Parameter = Name + Type +
zero or more Qualifiers**

```
[IN, Description ("Desired speed") ]
uint8 NewSpeed

[OUT]
uint8 PreviousSpeed
```


Copyright © 2003 Hewlett-Packard Development Company, L. P.

CIMParameter API

```
CIMParameter(const CIMParameter& x);
CIMParameter(
        const CIMName& name,
        CIMType type,
        Boolean isArray = false,
        Uint32 arraySize = 0,
        const CIMName& referenceClassName = CIMName());
CIMParameter& operator=(const CIMParameter& x);
CIMParameter clone() const;
```

```
~CIMParameter();
```

```
Boolean isUninitialized() const;
Boolean identical(const CIMConstParameter& x) const;
```


Copyright © 2003 Hewlett-Packard Development Company, L. P.

CIMParameter API

```
const CIMName& getName() const ;
void setName(const CIMName& name);
Boolean isArray() const;
Uint32 getArraySize() const;
const CIMName& getReferenceClassName() const ;
CIMType getType() const ;
```

```
CIMParameter& addQualifier(const CIMQualifier& x);
Uint32 findQualifier(const CIMName& name) const;
CIMQualifier getQualifier(Uint32 index);
void removeQualifier (Uint32 index);
CIMConstQualifier getQualifier(Uint32 index) const;
Uint32 getQualifierCount() const;
Boolean isUninitialized() const;
Boolean identical(const CIMConstParameter& x) const;
```


Copyright © 2003 Hewlett-Packard Development Company, L. P.

CIM Method – MOF

Method = Name + Return Type +
zero or more Parameters +
zero or more Qualifiers

```
[Description ("Change the vehicle speed. Returns true if successful.") ]
boolean ChangeSpeed (
    [IN, Description ("Desired speed") ]
    uint8 NewSpeed,
    [OUT]
    uint8 PreviousSpeed);
```


Copyright © 2003 Hewlett-Packard Development Company, L. P.

CIMMethod API

```
CIMMethod();
CIMMethod(const CIMMethod& x);
CIMMethod(
        const CIMName& name,
        CIMType type,
        const CIMName& classOrigin = CIMName(),
        Boolean propagated = false);
CIMMethod& operator=(const CIMMethod& x);
CIMMethod clone() const;
```

```
~CIMMethod();
```

```
Boolean isUninitialized() const;
Boolean identical(const CIMConstMethod& x) const;
```


Copyright © 2003 Hewlett-Packard Development Company, L. P.

CIMMethod API

```
const CIMName& getName() const;
void setName(const CIMName& name);
CIMType getType() const;
void setType(CIMType type);
const CIMName& getClassOrigin() const;
void setClassOrigin(const CIMName& classOrigin);
Boolean getPropagated() const;
void setPropagated(Boolean propagated);
```

```
CIMMethod& addQualifier(const CIMQualifier& x);
Uint32 findQualifier(const CIMName& name) const;
CIMQualifier getQualifier(Uint32 index);
CIMConstQualifier getQualifier(Uint32 index) const;
void removeQualifier(Uint32 index);
Uint32 getQualifierCount() const;
CIMMethod& addParameter(const CIMParameter& x);
Uint32 findParameter(const CIMName& name) const;
CIMParameter getParameter(Uint32 index);
CIMConstParameter getParameter(Uint32 index) const;
void removeParameter (Uint32 index);
Uint32 getParameterCount() const;
```


Copyright © 2003 Hewlett-Packard Development Company, L. P.

CIM Parameter Value

A CIM Parameter Value describes an argument to a CIM Method.

Parameter Value = Name + Type + Value

The return value from a CIM Method invocation is a CIM Value.

Copyright © 2003 Hewlett-Packard Development Company, L. P.
hp invent

CIMParamValue API

```
CIMParamValue();
CIMParamValue(const CIMParamValue& x);
CIMParamValue& operator=(const CIMParamValue& x);
CIMParamValue(
        String parameterName,
        CIMValue value,
        Boolean isTyped=true);
CIMParamValue clone() const;
```

```
~CIMParamValue();
```

```
String getParameterName() const;
CIMValue getValue() const;
Boolean isTyped() const;
void setParameterName(String& parameterName);
void setValue(CIMValue& value);
void setIsTyped(Boolean isTyped);
Boolean isUninitialized() const;
```


Copyright © 2003 Hewlett-Packard Development Company, L. P.

CIM Class

A CIM Class defines a set of related Instances with the same Properties and Methods.

Class = Name + Superclass Name +
zero or more Properties +
zero or more Methods +
zero or more Qualifiers

Copyright © 2003 Hewlett-Packard Development Company, L. P.
hp invent

CIM Class – MOF

Class = Name + Superclass Name +
zero or more Properties +
zero or more Methods +
zero or more Qualifiers

```
[Version ("1.2.0"), Description ("Sample class definition") ]
class HP_SampleCar : CIM_ManagedElement {
    [Key, Description ("Type of vehicle") ]
    string Model;

    [Units ("km/h") ]
    uint8 CurrentSpeed = 0;

    [Description ("Change the vehicle speed. Returns true if successful.") ]
    boolean ChangeSpeed (
        [IN, Description ("Desired speed") ]
        uint8 NewSpeed,
        [OUT]
        uint8 PreviousSpeed);
};
```


Copyright © 2003 Hewlett-Packard Development Company, L. P.

CIMClass API

CIMClass Constructor

```
CIMClass();
CIMClass(const CIMClass& x);
CIMClass(const CIMObject& x);
CIMClass(
        const CIMName& className,
        const CIMName& superClassName = CIMName());
CIMClass& operator=(const CIMClass& x);
CIMClass clone() const;
```

CIMClass Destructor

```
~CIMClass();
```

```
Boolean isUninitialized() const;
Boolean identical(const CIMConstClass& x) const;
const CIMName& getClassName() const;
const CIMName& getSuperClassName() const;
void setSuperClassName(const CIMName& superClassName);
```


Copyright © 2003 Hewlett-Packard Development Company, L. P.

CIMClass API

```
CIMClass& addQualifier(const CIMQualifier& qualifier);
Uint32 findQualifier(const CIMName& name) const;
CIMQualifier getQualifier(Uint32 index);
CIMConstQualifier getQualifier(Uint32 index) const;
void removeQualifier(Uint32 index);
Uint32 getQualifierCount() const;
```

Class Qualifiers

```
Boolean hasKeys() const;
void getKeyNames(Array<CIMName>& keyNames) const;
const CIMObjectPath& getPath() const;
void setPath (const CIMObjectPath & path);
```

Class Keys

```
Boolean isAssociation() const;
Boolean isAbstract() const;
```


Copyright © 2003 Hewlett-Packard Development Company, L. P.

CIMClass API

```
CIMClass& addProperty(const CIMProperty& x);
Uint32 findProperty(const CIMName& name) const;
CIMProperty getProperty(Uint32 index);
CIMConstProperty getProperty(Uint32 index) const;
void removeProperty(Uint32 index);
Uint32 getPropertyCount() const;
```

Class Properties

```
CIMClass& addMethod(const CIMMethod& x);
Uint32 findMethod(const CIMName& name) const;
CIMMethod getMethod(Uint32 index);
CIMConstMethod getMethod(Uint32 index) const;
void removeMethod(Uint32 index);
Uint32 getMethodCount() const;
```

Class Methods

Copyright © 2003 Hewlett-Packard Development Company, L. P.
invent

CIMClass Examples

```
CIMClass();
CIMClass(const CIMClass& x);
CIMClass& operator=(const CIMClass& x);
Boolean isUninitialized() const;
const CIMName& getClassName() const;
const CIMName& getSuperClassName() const;
```

```
try
  {
     CIMClass testClass1;
     CIMClass testClass2(CIMName("B"));
     CIMClass testClass3(CIMName("A"), CIMName("B"));
     CIMClass testClass4(CIMName("C"), CIMName("D"));

     assert(testClass1.isUninitialized());
     assert(testClass2.getClassName().equal(CIMName("B")));
     assert(testClass2.getClassName().equal(CIMName("b")));
     assert(testClass2.getSuperClassName().equal(CIMName()));
     assert(testClass3.getSuperClassName().equal(CIMName("b")));
  }
  catch(Exception& e)
  {
     cerr << "Constructor Error: " << e.getMessage() << endl;
     exit(1);
  }
```


Copyright © 2003 Hewlett-Packard Development Company, L. P.

CIMClass Examples

```
CIMClass& operator=(const CIMClass& x);
CIMClass clone() const;
void setSuperClassName(const CIMName& superClassName);
```

```
CIMClass testClass1(CIMName("A"), CIMName("B"));
CIMClass testClassCopy2(testClass1);
CIMClass testClassCopy3 = testClass1;
CIMClass testClassClone4 = testClass1.clone();

assert(testClassCopy2.identical(testClass1));
assert(testClass1.identical(testClassCopy2));
assert(testClass1.identical(testClassCopy3));
assert(testClass1.identical(testClassClone4));

testClassClone4.setSuperClassName(CIMName("C"));
assert(testClassClone4.getSuperClassName().equal(CIMName("C")));
assert(testClass1.getSuperClassName().equal(CIMName("B")));

testClassCopy3.setSuperClassName(CIMName("D"));
assert(testClassCopy3.getSuperClassName().equal(CIMName("D")));
assert(testClass1.getSuperClassName().equal(CIMName("D")));
assert(testClassCopy2.getSuperClassName().equal(CIMName("D")));
```


Copyright © 2003 Hewlett-Packard Development Company, L. P.

CIMClass Examples

```
CIMClass& addQualifier(const CIMQualifier& qualifier);
Uint32 findQualifier(const CIMName& name) const;
CIMQualifier getQualifier(Uint32 index);
void removeQualifier(Uint32 index);
Uint32 getQualifierCount() const;
Boolean isAssociation() const;
Boolean isAbstract() const;
```

```
CIMClass testClass1(CIMName("A"), CIMName("B"));
testClass1.addQualifier(CIMQualifier(CIMName("Association"), true));

assert(testClass1.isAssociation());
assert(!testClass1.isAbstract());
assert(testClass1.findQualifier(CIMName("Association")) != PEG_NOT_FOUND);
assert(testClass1.findQualifier(CIMName("association")) != PEG_NOT_FOUND);
assert(testClass1.findQualifier(CIMName("Abstract")) == PEG_NOT_FOUND);

assert(testClass1.findQualifier(CIMName("Abstract")) == PEG_NOT_FOUND);
assert(testClass1.getQualifierCount() == 1);
testClass1.addQualifier(CIMQualifier(CIMName("Abstract"), true));
assert(testClass1.getQualifierCount() == 2);

Uint32 index = testClass1.findQualifier(CIMName("Association"));
testClass1.removeQualifier(index);
assert(testClass1.getQualifierCount() == 1);
```


Copyright © 2003 Hewlett-Packard Development Company, L. P.
invent

CIM Instance

A CIM Instance represents an instance of a Class, including values for the Properties.

Instance = Class Name +
zero or more Properties +
zero or more Qualifiers

Copyright © 2003 Hewlett-Packard Development Company, L. P.

CIM Instance – MOF

Instance = Class Name +
zero or more Properties +
zero or more Qualifiers

```
[Description ("My car")]
instance of HP_SampleCar
{
    Model = "Mini Cooper";
    CurrentSpeed = 160;
};
```


Copyright © 2003 Hewlett-Packard Development Company, L. P.

CIMInstance API

```
CIMInstance();
CIMInstance(const CIMInstance& x);
CIMInstance(const CIMObject& x);
CIMInstance(const CIMName& className);
CIMInstance& operator=(const CIMInstance& x);
CIMInstance clone() const;
```

```
~CIMInstance();
```

```
Boolean identical(const CIMConstInstance& x) const;
Boolean isUninitialized() const;
```

```
const CIMName& getClassName() const;
const CIMObjectPath& getPath() const;
void setPath (const CIMObjectPath & path);
CIMObjectPath buildPath(const CIMConstClass& cimClass) const;
```


Copyright © 2003 Hewlett-Packard Development Company, L. P.

CIMInstance API

```
CIMInstance& addProperty(const CIMProperty& x);
Uint32 findProperty(const CIMName& name) const;
CIMProperty getProperty(Uint32 index);
CIMConstProperty getProperty(Uint32 index) const;
void removeProperty(Uint32 index);
Uint32 getPropertyCount() const;
```

```
CIMInstance& addQualifier(const CIMQualifier& qualifier);
Uint32 findQualifier(const CIMName& name) const;
CIMQualifier getQualifier(Uint32 index);
CIMConstQualifier getQualifier(Uint32 index) const;
Uint32 getQualifierCount() const;
```


Copyright © 2003 Hewlett-Packard Development Company, L. P.