Technical Standard

Systems Management:  
Common Manageability Programming Interface (CMPI)

Document History:

<table>
<thead>
<tr>
<th>Version 1.3</th>
<th>September 17th 2003</th>
<th>Martin Kirk</th>
<th>All functions updated. Data structures require updating/checking. There is still missing information for return values.</th>
</tr>
</thead>
</table>
© Copyright May 2003, The Open Group

All rights reserved.

No part of this publication may be reproduced, stored in a retrieval system, or transmitted, in any form or by any means, electronic, mechanical, photocopying, recording, or otherwise, without the prior permission of the copyright owner.

This specification has not been verified for avoidance of possible third-party proprietary rights. In implementing this specification, usual procedures to ensure the respect of possible third-party intellectual property rights should be followed.

Technical Standard

**Systems Management: Common Management Programming Interface (CMPI)**

ISBN:

Document Number:

Published by The Open Group, May 2003.

Comments relating to the material contained in this document may be submitted to:

The Open Group
Apex Plaza
Forbury Road
Reading
Berkshire, RG1 1AX
United Kingdom

or by electronic mail to:

ogspecs@opengroup.org
Contents

Contents ........................................................................................................ iv

Preface ........................................................................................................ vii

Trademarks........................................................................................................ ix

Acknowledgements ................................................................................................. x

Referenced Documents ............................................................................................ xi

1 Introduction ........................................................................................................ 1
   1.1 Purpose .......................................................................................................... 1
   1.2 Intended Audience ......................................................................................... 1
   1.3 Summary of Requirements .............................................................................. 1

2 CMPI Interface General Structure ..................................................................... 2

3 API Concepts ...................................................................................................... 3
   3.1 Highlights ....................................................................................................... 3
       3.1.1 Guiding Principles ................................................................................. 3
       3.1.2 Encapsulation ......................................................................................... 3
       3.1.3 Data Transfer ......................................................................................... 3
       3.1.4 Error Indication ...................................................................................... 4
       3.1.5 Threading .............................................................................................. 5
       3.1.6 Memory Ownership .............................................................................. 5
       3.1.7 The Other Side of the API .................................................................... 5
       3.1.8 Asymmetric Properties ....................................................................... 5
   3.2 Broker Services ............................................................................................ 6
   3.3 Encapsulated Types ...................................................................................... 6
   3.4 Programming Convenience Support ............................................................ 7
       3.4.1 Macro Support ....................................................................................... 7
       3.4.2 C++ Class Support .............................................................................. 8
   3.5 Query and Indication Filtering ..................................................................... 9
       3.5.1 Indication Filtering .............................................................................. 9
       3.5.2 Query .................................................................................................... 9
       3.5.3 Normalized Select Conditions .............................................................. 10

4 Data Types ........................................................................................................ 11
   4.1 CMPI Encapsulated Data Types ................................................................. 11
   4.2 CMPI String Data ....................................................................................... 12
   4.3 CMPI Simple Data Types ........................................................................... 12
   4.4 CMPI Miscellaneous Data Types ............................................................... 14
   4.5 CMPI Types and Values ............................................................................. 14
       4.5.1 CMPIData ............................................................................................ 14
       4.5.2 CMPIType .......................................................................................... 14
<table>
<thead>
<tr>
<th>Chapter</th>
<th>Section</th>
<th>Topic</th>
</tr>
</thead>
<tbody>
<tr>
<td>4.5.3</td>
<td>CMPIValueState</td>
<td>16</td>
</tr>
<tr>
<td>4.5.4</td>
<td>CMPIValue</td>
<td>16</td>
</tr>
<tr>
<td>4.6</td>
<td>Null Value Specification</td>
<td>17</td>
</tr>
<tr>
<td>5</td>
<td>MI Function Signatures</td>
<td>18</td>
</tr>
<tr>
<td>5.1</td>
<td>MI Factory</td>
<td>19</td>
</tr>
<tr>
<td>5.1.1</td>
<td>&lt;mi-name&gt;_Create&lt;mi-type&gt;MI</td>
<td>19</td>
</tr>
<tr>
<td>5.1.2</td>
<td>CMPI_MIType_xxx</td>
<td>19</td>
</tr>
<tr>
<td>5.1.3</td>
<td>CMPIInstanceMIFT</td>
<td>19</td>
</tr>
<tr>
<td>5.1.4</td>
<td>CMPIAssociationMIFT</td>
<td>20</td>
</tr>
<tr>
<td>5.1.5</td>
<td>CMPIMethodMIFT</td>
<td>20</td>
</tr>
<tr>
<td>5.1.6</td>
<td>CMPIPropertyMIFT</td>
<td>21</td>
</tr>
<tr>
<td>5.1.7</td>
<td>CMPIIndicationMIFT</td>
<td>21</td>
</tr>
<tr>
<td>5.1.8</td>
<td>Functions</td>
<td>21</td>
</tr>
<tr>
<td>5.2</td>
<td>Instance MI Signatures</td>
<td>27</td>
</tr>
<tr>
<td>5.2.1</td>
<td>CMPIFlags</td>
<td>27</td>
</tr>
<tr>
<td>5.2.2</td>
<td>Functions</td>
<td>27</td>
</tr>
<tr>
<td>5.3</td>
<td>Association MI Signatures</td>
<td>35</td>
</tr>
<tr>
<td>5.4</td>
<td>Property MI Signatures</td>
<td>43</td>
</tr>
<tr>
<td>5.5</td>
<td>Method MI Signatures</td>
<td>46</td>
</tr>
<tr>
<td>5.6</td>
<td>Indication MI Signatures</td>
<td>48</td>
</tr>
<tr>
<td>5.7</td>
<td>CMPI Return Codes</td>
<td>53</td>
</tr>
<tr>
<td>5.8</td>
<td>CMPI Result Data Support</td>
<td>53</td>
</tr>
<tr>
<td>5.9</td>
<td>Context Data Support</td>
<td>59</td>
</tr>
<tr>
<td>6</td>
<td>Data Type Manipulation Functions</td>
<td>64</td>
</tr>
<tr>
<td>6.1</td>
<td>Miscellaneous Services</td>
<td>65</td>
</tr>
<tr>
<td>6.2</td>
<td>CMPIString Support</td>
<td>96</td>
</tr>
<tr>
<td>6.3</td>
<td>CMPIArray Support</td>
<td>99</td>
</tr>
<tr>
<td>6.4</td>
<td>CMPIEnumeration Support</td>
<td>105</td>
</tr>
<tr>
<td>6.5</td>
<td>CMPIInstance Support</td>
<td>109</td>
</tr>
<tr>
<td>6.6</td>
<td>CMPIObjectPath Support</td>
<td>116</td>
</tr>
<tr>
<td>6.7</td>
<td>CMPIArgs Support</td>
<td>129</td>
</tr>
<tr>
<td>6.8</td>
<td>CMPIDateTime Support</td>
<td>135</td>
</tr>
<tr>
<td>6.9</td>
<td>CMPISelectExp Support</td>
<td>142</td>
</tr>
<tr>
<td>6.10</td>
<td>CMPISelectCond Support</td>
<td>148</td>
</tr>
<tr>
<td>6.11</td>
<td>CMPISubCond Support</td>
<td>151</td>
</tr>
<tr>
<td>6.12</td>
<td>CMPIPredicate Support</td>
<td>155</td>
</tr>
<tr>
<td>7</td>
<td>Qualifier Support</td>
<td>158</td>
</tr>
<tr>
<td>8</td>
<td>Schema Support</td>
<td>163</td>
</tr>
<tr>
<td>9</td>
<td>MB Services</td>
<td>164</td>
</tr>
<tr>
<td>9.1</td>
<td>Minimal Services</td>
<td>164</td>
</tr>
<tr>
<td>9.2</td>
<td>MB Classification and Optional Feature Support</td>
<td>164</td>
</tr>
<tr>
<td>9.3</td>
<td>Class 0 Services</td>
<td>165</td>
</tr>
<tr>
<td>9.4</td>
<td>Class 1 Services</td>
<td>167</td>
</tr>
<tr>
<td>9.5</td>
<td>Class 2 Services</td>
<td>170</td>
</tr>
<tr>
<td>9.6</td>
<td>Accessing MB Services</td>
<td>189</td>
</tr>
<tr>
<td>9.6.1</td>
<td>CMPIBrokerFT</td>
<td>189</td>
</tr>
</tbody>
</table>
A  Header Files ................................................................. 191
  A.1  Data Types (<cmpidt.h>) ........................................ 191
  A.2  Function Tables (<cmpift.h>) .................................... 197

B  MI Convenience Support .............................................. 206
  B.1  C++ Convenience Classes ....................................... 206
  B.2  Convenience Macros .............................................. 216

C  Sample Instance MI.................................................... 222

Glossary ........................................................................... 227

Index ................................................................................. 228
Preface

The Open Group

The Open Group, a vendor and technology-neutral consortium, has a vision of Boundaryless Information Flow achieved through global interoperability in a secure, reliable, and timely manner. The Open Group mission is to drive the creation of Boundaryless Information Flow by:

- Working with customers to capture, understand, and address current and emerging requirements, establish policies, and share best practices
- Working with suppliers, consortia, and standards bodies to develop consensus and facilitate interoperability, to evolve and integrate open specifications and open source technologies
- Offering a comprehensive set of services to enhance the operational efficiency of consortia
- Developing and operating the industry's premier certification service and encouraging procurement of certified products

The Open Group provides opportunities to exchange information and shape the future of IT. The Open Group members include some of the largest and most influential organizations in the world. The flexible structure of The Open Group membership allows for almost any organization, no matter what their size, to join and have a voice in shaping the future of the IT world.

More information is available at www.opengroup.org.

The Open Group has over 15 years' experience in developing and operating certification programs and has extensive experience developing and facilitating industry adoption of test suites used to validate conformance to an open standard or specification.

More information is available at www.opengroup.org/testing.

The Open Group publishes a wide range of technical documentation, the main part of which is focused on development of Technical and Product Standards and Guides, but which also includes white papers, technical studies, branding and testing documentation, and business titles. Full details and a catalog are available at www.opengroup.org/pubs.

As with all live documents, Technical Standards and Specifications require revision to align with new developments and associated international standards. To distinguish between revised specifications which are fully backwards-compatible and those which are not:

- A new Version indicates there is no change to the definitive information contained in the previous publication of that title, but additions/extensions are included. As such, it replaces the previous publication.
- A new Issue indicates there is substantive change to the definitive information contained in the previous publication of that title, and there may also be
additions/extensions. As such, both previous and new documents are maintained as current publications.

Readers should note that Corrigenda may apply to any publication. Corrigenda information is published at www.opengroup.org/corrigenda.

This Document

This document is a Technical Standard (see above).

Typographical Conventions

The following typographical conventions are used throughout this document:

- **Bold** font is used in text for options to commands, filenames, keywords, type names, data structures, and their members.

- **Italic** strings are used for emphasis or to identify the first instance of a word requiring definition. Italics in text also denote:
  - Command operands, command option-arguments, or variable names; for example, substitutable argument prototypes
  - Environment variables, which are also shown in capitals
  - Utility names
  - External variables, such as `errno`
  - Functions; these are shown as follows: `name()`. Names without parentheses are C external variables, C function family names, utility names, command operands, or command option-arguments.

- Normal font is used for the names of constants and literals.

- The notation `<file.h>` indicates a header file.

- Names surrounded by braces – for example, `{ARG_MAX}` – represent symbolic limits or configuration values which may be declared in appropriate headers by means of the C `#define` construct.

- The notation `[ABCD]` is used to identify a return value ABCD, including if this is an error value.

- Syntax, code examples, and user input in interactive examples are shown in fixed width font. Brackets shown in this font, `[]`, are part of the syntax and do not indicate optional items. In syntax the `|` symbol is used to separate alternatives, and ellipses (…) are used to show that additional arguments are optional.
Trademarks

Boundaryless Information Flow™ is a trademark and UNIX® and The Open Group® are registered trademarks of The Open Group in the United States and other countries.

The Open Group acknowledges that there may be other brand, company, and product names used in this document that may be covered by trademark protection and advises the reader to verify them independently.
Acknowledgements

The Open Group gratefully acknowledges the contribution of the following people in the development of this Technical Standard:

Guru Bhat Sun Microsystems, Inc.
Mike Day IBM Corporation
Andreas Maier IBM Corporation
Viktor Mihajlovski IBM Corporation, Linux Technology Centre
Rick Ratta Sun Microsystems, Inc.
Adrian Schuur IBM Corporation, Linux Technology Centre
Doug Wood IBM Corporation
Referenced Documents

The following documents are referenced in this Technical Standard:

1 Introduction

1.1 Purpose

The purpose of this document is to describe a standard C API to be supported by Management Brokers (MB, aka CIM Object Managers) for interacting with Management Instrumentation (MI, aka CIM Providers).

1.2 Intended Audience

This document is not a programming guide, although it is recognized that one is needed eventually. The intended audience should be knowledgeable in CIM terms and concepts in general, and have knowledge of one or more CIMOM implementations.

1.3 Summary of Requirements

In a number of related discussions, requirements and suggestions have been collected and form the basis for this specification. The following list summarizes these requirements:

1. Reduce the complexity of writing MI.
2. Complete encapsulation and independence of data structures used by MBs.
3. No (MB implementation-specific) library needed for writing MIs; a C header file is all that is needed.
4. Support for interface opaque mapping strings to be passed by the MB down to the MI.
5. Support for remote MI using Pegasus-style response handling and progress indications.
6. Place no requirement on the instrumentation to maintain state between calls. This is to support instrumentation implemented as shell scripts, and generic providers that get their context from the registration schema.
7. Be thread-safe and reentrant. The MB may call into the MI on multiple threads simultaneously.
8. Support an arbitrary number of providers implemented in a single library.
9. Allow instrumentation libraries to be secure.
10. Allow one instrumentation library to easily load another.

---

1 The terms “Management Broker” and “Management Instrumentation” are used instead of CIMOM and providers because CMPI is not necessarily restricted to CIMOM environments.
CMPI Interface General Structure

The environment provided by Management Brokers (MBs) in which Management Instrumentation (MI) executes has to provide the following support:

1. Access to data type manipulation functions.
   This support is made accessible to the MI via vectors of entry points. MBs must provide these vectors and provide MB-specific implementation for this support. All defined functions must be implemented.

2. Access to MB services.
   This support is made accessible to MIs via a vector of entry points. MBs must provide this vector and provide MB-specific implementation. MBs may opt to implement a subset of these services according to its classification (see Chapter 9).

MIs must provide the following:

1. Access to the MI function entry points.
   Access is provided by passing a vector of entry points to the MB at initial invocation time. In addition, this vector will have information about the groups of MI functions supported (instance, associations, etc.) and the level of API used.

Layout of the vectors and function signatures will be described by a set of C header files. A set of C macros will be provided to hide the vector indirection. For C++ environments, methods will be defined that enable C++-style processing of CMPI entities.
3 API Concepts

In this chapter the principles of CMPI are explained. Code fragments must be seen in the context of the text describing them, and are not complete or are generic descriptions. The notation “/...” indicates pieces that are left out. Appendix A is used to describe the complete set of APIs and data structures and will be guaranteed to compile without failure (eventually).

3.1 Highlights

3.1.1 Guiding Principles

There are four basic thoughts that have governed the design of this interface:

1. Existing Management Brokers (MBs) must be able to use CMPI without the need for extensive rework. This resulted in the concept of encapsulation.

2. The external API signatures should be structured such that the number of them is containable and where possible have a high degree of regularity. Ease-of-use is achieved by defining a set of convenience macros and C++ wrappers.

3. Since most functions result in transformations to the respective MB APIs, CMPI should not introduce new structures containing function parameter data; instead, data should be passed directly as function parameters.

4. CMPI strives to make Management Instrumentation (MI) programming simpler and will eliminate, as far as possible, intermediate objects, in particular CIMValue and CIMProperty.

3.1.2 Encapsulation

Any CMPI data type that has MB implementation-dependent details or contains CMPI data areas is called an “encapsulated data type”. An encapsulated data type instance has two parts: a pointer to its MB-dependent implementation and a pointer to the function table giving access to all operations supported by this type. The operation repertoire of encapsulated data types is a major portion of the CMPI interface. This repertoire includes, at a minimum, lifecycle operations (release and clone).

3.1.3 Data Transfer

Most of the API functions deal with transferring property values to and from MBs. Generically there are at least two components needed to define data items: the value itself and its type. Since CIM also supports the notion of NULL values, meaning a property has no value set, a third component is used to define the state of a value indicating whether or not a value has been set.

Values are represented as a union of all types supported by CMPI (CMPIValue). Types are represented as a typedef (CMPIType). Value states are also represented as a typedef (CMPIValueState).
Simple Data Items

For property data transfer calls into CMPI two parameters are used: the address of the value (CMPIData *) and its type (CMPIType). If the type does not conform to the schema-defined type, the MB will attempt to cast to the schema-defined type where possible.

Assume a schema-defined property contains a 32-bit signed integer. If the MI wants to set a value, it will pass the address of a CMPIData item containing the value and a CMPIType of CMPI_sint32. If the MI wants to set a NULL value, it will pass a NULL CMPIData address and a CMPIType of CMPI_sint32.

Property data transfers from CMPI functions to the MI return three components: the value, its state, and its type as defined by the schema. All three components are bundled into one structure (CMPIData).

Assume a schema-defined property contains a 32-bit signed integer. If the value is set, the MB uses the CMPIData structure to return the value and the type is set to CMPI_sint32. If the value is not set, the MB uses the CMPIData structure to return the type set to CMPI_sint32, with its state set to CMPI_nullValue. In this case, the value union contains no meaningful data.

Array Data Items

Because NULL values must be supported, CMPI cannot directly support C-style arrays. Instead, arrays must be represented as if they are arrays of CMPIData structures. CMPI arrays are encapsulated data types (CMPIData).

Before transferring arrays into CMPI the MI first has to create a CMPIData of a CMPIType-defined array type (any of CMPI_xxxA) and fill it with data using the setElementAt() function provided by the CMPIData encapsulated object using the simple data transfer scheme described above.

For property data transfer calls of completed arrays into CMPI the same scheme is used as for transferring simple items; the two parameters are the address of the CMPIData union containing the pointer to the CMPIData and CMPI_xxxA as a type.

Property operations returning arrays to an MI use the scheme as used for simple items. If the array property is set, then CMPIData holds a pointer to a CMPIData encapsulated data type with any of the CMPI_xxxA types. Otherwise, a CMPI_nullValue state is returned along with the schema-defined type (any of CMPI_xxxA).

Access to the individual elements of the array is done using the getelementAt() function provided by the CMPIData encapsulated data type following the simple data item transfer scheme.

3.1.4 Error Indication

All MI and CMPI functions return a CMPIStatus structure, either as a return value or via an output parameter, which contains a CMPIrc return code and an optional message in CMPIString format. The return codes are compatible with DMTF’s CIM error codes. Special codes for additional error indications from CMPI data manipulation routines will be added if needed.

If the CMPIStatus output parameter equals NULL, no status structure will be returned.
3.1.5 Threading

CMPI imposes no limits on threading concepts and is itself thread-safe. Whether MIs themselves are thread-safe and can maintain data integrity is a different story. This depends entirely on the nature of the MI and the resources it controls or represents and on the semantics, or lack thereof, imposed by CIM. MIs must ensure thread-safety, if needed, by using serialization techniques internally.

Support for security context will be provided. If an MI wishes to use threads in which CMPI functions are invoked, the thread has to be introduced to CMPI prior to invoking any CMPI functions using the `prepareAttachThread()`, `attachThread()`, and `detachThread()` broker functions. The `prepareAttachThread()` function requires the original `CMPIContext` object to be passed as a parameter and produces a new `CMPIContext` object to be used by `attachThread()`.

3.1.6 Memory Ownership

Due to its encapsulating nature, all complex data structures are created by or via CMPI functions and should therefore be released by or via CMPI. This enables the insulation of MIs from specific memory management techniques used by the MB.

CMPI performs automatic release of all its encapsulated objects used and/or created during an MI function invocation cycle, except for those structures that have been copied explicitly using the `clone()` function by the MI.

Structures that have been copied must eventually be released by the MI explicitly using the `release()` function. This function can be used by long-running indication threads or during invocation cycles to indicate that particular encapsulated objects (not necessarily cloned) are not needed anymore.

3.1.7 The Other Side of the API

The other side of the API is the implementation of CMPI. It mostly is an adaptation layer that transforms CMPI semantics to the semantics employed by the respective MB. In general, this is a straightforward process; however, there are cases where this is not so easy. Management applications that wish to call CMPI-style MI without using an MB must provide support for all CMPI encapsulated data types. There is an opportunity for a reference implementation for this purpose, but that is outside the scope of this document.

3.1.8 Asymmetric Properties

The API concept has some asymmetric properties due to the fact that MBs have very diverse implementation principles. For instance, Pegasus is enabled for asynchronous delivery of partial results from the MI towards the MB, whereas other MBs expect an enumeration of some kind to be returned. The lowest common denominator chosen is to use Pegasus’ `ReturnHandler` concept. MBs expecting enumeration returns can easily adapt the `returnData` mechanism to create the required enumeration. On the other hand, an MI invoking MB services always gets back a `CMPIEnumeration`. This asymmetry is visible, but is not considered to be a problem.
3.2 Broker Services

The MB offers services to invoke complex operations similar to operations issued by a CIMClient. In addition, the MB gives access to factory services for creating encapsulated types.

A CMPIBroker data type is defined to provide access to the broker services. The format of the CMPIBroker data type is as follows:

```c
typedef struct _CMPIBroker {
    CMPIBrokerFT* bft;
    CMPIBrokerEncFT* eft;
} CMPIBroker;
```

The CMPIBroker object is passed as a pointer to an MI made available to the MI factory (<mi-name>_Create<mi-type>MI) at initialization time. The MI will cache this pointer across calls and use it where appropriate.

3.3 Encapsulated Types

As mentioned before, encapsulated data types have a per-type function table. For every type a mandatory repertoire of functions is defined and must be implemented by respective MBs. The sum of all routines accessible via the function tables constitutes a major part of the CMPI interface. This concept enables MI to work transparently (and simultaneously if needed) to work with data type instances coming from different MBs using different heap management concepts.

Generically, an encapsulated type has the following format:

```c
typedef struct _<E-Type> {
    void *hdl;
    <E-Type>FT ft;
} <E-Type>;
```

The following is a partial example of the type and function table for CMPIInstance:

```c
typedef struct _CMPIInstance {
    void *hdl;
    CMPI_Instance_Ftab;
} CMPIInstance;

typedef struct _CMPI_Instance_Ftab {
    CMPIStatus    (*release)(CMPIInstance*);
    CMPIInstance *(*clone)(CMPIInstance*,CMPIStatus*);
    CMPIStatus    (*setProperty)(CMPIInstance*,char*,CMPIValue*,
        CMPIType);
    CMPIData     (*getProperty)(CMPIInstance*,char*,CMPIStatus*);
} CMPI_Instance_Ftab;
```

The release() and clone() functions are common across and mandatory for every encapsulated object. The setProperty() and getProperty() functions are typical examples for setting and getting the values in/from encapsulated objects. The setProperty()-like operations always have a CMPIValue and CMPIType pair as parameters. The getProperty()-like operations always return a CMPIData structure containing the actual value and its type and
typedef struct _CMPIData {
    CMPIType type;
    CMPIValueState state;
    CMPIValue value;
} CMPIData;

Arrays are returned as CMPIArray encapsulated types. This type has functions to get data pertaining to arrays like type, size, and individual elements.

The following example shows some of the actual CMPI functions related to the CMPIInstance data type:

```c
void aUselessRoutine(CMPIInstance *ci) {
    CMPInstance *cic;
    CMPIArray *ar;
    CMPICount count;
    CMPIStatus rc;
    CMPIValue val;
    int intVal;

    val.sint32 = 25;
    cic = ci->ft->clone(ci, &rc);
    rc =
    cic->ft->getProperty(cic, "prop1", &value, CMPI_sint32);
    intVal =
    cic->ft->getProperty(cic, "prop1", &rc).value.sint32;
    ar = cic->ft->getProperty(cic, "propA", &rc).value.array;
    count = cic->ft->getSize(ar, &rc);
    for (int i = 0; i < count; i++)
        intVal =
        ar->ft->getElementAt(ar, i, &rc).value.sint32;
}
```

### 3.4 Programming Convenience Support

Notice that the mapping macros and classes described here are programming convenience supportive elements. Since they are resolved by the compiler, they do not affect the interoperability between MBs and MIs. Nevertheless they are important and should be considered part of the specification.

#### 3.4.1 Macro Support

Using direct, unaided invocation can become tedious. C macros can be used to make this easier. CMPI data manipulation macro names start with CM.

```c
#define CMRelease(o)         ((o) -> ft-> release((o)))
#define CMClone(o,rc)        ((o) -> ft-> clone((o),(rc)))

#define CMSSetProperty(o,v,t) \
    ((o) -> ft->setProperty((o), (CMPIValue*) (v), (t)))
#define CMGetProperty(o,rc)  ((o) -> ft-> get((o), (rc)))
```

//...
The example above recoded using the macro support:

```c
void aUselessRoutine(CMPIInstance *ci)
{
    CMPIInstance *cic;
    CMPIArray* ar;
    CMPICount count;
    CMPIStatus rc;
    int IntVal=25;

cic=CMClone(ci,&rc);
rc=CMSetProperty(cic,"prop1",&val,CMPI_sint32);
intVal=CMGetProperty(cic,"prop1",&rc).value.sint32;

ar=CMGetProperty(cic,"propA",&rc).value.array;
count=CMGetArraySize(ar,&rc);
for (int i=0; i<count; i++)
    intVal=CMGetArrayElementAt(ar,i,&rc).value.sint32;
}
```

A complete listing can be found in Section B.2.

### 3.4.2 C++ Class Support

In a C++ environment, C++ classes can be used to access and manipulate encapsulated types. There are two possible ways to do this:

1. Inline method declarations within the encapsulated types
2. Wrapping of the encapsulated types

Both methods have their advantages and are independent of the C ABI interface. Wrappers will introduce a proper class structure that is extendible as first-class C++ objects. For that reason the wrapping model will be used. Classes use their own naming structure, starting with “Cmpi”. The following is an example; the complete class definitions can be found in Section B.1.

```c
Class CmpiInstance {
private:
    CMPIInstance *enc;
    CmpiInstance();
    CmpiInstance(CMPIInstance*);
public:
    CmpiInstance(const CmpiInstance&);
    ~CmpiInstance();
    CmpiInstance(const CmpiObjectPath&);
    CmpiData getProperty(char*);
    void getProperty(char*,CMPIsint8&);
    void getProperty(char*,CMPIsint8&);
    //...
    void getProperty(char*,CmpiObjectPath&);
    void setProperty(char*,CmpiData&);
    void setProperty(char*,CmpiObjectPath&);
    //...
};
```

The examples above recoded using the C++ methods and macro support:

```c
void xx::aUselessRoutine(CmpiInstance &ci, ..) {
```
CMPIType type;
CMPICount count;
CmpiArray ar;
int val=25;

CmpiInstance& cic(ci);
cic.setProperty("prop1",CmpiValue(val));
cic.getProperty("prop1",val);
cic.getProperty("prop1",val);

cic.getProperty("propA",ar);
count=ar.getSize();
for (int i=0; i<count; i++)
    ar.getElementAt(i,val);
}

The C++ support also introduces a helper class called CmpiValue. It is used to exploit
method overloading and the facilities of C++ not available in C. It creates the
CMPIType/CmpiValue pair.

3.5 Query and Indication Filtering

The filtering concept for indication support is based on experiences with Sun WBEM
Services and The Open Group OpenCimom implementations. In essence the filter is
representing the complete select expression as a combination of a list defining the projection
and a parse tree defining the select condition. A select expression is encapsulated by
CMPISelectExp. Similarly, for query support, a string containing a select condition is
passed to the execQuery() function.

3.5.1 Indication Filtering

For indication filtering there are two distinct reasons for having access to the select
condition. The most obvious one is for filtering indication candidates. This is probably best
done by the MB itself, and presumably the filter structure is set up for effective filter
processing.

The other reason is to enable an MI supporting indications to inspect and understand the
filter and instrument itself for effective monitoring of the resources it controls. For example,
an MI that monitors the availability of a critical service or daemon in a system could
arbitrarily poll all services in a system regularly, or, intelligently use instrumentation in a
system that natively monitors the availability of services by specifying which service(s)
should be monitored.

3.5.2 Query

Effective query processing has similar needs: the query process might transform the query
into a native format known by the domain this MI function represents. The execQuery()
function has one more difficulty: it has to parse and query the string which is not necessarily
WQL. In case a WQL-style language is used, a factory function is offered to transform a
query string into a CMPISelectExp.
3.5.3 Normalized Select Conditions

In order to inspect a select expression, MIs can request the expression to be transformed into either a conjunction of disjunctions (CoD), or a disjunction of conjunctions (DoC) canonical form, encapsulated as `CMPISelectCond`. In both forms all parenthesis and negations are resolved. As part of this function the projection list can also be requested.

The `CMPISelectCond` is a list of `CMPISubCond` objects. A `CMPISubCond` object is a list of `CMPIPredicates`. Depending on the requests (DoC or CoD), the resulting predicates of a sub-expression list are either conjunctive (to be AND’ed) for DoC, or disjunctive (to be OR’ed) for CoD requests. The resulting expressions of a normalized select condition are either disjunctive (to be OR’ed) for DoC, or conjunctive (to be AND’ed) for CoD requests.
4 Data Types

The CMPI data types are the C data types that Management Instrumentations (Mis) should use in their code in order to encapsulate the specifics of the C data type. They are used in the data manipulation functions, in Management Broker (MB) services, and in the MI functions.

4.1 CMPI Encapsulated Data Types

A major requirement for CMPI is to hide implementation details of MB-specific data structures. CMPI uses pointers to opaque structures containing the MB-specific implementations, including a function pointer table providing type-specific support. This table, among others, contains MB-specific memory management support functions. In this document, the symbol <E-Type> is used to generically refer to the encapsulated types listed in the following table:

Table 1: Encapsulated Data Types

<table>
<thead>
<tr>
<th>CMPI Data Type &lt;E-Type&gt;</th>
<th>C Data Type</th>
</tr>
</thead>
<tbody>
<tr>
<td>CMPIInstance</td>
<td>struct _CMPIInstance</td>
</tr>
<tr>
<td>CMPIObjectPath</td>
<td>struct _CMPIObjectPath</td>
</tr>
<tr>
<td>CMPIArgs</td>
<td>struct _CMPIArgs</td>
</tr>
<tr>
<td>CMPIEnumeration</td>
<td>struct _CMPIEnumeration</td>
</tr>
<tr>
<td>CMPIArray</td>
<td>struct _CMPIArray</td>
</tr>
<tr>
<td>CMPISelectExp</td>
<td>struct _CMPISelectExp</td>
</tr>
<tr>
<td>CMPISelectCond</td>
<td>struct _CMPISelectCond</td>
</tr>
<tr>
<td>CMPISubCond</td>
<td>struct _CMPISubCond</td>
</tr>
<tr>
<td>CMPIPredicate</td>
<td>struct _CMPIPredicate</td>
</tr>
<tr>
<td>CMPIDateTime</td>
<td>struct _CMPIDateTime</td>
</tr>
<tr>
<td>CMPIContext</td>
<td>struct _CMPIContext</td>
</tr>
<tr>
<td>CMPIResult</td>
<td>struct _CMPIResult</td>
</tr>
</tbody>
</table>

struct _CMPIInstance;
struct _CMPIObjectPath;
struct _CMPIArgs;
struct _CMPIEnumeration;
struct _CMPIArray;
struct _CMPISelectExp;
struct _CMPISelectCond;
struct _CMPISubCond;
struct _CMPIPredicate;
struct _CMPIDateTime;
struct _CMPIContext;
struct _CMPIResult;
typedef CMPIInstance struct _CMPIInstance;
typedef CMPIObjectPath struct _CMPIObjectPath;
typedef CMPIArgs struct _CMPIArgs;
typedef CMPIEnumeration struct _CMPIEnumeration;
typedef CMPIArray struct _CMPIArray;
typedef CMPISelectExp struct _CMPISelectExp;
typedef CMPISelectCond struct _CMPISelectCond;
typedef CMPISubCond struct _CMPISubCond;
typedef CMPIPredicate struct _CMPIPredicate;
typedef CMPIContext struct _CMPIContext;
typedef CMPIResult struct _CMPIResult

4.2 CMPI String Data

For string data, two formats are supported, depending on usage: input to CMPI functions is as ordinary zero-terminated UTF-8 character arrays, and strings generated by CMPI functions are returned as pointers to CMPIString structures.

The reason for differentiating here is a matter of convenience and technical necessity:

1. Specifying a property name as input to a function using CMPIString would require using a CMPIString factory without any visible and/or measurable advantage.

2. Strings being returned from CMPI are dynamic in nature and minimally require memory management support. Therefore they follow the encapsulation concept.

Accessing the zero-terminated UTF-8 character arrays portion of a CMPIString is supported by a simple macro invocation.

Table 2: String Data Types

<table>
<thead>
<tr>
<th>CMPI Data Type</th>
<th>C Data Type</th>
</tr>
</thead>
<tbody>
<tr>
<td>CMPIString</td>
<td>struct _CMPIString</td>
</tr>
</tbody>
</table>

struct _CMPIString;
typedef _CMPIString struct CMPIString;

4.3 CMPI Simple Data Types

Most of the CMPI data type manipulation functions use as input and return simple data items such as integers, booleans, and floating-point numbers. All simple CIM types are supported via a convenience layer. The following is a mapping of simple CIM data types to the corresponding simple CMPI data types.

Table 3: Simple CMPI Data Types

<table>
<thead>
<tr>
<th>Simple CIM Data Type</th>
<th>CMPI Data Type &lt;C-Type&gt;</th>
<th>C Data Type</th>
</tr>
</thead>
<tbody>
<tr>
<td>boolean</td>
<td>CMPIBoolean</td>
<td>unsigned char (0: false, any other value: true)</td>
</tr>
<tr>
<td>char16</td>
<td>CMPIChar16</td>
<td>16-bit unsigned integer</td>
</tr>
<tr>
<td>Simple CIM Data Type</td>
<td>CMPI Data Type &lt;C-Type&gt;</td>
<td>C Data Type</td>
</tr>
<tr>
<td>---------------------</td>
<td>-------------------------</td>
<td>-------------</td>
</tr>
<tr>
<td>uint8</td>
<td>CMPIUint8</td>
<td>8-bit unsigned integer</td>
</tr>
<tr>
<td>uint16</td>
<td>CMPIUint16</td>
<td>16-bit unsigned integer</td>
</tr>
<tr>
<td>uint32</td>
<td>CMPIUint32</td>
<td>32-bit unsigned integer</td>
</tr>
<tr>
<td>uint64</td>
<td>CMPIUint64</td>
<td>64-bit unsigned integer</td>
</tr>
<tr>
<td>sint8</td>
<td>CMPIUint8</td>
<td>8-bit integer</td>
</tr>
<tr>
<td>sint16</td>
<td>CMPIUint16</td>
<td>16-bit integer</td>
</tr>
<tr>
<td>sint32</td>
<td>CMPIUint32</td>
<td>32-bit integer</td>
</tr>
<tr>
<td>sint64</td>
<td>CMPIUint64</td>
<td>64-bit integer</td>
</tr>
<tr>
<td>real32</td>
<td>CMPIReal32</td>
<td>32-bit floating-point</td>
</tr>
<tr>
<td>real64</td>
<td>CMPIReal64</td>
<td>64-bit floating-point</td>
</tr>
<tr>
<td>string</td>
<td>CMPIString (^2) char*</td>
<td>Encapsulated string type (Zero-terminated UTF-8 char array)</td>
</tr>
<tr>
<td>datetime</td>
<td>CMPIDateTime (^3)</td>
<td>Date/time structure</td>
</tr>
</tbody>
</table>

```c
typedef unsigned char          CMPIBoolean;
typedef unsigned short         CMPIChar16;
typedef unsigned char          CMPIUint8;
typedef unsigned short         CMPIUint16;
typedef unsigned long          CMPIUint32;
typedef unsigned long long     CMPIUint64;
typedef signed char            CMPIInt8;
typedef short                  CMPIInt16;
typedef long                   CMPIInt32;
typedef long long              CMPIInt64;
typedef float                  CMPIReal32;
typedef double                 CMPIReal64;
```

For the moment, it is assumed that compilers support 64-bit arithmetic data types and operations.\(^4\)

As to the floating-point declarations, it is assumed that MBs and MIs will use standard float/double ANSI C declarations and operations. Where there are compiler options/restrictions to the binary representations, MIs must follow MB implementation specifications.

---

\(^2\) For this discussion, **CMPIString** is considered to be a simple CIM data type, although it is an encapsulated type. See Section 4.2 for declaration and rationale.

\(^3\) **CMPIDateTime** is implemented as an encapsulated data type.

\(^4\) For situations where this is not the case, the **CMPIxint64** declarations may need to be conditional so that externally definable constructs can be used instead, assuming that the MB and MI can agree on a common format and the use of a common 64-bit emulation package.
Ultimately, when floating numbers have to be exposed to the outside world via DMTF-defined specifications, the MB has to transform these numbers into UTF-8 character strings as floating-point constants according to ANSI/IEEE Std 754-1985.

The same applies for little/big-endian dependencies.

### 4.4 CMPI Miscellaneous Data Types

**CMPIValuePtr** is used for context data only. It is used to describe raw unformatted data areas. It points to a data area and contains the length of the area.

<table>
<thead>
<tr>
<th>CMPI Data Type</th>
<th>C Data Type</th>
</tr>
</thead>
<tbody>
<tr>
<td>CMPIValuePtr</td>
<td>struct _CMPIValuePtr</td>
</tr>
</tbody>
</table>

```c
typedef struct _CMPIValuePtr {
    void *ptr;
    unsigned int length;
} CMPIValuePtr;
```

### 4.5 CMPI Types and Values

Three components are used to define value, state, and type (**CMPIValue**, **CMPIValueState**, and **CMPIType**). The value and type components are used by MIs when transferring property data to the MB (set operations). All three components are returned to an MI by the MB when accessing properties; in which case they are bundled in a structure called **CMPIData**.

#### 4.5.1 CMPIData

**CMPIData** is a structure that holds all three components returned to an MI when accessing property data using functions like `getProperty()`.

```c
typedef struct _CMPIData {
    CMPIType type;
    CMPIValueState state;
    CMPIValue value;
} CMPIData;
```

The `type` parameter is set to reflect the property type as defined in the schema, the `state` parameter indicates whether a value is set (not a NULL value) or originates from a **CMPIObjectPath** key-value, and the `value` parameter contains the actual value if not a NULL value.

#### 4.5.2 CMPIType

A type discriminator (**CMPIType**) is used to define data types of corresponding values. It is used either as part of a **CMPIData** structure, or as an additional parameter when setting property values.

```c
typedef unsigned short CMPIType;
```
#define CMPI_SIMPLE       (2)
#define CMPI_boolean      (2+0)
#define CMPI_char16       (2+1)

#define CMPI_REAL         ((2)<<2)
#define CMPI_real32       ((2+0)<<2)
#define CMPI_real64       ((2+1)<<2)

#define CMPI_UINT         ((8)<<4)
#define CMPI_uint8        ((8+0)<<4)
#define CMPI_uint16       ((8+1)<<4)
#define CMPI_uint32       ((8+2)<<4)
#define CMPI_uint64       ((8+3)<<4)
#define CMPI_SINT         ((8+4)<<4)
#define CMPI_sint8        ((8+4)<<4)
#define CMPI_sint16       ((8+5)<<4)
#define CMPI_sint32       ((8+6)<<4)
#define CMPI_sint64       ((8+7)<<4)
#define CMPI_INTEGER      ((CMPI_UINT | CMPI_SINT))

#define CMPI_ENC          ((16)<<8)
#define CMPI_instance     ((16+0)<<8)
#define CMPI_ref          ((16+1)<<8)
#define CMPI_args         ((16+2)<<8)
#define CMPI_class        ((16+3)<<8)
#define CMPI_filter       ((16+4)<<8)
#define CMPI_enumeration  ((16+5)<<8)
#define CMPI_string       ((16+6)<<8)
#define CMPI_chars        ((16+7)<<8)
#define CMPI_dateTime     ((16+8)<<8)
#define CMPI_ptr          ((16+9)<<8)

#define CMPI_ARRAY        ((1)<<13)
#define CMPI_SIMPLEA      (CMPI_ARRAY | CMPI_SIMPLE)
#define CMPI_booleanA     (CMPI_ARRAY | CMPI_boolean)
#define CMPI_char16A      (CMPI_ARRAY | CMPI_char16)

#define CMPI_REALA        (CMPI_ARRAY | CMPI_REAL)
#define CMPI_real32A      (CMPI_ARRAY | CMPI_real32)
#define CMPI_real64A      (CMPI_ARRAY | CMPI_real64)

#define CMPI_UINTA        (CMPI_ARRAY | CMPI_UINT)
#define CMPI_uint8A       (CMPI_ARRAY | CMPI_uint8)
#define CMPI_uint16A      (CMPI_ARRAY | CMPI_uint16)
#define CMPI_uint32A      (CMPI_ARRAY | CMPI_uint32)
#define CMPI_uint64A      (CMPI_ARRAY | CMPI_uint64)

#define CMPI_SINTA        (CMPI_ARRAY | CMPI_SINT)
#define CMPI_sint8A       (CMPI_ARRAY | CMPI_sint8)
#define CMPI_sint16A      (CMPI_ARRAY | CMPI_sint16)
#define CMPI_sint32A      (CMPI_ARRAY | CMPI_sint32)
#define CMPI_sint64A      (CMPI_ARRAY | CMPI_sint64)

#define CMPI_INTEGERA     (CMPI_ARRAY | CMPI_INTEGER)
#define CMPI_ENC          (CMPI_ARRAY | CMPI_ENC)
#define CMPI_stringA      (CMPI_ARRAY | CMPI_string)
#define CMPI_charsA       (CMPI_ARRAY | CMPI_chars)
#define CMPI_dateTimeA    (CMPI_ARRAY | CMPI_dateTime)
The following are CMPIObjectPath key-type synonyms and are valid only when CMPI_keyValue of CMPIValueState is set.

```
#define CMPI_keyInteger   (CMPI_sint64)
#define CMPI_keyString    (CMPI_string)
#define CMPI_keyBoolean   (CMPI_boolean)
#define CMPI_keyRef       (CMPI_ref)
```

The following are predicate types only.

```
#define CMPI_charString      (CMPI_string)
#define CMPI_numericString   (CMPI_string | CMPI_sint64)
#define CMPI_booleanString   (CMPI_string | CMPI_boolean)
#define CMPI_dateTimeString  (CMPI_string | CMPI_dateTime)
#define CMPI_classNameString (CMPI_string | CMPI_class)
```

Notice that CMPI_null is never returned by CMPI; it is used only by the MI with property setting operations to indicate a NULL value being set.

### 4.5.3 CMPIValueState

**CMPIValueState** describes the state of the actual value, whether it is set (not a NULL value) or emanated from a CMPIObjectPath key value.

```
typedef unsigned short CMPIValueState;
```

```
#define CMPI_nullValue (1<<8)
#define CMPI_keyValue  (2<<8)
#define CMPI_badValue  (0x80<<8)
```

### 4.5.4 CMPIValue

**CMPIValue** is a union that can hold any of the data types defined in CMPI.

```
typedef union _CMPIValue {
  CMPIBoolean             boolean;
  CMPICchar16              char16;
  CMPIUint8               uint8;
  CMPIUint16              uint16;
  CMPIUint32              uint32;
  CMPIUint64              uint64;
  CMPISint8               sint8;
  CMPISint16              sint16;
  CMPISint32              sint32;
  CMPISint64              sint64;
  CMPIReal32              real32;
  CMPIReal64              real64;
  CMPIInstance*           inst;
  CMPIObjectPath*         ref;
  CMPIArgs*               args;
  CMPISelectExp*          filter;
  CMPIEnumeration*        Enum;
  CMPIArray*              array;
  CMPIString*             string;
  char*                   chars;
  CMPIDateTime*           dateTime;
  CMPIValuePtr            dataPtr;
```
4.6 Null Value Specification

As mentioned in Section 3.1.3, CMPI supports the notion of NULL values. In the case that CMPI returns a `CMPIData` structure with `CMPIValue` set to all binary zeros, the `CMPI_nullValue` flag of `CMPIValueState` indicates that no value is available and `CMPIType` contains a valid type.

For transfer into CMPI two forms are supported: using a NULL pointer as a `CMPIValue` pointer, or having a NULL pointer in `CMPIValue` for an encapsulated data type pointer.

Examples

```c
CMPIInstance* ci;
CMPIValue val;

// Returning a simple NULL data item:
CMSetProperty(ci,"propName1",NULL,CMPI_sint32);

// Returning a NULL array data type:
CMSetProperty(ci,"propName2",NULL,CMPI_sint32A);

// Returning a pointer to a NULL array data type:
val.array=NULL;
CMSetProperty(ci,"propName2",&val,CMPI_sint32A);
```
MI Function Signatures

Management Implementation (MI) as used in this document corresponds to what is commonly known as a “Provider” in the CIMOM world. In the past, it has been common practice to subdivide functions offered via Providers into optional groups of related functions. CMPI supports five related groups of functions as follows:

- Instance MI
- Association MI
- Property MI
- Method MI
- Indication MI

The related groups are usually linked together in a dynamically loadable library that is known by the Management Broker (MB) using a naming convention. Every group is identifiable by the following duet:

```
load-lib-name
mi-name
```

The MB will determine these names either by convention or via “provider registration”. `Load-lib-name` is used by the MB to load a library where it expects to find MI groups defined above.

`Mi-name`, which is normally the same as the schema or class name, is used to locate the initializing functions within the library. This function is called to initialize the individual MI groups and returns the function table for this group.

Notice that all functions of implemented groups must be presented as valid entry points (not as a NULL pointer). When there are legitimate reasons not to support a specific function, a [CMPI_ERR_NOT_SUPPORTED] return code must be used to indicate this.

The signatures correspond largely with the similar constructs found in most CIM Object Manager implementations. CMPI has adopted the Pegasus Provider-2 concepts of handling return values. See Section 5.8 for more details.

All functions use CMPIrc enum as the return value. CMPI_RC_OK is considered successful completion. A positive value is considered an exceptional situation and will initiate an MB-specific error process.

---

5 Pegasus supports the notion of Class Providers. Though this in itself is a useful concept, it is assumed for the moment that this falls outside of the category of Providers providing MI and therefore will not be supported by CMPI.
5.1 MI Factory

MIs are loaded by the MB and instantiated by the MI factory which is invoked by an MB after it is loaded. The MI is represented by a `CMPI<mi-type>MI` structure. This structure has to be returned by the MI factory. Its major component is the function table representing the functions repertoire and a `void` pointer for CMPI implementation purposes.

`CMPI<mi-type>MI` can be extended by MIs for maintaining MI-specific data across MI function invocations. It should be noted, however, that MIs can be unloaded by an MB at any time.

5.1.1 `<mi-name>_Create<mi-type>MI`

```c
CMPIInstanceMI *
    <mi-name>_CreateInstanceMI(CMPIBroker*,CMPIContext*);
CMPIAssociationMI *
    <mi-name>_CreateAssociationMI(CMPIBroker*,CMPIContext*);
CMPIMethodMI *
    <mi-name>_CreateMethodMI(CMPIBroker*,CMPIContext*);
CMPIPropertyMI *
    <mi-name>_CreatePropertyMI(CMPIBroker*,CMPIContext*);
CMPIIndicationMI *
    <mi-name>_CreateIndicationMI(CMPIBroker*,CMPIContext*);
```

The `CMPIContext` object is used to pass context information to the initializing function for optimization purposes. Currently the optional context property `CMPIInitNameSpace` has been defined to signal the MI for which namespace initialization is to be performed.

Each function returns a pointer to its respective `CMPI<mi-type>MIIFT` function table structure.

5.1.2 `CMPI_MIType_xxx`

```c
#define CMPI_MIType_Instance 1
#define CMPI_MIType_Association 2
#define CMPI_MIType_Method 4
#define CMPI_MIType_Property 8
#define CMPI_MIType_Indication 16
```

5.1.3 `CMPIInstanceMIFT`

```c
struct _CMPIInstanceMIFT {
    int ftVersion;
    int miVersion;
    char *miName;
    CMPIStatus (*cleanup)
        (CMPIInstanceMI*,CMPIContext*);
    CMPIStatus (*enumInstanceNames)
```

---

6 The `<mi-type>` value can be either Instance, Association, Property, Method, or Indication.
5.1.4 CMPIAssociationMIFT

struct _CMPIAssociationsMIFT {
  int ftVersion;
  int miVersion;
  char *miName;
  CMPIStatus (*cleanup)(CMPIAssociationMI*, CMPIContext*);
  CMPIStatus (*associators)(CMPIAssociationMI*, CMPIContext*, CMPIResult*,
    CMPIObjectPath*, char*, char*, char*, char*, char*);
  CMPIStatus (*associatorNames)(CMPIAssociationMI*, CMPIContext*, CMPIResult*,
    CMPIObjectPath*, char*, char*, char*, char*);
  CMPIStatus (*references)(CMPIAssociationMI*, CMPIContext*, CMPIResult*,
    CMPIObjectPath*, char*, char*, char*);
  CMPIStatus (*referenceNames)(CMPIAssociationMI*, CMPIContext*, CMPIResult*,
    CMPIObjectPath*, char*, char*);
};

5.1.5 CMPIMethodMIFT

struct _CMPIMethodMIFT {
  int ftVersion;
  int miVersion;
  char *miName;
  CMPIStatus (*cleanup)(CMPIMethodMI*, CMPIContext*);
  CMPIStatus (*invokeMethod)(CMPIMethodMI*, CMPIContext*, CMPIResult*,
    CMPIObjectPath*, char*, CMPIArgs*, CMPIArgs*);
};
5.1.6 **CMPIPropertyMIFT**

```c
struct _CMPIPropertyMIFT {
    int ftVersion;
    int miVersion;
    char *miName;
    CMPIStatus (*cleanup)(CMPIPropertyMI*,CMPIContext*);
    CMPIStatus (*setProperty)(CMPIPropertyMI*,CMPIContext*,CMPIResult*,
                               CMPIObjectPath*,char*,CMPIValue*,CMPIType);
    CMPIStatus (*getProperty)(CMPIPropertyMI*,CMPIContext*,CMPIResult*,
                               CMPIObjectPath*,char*);
};
```

5.1.7 **CMPIIndicationMIFT**

```c
struct _CMPIIndicationMIFT {
    int ftVersion;
    int miVersion;
    char *miName;
    CMPIStatus (*cleanup)(CMPIIndicationMI*,CMPIContext*);
    CMPIStatus (*authorizeFilter)(CMPIIndicationMI*,CMPIContext*,CMPIResult*,
                                   CMPISelectExp*,char*,CMPIObjectPath*,char*);
    CMPIStatus (*mustPoll)(CMPIIndicationMI*,CMPIContext*,CMPIResult*,
                           CMPISelectExp*,char*,CMPIObjectPath*);
    CMPIStatus (*activateFilter)(CMPIIndicationMI*,CMPIContext*,CMPIResult*,
                                 CMPISelectExp*,char*,CMPIObjectPath*,CMPIBoolean);
    CMPIStatus (*deActivateFilter)(CMPIIndicationMI*,CMPIContext*,CMPIResult*,
                                   CMPISelectExp*,char*,CMPIObjectPath*,CMPIBoolean);
};
```

5.1.8 **Functions**

The MI Factory is accessed through the following function.
CMPIAssociationMIFT.cleanup()

NAME
CMPIAssociationMIFT.cleanup – perform cleanup prior to unloading the Association provider

SYNOPSIS
CMPIStatus CMPIAssociationMIFT.cleanup(
    CMPIAssociationMI* mi,
    CMPIContext* ctx
);  

DESCRIPTION
The CMPIAssociationMIFT.cleanup() function shall perform any necessary cleanup operation prior to the unloading of the library of which this MI group is part. This function is called prior to the unloading of the provider.

The mi argument is a pointer to a CMPIAssociationMI structure. The ctx argument is a pointer to a CMPIContext structure containing the Invocation Context.

RETURN VALUE
The CMPIAssociationMIFT.cleanup() function returns a CMPIStatus structure.

ERRORS
None.

EXAMPLES
None.

APPLICATION USAGE
None.

SEE ALSO
None.

CHANGE HISTORY
None.
NAME
CMPIIndicationMIFT.cleanup – perform cleanup prior to unloading the Indication provider

SYNOPSIS
CMPIStatus CMPIIndication.cleanup(
    CMPIIndicationMI* mi,
    CMPIContext* ctx
);

DESCRIPTION
The CMPIIndicationMIFT.cleanup() function shall perform any necessary cleanup
operation prior to the unloading of the library of which this MI group is part. This function
is called prior to the unloading of the provider.

The mi argument is a pointer to a CMPIIndicationMI structure. The ctx argument is a
pointer to a CMPIContext structure containing the Invocation Context.

RETURN VALUE
The CMPIIndicationMIFT.cleanup() function returns a CMPIStatus structure.

ERRORS
None.

EXAMPLES
None.

APPLICATION USAGE
None.

SEE ALSO
None.

CHANGE HISTORY
None.
CMPIInstanceMIFT.cleanup()

NAME
CMPIInstanceMIFT.cleanup – perform cleanup prior to unloading the Instance provider

SYNOPSIS
CMPIStatus CMPIInstanceMIFT.cleanup(
    CMPIInstanceMI* mi,
    CMPIContext* ctx
);

DESCRIPTION
The CMPIInstanceMIFT.cleanup() function shall perform any necessary cleanup operations prior to the unloading of the library of which this MI group is part. This function is called prior to the unloading of the provider.

The mi argument is a pointer to a CMPIInstanceMI structure. The ctx argument is a pointer to a CMPIContext structure containing the Invocation Context.

RETURN VALUE
The CMPIInstanceMIFT.cleanup() function returns CMPIStatus structure.

ERRORS
None.

EXAMPLES
None.

APPLICATION USAGE
None.

SEE ALSO
None.

CHANGE HISTORY
None.
CMPIMethodMIFT.cleanup()

NAME
CMPIMethodMIFT.cleanup – perform cleanup prior to unloading the Method provider

SYNOPSIS
CMPIServiceStatus CMPIMethodMIFT.cleanup(
    CMPIMethodMI* mi,
    CMPIClassContext* ctx
);

DESCRIPTION
The CMPIMethodMIFT.cleanup() function shall perform any necessary cleanup operation prior to the unloading of the library of which this MI group is part. This function is called prior to the unloading of the provider.

The mi argument is a pointer to a CMPIMethodMI structure. The ctx argument is a pointer to a CMPIClassContext structure containing the Invocation Context.

RETURN VALUE
The CMPIMethodMIFT.cleanup() function returns a CMPISStatus structure.

ERRORS
None.

EXAMPLES
None.

APPLICATION USAGE
None.

SEE ALSO
None.

CHANGE HISTORY
None.
CMPIPropertyMIFT.cleanup()

NAME
CMPIPropertyMIFT.cleanup – perform cleanup prior to unloading the Property provider

SYNOPSIS
CMPIStatus CMPIPropertyMIFT.cleanup(
  CMPIPropertyMI* mi,
  CMPIContext* ctx
);

DESCRIPTION
The CMPIPropertyMIFT.cleanup() function shall perform any necessary cleanup operations prior to the unloading of the library of which this MI group is part. This function is called prior to the unloading of the provider.

The mi argument is a pointer to a CMPIPropertyMI structure. The ctx argument is a pointer to a CMPIContext structure containing the Invocation Context.

RETURN VALUE
The CMPIPropertyMIFT.cleanup() function returns a CMPIStatus structure.

ERRORS
None.

EXAMPLES
None.

APPLICATION USAGE
None.

SEE ALSO
None.

CHANGE HISTORY
None.
5.2 Instance MI Signatures

5.2.1 CMPIFlags

The CMPIFlags type is used to inform MI functions about options specified by the client and passed on to the MI for certain requests. Normally, MIs will ignore these flags; however, these flags can be useful when MB services are invoked, or an external MB is contacted. CMPIFlags are not passed to MIs directly. MIs can use CMPIContext services to gain access under the name CMPIInvocationFlags.

typedef unsigned int CMPIFlags;

#define CMPI_FLAG_LocalOnly 1
#define CMPI_FLAG_DeepInheritance 2
#define CMPI_FLAG_IncludeQualifiers 4
#define CMPI_FLAG_IncludeClassOrigin 8

5.2.2 Functions

The Instance MI is accessed through the following functions.
CMPIInstanceMIFT.createInstance()

NAME
CMPIInstanceMIFT.createInstance – create an Instance using an ObjectPath as reference

SYNOPSIS
CMPIStatus CMPIInstanceMIFT.createInstance(
    CMPIInstanceMI* mi,
    CMPIContext* ctx,
    CMPIResult* rslt,
    CMPIObjectPath* op,
    CMPIInstance* inst
);  

DESCRIPTION
The CMPIInstanceMIFT.createInstance() function shall create an Instance using an ObjectPath as reference.

The \textit{mi} argument is a pointer to a CMPIInstanceMI structure. The \textit{ctx} argument is a pointer to a CMPIContext structure containing the Invocation Context.

The \textit{rslt} argument is a pointer to a CMPIResult structure that is the result data container.

The \textit{op} argument is a pointer to a CMPIObjectPath structure containing namespace, classname, and key components.

The \textit{inst} argument is a pointer to the created Instance.

RETURN VALUE
The CMPIInstanceMIFT.createInstance() function shall return a CMPIStatus structure containing the service return status.

ERRORS
None.

EXAMPLES
None.

APPLICATION USAGE
None.

SEE ALSO
returnData()

CHANGE HISTORY
None.
CMPIInstanceMIFT.deleteInstance()

NAME
CMPIInstanceMIFT.deleteInstance – delete an Instance defined by an ObjectPath

SYNOPSIS
CMPIStatus CMPIInstanceMIFT.deleteInstance(
  CMPIInstanceMI* mi,
  CMPIContext* ctx,
  CMPIResult* rslt,
  CMPIObjectPath* op
);

DESCRIPTION
The CMPIInstanceMIFT.deleteInstance() function shall create an Instance using an ObjectPath as reference.

The mi argument is a pointer to a CMPIInstanceMI structure. The ctx argument is a pointer to a CMPIContext structure containing the Invocation Context.

The rslt argument is a pointer to a CMPIResult structure that is the result data container.

The op argument is a pointer to a CMPIObjectPath structure containing namespace, class, and key components.

RETURN VALUE
The CMPIInstanceMIFT.deleteInstance() function shall return a CMPIStatus structure containing the service return status.

ERRORS
None.

EXAMPLES
None.

APPLICATION USAGE
None.

SEE ALSO
None.

CHANGE HISTORY
None.
CMPIInstanceMIFT.enumInstanceNames()

NAME
CMPIInstanceMIFT.enumInstanceNames – enumerate the ObjectPaths of Instances serviced by this provider

SYNOPSIS
CMPIStatus CMPIInstanceMIFT.enumInstanceNames(
    CMPIInstanceMI* thisMI,
    CMPIContext* ctx,
    CMPIResult* rslt,
    CMPIObjectPath* op
);

DESCRIPTION
None.

RETURN VALUE
The CMPIInstanceMIFT.enumInstanceNames() function shall enumerate the ObjectPaths of Instances serviced by this provider.

The mi argument is a pointer to a CMPIInstanceMI structure. The ctx argument is a pointer to a CMPIContext structure containing the Invocation Context.

The rslt argument is a pointer to a CMPIResult structure that is the result data container.

The op argument is a pointer to a CMPIObjectPath structure containing namespace and classname components.

ERRORS
The CMPIInstanceMIFT.enumInstanceNames() function shall return a CMPIStatus structure containing the service return status.

EXAMPLES
None.

APPLICATION USAGE
None.

SEE ALSO
returnData()

CHANGE HISTORY
None.
CMPIInstanceMIFT.enumInstances()

NAME

CMPIInstanceMIFT.enumInstances – enumerate the Instances serviced by this provider

SYNOPSIS

CMPIStatus CMPIInstanceMIFT.enumInstances(
    CMPIInstanceMI* thisMI,
    CMPIContext* ctx,
    CMPIResult* rslt,
    CMPIObjectPath* op,
    char*** properties
);

DESCRIPTION

The CMPIInstanceMIFT.enumInstances() function shall enumerate the ObjectPaths of Instances serviced by this provider.

The mi argument is a pointer to a CMPIInstanceMI structure. The ctx argument is a pointer to a CMPIContext structure containing the Invocation Context.

The rslt argument is a pointer to a CMPIResult structure that is the result data container.

The op argument is a pointer to a CMPIObjectPath structure containing namespace and classname components.

The properties argument, if not NULL, is an array of elements defining one or more Property names. Each returned Object must not include elements for any Properties missing from this list.

RETURN VALUE

The CMPIInstanceMIFT.enumInstances() function shall return a CMPIStatus structure containing the service return status.

ERRORS

None.

EXAMPLES

None.

APPLICATION USAGE

None.

SEE ALSO

returnData()

CHANGE HISTORY

None.
**NAME**

CMPIInstanceMIFT.execQuery – query the enumeration of Instances of the class (and subclass) defined by an ObjectPath

**SYNOPSIS**

```c
CMPIStatus CMPIInstanceMIFT.execQuery(
    CMPIInstanceMI* mi
    CMPIContext* ctx,
    CMPIResult* rslt,
    CMPIObjectPath* op,
    char* query,
    char* lang
);
```

**DESCRIPTION**

The `CMPIInstanceMIFT.execQuery()` function shall query the enumeration of Instances of the class (and subclass) defined by an ObjectPath.

The `mi` argument is a pointer to a `CMPIInstanceMI` structure. The `ctx` argument is a pointer to a `CMPIContext` structure containing the Invocation Context.

The `rslt` argument is a pointer to a `CMPIResult` structure that is the result data container.

The `op` argument is a pointer to a `CMPIObjectPath` structure containing namespace and classname components.

The `query` argument is a pointer to a string containing the select expression. The `lang` argument is a pointer to a string containing the query language.

**RETURN VALUE**

The `CMPIInstanceMIFT.execQuery()` function shall return a `CMPIStatus` structure containing the service return status.

**ERRORS**

None.

**EXAMPLES**

None.

**APPLICATION USAGE**

None.

**SEE ALSO**

`returnData()`

**CHANGE HISTORY**

None.
CMPIInstanceMIFT.getInstance()

NAME
CMPIInstanceMIFT.getInstance – get the Instances defined by an ObjectPath

SYNOPSIS
CMPIStatus CMPIInstanceMIFT.getInstance(
    CMPIInstanceMI* thisMI,
    CMPIContext* ctx,
    CMPIResult* rslt,
    CMPIObjectPath* op,
    char **properties
);

DESCRIPTION
The CMPIInstanceMIFT.getInstance() function shall get the Instances defined by an ObjectPath.

The mi argument is a pointer to a CMPIInstanceMI structure. The ctx argument is a pointer to a CMPIContext structure containing the Invocation Context.

The rslt argument is a pointer to a CMPIResult structure that is the result data container.

The op argument is a pointer to a CMPIObjectPath structure containing namespace and classname components.

The properties argument, if not NULL, is an array of elements defining one or more Property names. Each returned Object must not include elements for any Properties missing from this list.

RETURN VALUE
The CMPIInstanceMIFT.getInstance() function shall return a CMPIStatus structure containing the service return status.

ERRORS
None.

EXAMPLES
None.

APPLICATION USAGE
None.

SEE ALSO
returnData()

CHANGE HISTORY
None.
**CMPIInstanceMIFT.setInstance()**

**NAME**
CMPIInstanceMIFT.setInstance – replace an existing Instance using an ObjectPath as reference

**SYNOPSIS**
```c
CMPIStatus CMPIInstanceMIFT.setInstance(
    CMPIInstanceMI* mi,
    CMPIContext* ctx,
    CMPIResult* rslt,
    CMPIObjectPath* op,
    CMPIInstance* inst,
    char** properties
);
```

**DESCRIPTION**
The `CMPIInstanceMIFT.getInstance()` function shall replace an existing Instance using an ObjectPath as reference.

The `mi` argument is a pointer to a `CMPIInstanceMI` structure. The `ctx` argument is a pointer to a `CMPIContext` structure containing the Invocation Context.

The `rslt` argument is a pointer to a `CMPIResult` structure that is the result data container.

The `op` argument is a pointer to a `CMPIObjectPath` structure containing namespace, classname, and key components.

The `inst` argument is a pointer to a `CMPIInstance` structure containing the Instance.

The `properties` argument, if not NULL, is an array of elements defining one or more Property names. The process must not replace elements for any Properties missing from this list. If NULL all properties will be replaced.

**RETURN VALUE**
The `CMPIInstanceMIFT.getInstance()` function shall return a `CMPIStatus` structure containing the service return status.

**ERRORS**
None.

**EXAMPLES**
None.

**APPLICATION USAGE**
None.

**SEE ALSO**
None.

**CHANGE HISTORY**
None.
5.3 Association MI Signatures

The Association MI is accessed using the following functions.
NAME
CMPIAssociationMIFT.associatorNames – enumerate ObjectPaths associated with an Instance

SYNOPSIS
CMPIStatus CMPIAssociationMIFT.associatorNames(
    CMPIAssociationMI* mi,
    CMPIContext* ctx,
    CMPIResult* rslt,
    CMPIObjectPath* op,
    char* assocClass,
    char* resultClass,
    char* role,
    char* resultRole
);

DESCRIPTION
The CMPIAssociationMIFT.associatorNames() function shall enumerate ObjectPaths associated with an Instance.

The mi argument points to an Association Instance. The ctx argument points to the Invocation Context, and the rslt argument points to the result data container. The op argument points to the source ObjectPath containing namespace, classname, and key components.

The assocClass argument, if not NULL, shall be a valid Association Class name. It acts as a filter on the returned set of Objects by mandating that each returned Object must be associated to the source Object via an Instance of this Class or one of its subclasses.

The resultclass argument, if not NULL, shall be a valid Class name. It acts as a filter on the returned set of Objects by mandating that each returned Object must be either an Instance of this Class (or one of its subclasses).

The role argument, if not NULL, shall be a valid Property name. It acts as a filter on the returned set of Objects by mandating that each returned Object must be associated to the source Object via an Association in which the source Object plays the specified role (i.e. the name of the Property in the Association Class that refers to the source Object must match the value of this parameter).

The resultrole argument, if not NULL, shall be a valid Property name. It acts as a filter on the returned set of Objects by mandating that each returned Object must be associated to the source Object via an Association in which the returned Object plays the specified role (i.e. the name of the Property in the Association Class that refers to the returned Object must match the value of this parameter).

RETURN VALUE
The CMPIAssociationMIFT.associatorNames() shall return a CMPIStatus structure containing the service return status.

ERRORS
None.
EXAMPLES
None.

APPLICATION USAGE
None.

SEE ALSO

CHANGE HISTORY
None.
CMPIAssociationMIFT.associators()

NAME
CMPIAssociationMIFT.associators – enumerate Object Paths associated with an Instance

SYNOPSIS
CMPIStatus CMPIAssociationMIFT.associators(
    CMPIAssociationMI* mi,
    CMPIClass* assocClass,
    char* role,
    char* resultRole,
    char** properties
);

DESCRIPTION
The CMPIAssociationMIFT.associators() function shall enumerate ObjectPaths associated with an Instance.

The mi argument points to an Association Instance. The ctx argument points to the Invocation Context, and the rslt argument points to the result data container. The op argument points to the source ObjectPath containing namespace, classname, and key components.

The assocClass argument, if not NULL, shall be a valid Association Class name. It acts as a filter on the returned set of Objects by mandating that each returned Object must be associated to the source Object via an Instance of this Class or one of its subclasses.

The resultclass argument, if not NULL, shall be a valid Class name. It acts as a filter on the returned set of Objects by mandating that each returned Object must be either an Instance of this Class (or one of its subclasses).

The role argument, if not NULL, shall be a valid Property name. It acts as a filter on the returned set of Objects by mandating that each returned Object must be associated to the source Object via an Association in which the source Object plays the specified role (i.e. the name of the Property in the Association Class that refers to the source Object must match the value of this parameter).

The resultrole argument, if not NULL, shall be a valid Property name. It acts as a filter on the returned set of Objects by mandating that each returned Object must be associated to the source Object via an Association in which the returned Object plays the specified role (i.e. the name of the Property in the Association Class that refers to the returned Object must match the value of this parameter).

The properties argument, if not NULL, is an array of elements defining one or more Property names. Each returned Object must not include elements for any Properties missing from this list. If NULL all properties must be returned. The returned Object must match the value of this parameter).

RETURN VALUE
The CMPIAssociationMIFT.associatorNames() function shall return a CMPIStatus structure containing the service return status.
ERRORS
None.

EXAMPLES
None.

APPLICATION USAGE
None.

SEE ALSO

CHANGE HISTORY
None.
CMPIAssociationMIFT.referenceNames()

NAME
CMPIAssociationMIFT.referenceNames – enumerate the association ObjectPaths that refer to an Instance

SYNOPSIS
CMPIStatus CMPIAssociationMIFT.referenceNames(
    CMPIAssociationMI* mi,
    CMPIContext* ctx,
    CMPIResult* rslt,
    CMPIObjectPath* op,
    char* role,
    char* resultRole
);

DESCRIPTION
The CMPIAssociationMIFT.referenceNames() function shall enumerate the association ObjectPaths associated with an Instance.

The mi argument points to an Association Instance. The ctx argument points to the Invocation Context, and the rslt argument points to the result data container. The op argument points to the source ObjectPath containing namespace, classname, and key components.

The resultclass argument, if not NULL, shall be a valid Class name. It acts as a filter on the returned set of Objects by mandating that each returned Object must be either an Instance of this Class (or one of its subclasses).

The role argument, if not NULL, shall be a valid Property name. It acts as a filter on the returned set of Objects by mandating that each returned Object must be associated to the source Object via an Association in which the source Object plays the specified role (i.e. the name of the Property in the Association Class that refers to the source Object must match the value of this parameter).

RETURN VALUE
The CMPIAssociationMIFT.referenceNames() function shall return a CMPIStatus structure containing the service return status.

ERRORS
None.

EXAMPLES
None.

APPLICATION USAGE
None.

SEE ALSO
None.

CHANGE HISTORY
None.
NAME
CMPIAssociationMIFT.references – enumerate the association ObjectPaths that refer to an
Instance

CMPIStatus CMPIAssociationMIFT.references(
    CMPIAssociationMI* mi,
    CMPIContext* ctx,
    CMPIResult* rslt,
    CMPIObjectPath* objName,
    char* role,
    char* resultRole,
    char** properties
);

DESCRIPTION
The CMPIAssociationMIFT.references() function shall enumerate the association ObjectPaths
associated with an Instance.

The mi argument points to an Association Instance. The ctx argument points to the
Invocation Context, and the rslt argument points to the result data container. The op
argument points to the source ObjectPath containing namespace, classname, and key
components.

The resultclass argument, if not NULL, shall be a valid Class name. It acts as a filter on the
returned set of Objects by mandating that each returned Object must be either an Instance of
this Class (or one of its subclasses).

The role argument, if not NULL, shall be a valid Property name. It acts as a filter on the
returned set of Objects by mandating that each returned Object must be associated to the
source Object via an Association in which the source Object plays the specified role (i.e. the
name of the Property in the Association Class that refers to the source Object must match the
value of this parameter).

The properties argument, if not NULL, is an array of elements defining one or more
Property names. Each returned Object must not include elements for any Properties missing
from this list

RETURN VALUE
The CMPIAssociationMIFT.references() function shall return a CMPIStatus structure
containing the service return status.

ERRORS
None.

EXAMPLES
None.

APPLICATION USAGE
None.

SEE ALSO
CHANGE HISTORY

None.
5.4 Property MI Signatures

The Property MI is accessed using the following functions.
NAME

CMPIPropertyMIFT.getProperty – get a named property value of an Instance defined by an ObjectPath

SYNOPSIS

CMPIStatus CMPIPropertyMIFT.getProperty(
    CMPIPropertyMI* mi,
    CMPIContext* ctx,
    CMPIResult* rslt,
    CMPIObjectPath* op,
    char *name
);

DESCRIPTION

None.

RETURN VALUE

The CMPIPropertyMIFT.getProperty() function shall get a named property of an Instance defined by an ObjectPath.

The mi argument points to a CMPIPropertyMI structure.

The ctx argument points to a CMPIContext structure containing the Invocation Context.

The rslt argument points to a CMPIResult structure which is the result data container.

The op argument points to a CMPIObjectPath structure containing namespace, classname, and key components.

The name argument is a string containing the property name.

ERRORS

None.

EXAMPLES

None.

APPLICATION USAGE

The CMPIPropertyMIFT.getProperty() function shall return a CMPIStatus structure containing the service return status.

SEE ALSO

returnData()

CHANGE HISTORY

None.
NAME

CMPIPropertyMIFT.setProperty – set a named property value of an Instance defined by an ObjectPath

SYNOPSIS

CMPIStatus CMPIPropertyMIFT.setProperty(
    CMPIPropertyMI* mi,
    CMPIContext* ctx,
    CMPIResult* rslt,
    CMPIObjectPath* op,
    char* name,
    CMPIData data
);

DESCRIPTION

The CMPIPropertyMIFT.setProperty() function shall set a named property of an Instance defined by an ObjectPath.

The mi argument points to a CMPIPropertyMI structure.

The ctx argument points to a CMPIContext structure containing the Invocation Context.

The rslt argument points to a CMPIResult structure which is the result data container.

The op argument points to a CMPIObjectPath structure containing namespace, classname, and key components.

The name argument is a string containing the property name.

The data argument points to a CMPIData structure containing the new property value.

{Ed: Should this function have a CMPIResult argument? What does it return?}

RETURN VALUE

The CMPIPropertyMIFT.setProperty() function shall return a CMPIStatus structure containing the service return status.

ERRORS

None.

EXAMPLES

None.

APPLICATION USAGE

None.

SEE ALSO

None.

CHANGE HISTORY

None.
5.5 Method MI Signatures

The Method MI is accessed using the following function.
NAME
CMPIMethodMIFT.invokeMethod – invoke a named, extrinsic method of an Instance
defined by an ObjectPath

SYNOPSIS

```c
CMPStatus CMPIMethodMIFT.invokeMethod(
    CMPIMethodMI* mi,
    CMPIContext* ctx,
    CMPIResult* rslt,
    CMPIObjectPath* op,
    char* method,
    CMPIArg* in,
    CMPIArg* out
);
```

DESCRIPTION
The `CMPIMethodMIFT.invokeMethod()` function shall invoke a named, extrinsic method of
an instance defined by a specified ObjectPath.

The `mi` argument points to a `CMPIMethodMI` object. The `ctx` argument points to the
Invocation Context object, and the `op` argument points to the source ObjectPath containing
namespace, classname, and key elements.

The `method` argument points to a string containing the method name. The `in` argument points
to a `CMPIArgs` structure containing the input parameters. The `out` argument points to a
`CMPIArgs` structure containing the output parameters.

RETURN VALUE
The `CMPIMethodMIFT.invokeMethod()` function shall return a `CMPIStatus` structure
containing the service return status.

ERRORS
None.

EXAMPLES
None.

APPLICATION USAGE
None.

SEE ALSO

`CMPIArgsFT.getArg()`, `CMPIArgsFT.setArg()`, `returnData()`

CHANGE HISTORY
None.
5.6 Indication MI Signatures

The functions defined here are largely modeled after existing CIMOM implementations (Sun WBEMServices and The Open Group OpenCimom). They are, at the moment, the most widely used and therefore a de facto standard.

The Indication MI is accessed using the following functions.
NAME
CMPIIndicationMIFT.activateFilter – ???

SYNOPSIS
CMPIStatus CMPIIndicationMIFT.activateFilter(
    CMPIIndicationMI* thisMI,
    CMPIContext* ctx,
    CMPIResult* rslt,
    CMPISelectExp* filter,
    char *eventType,
    CMPIObjectPath* classPath,
    int firstActivation
);

DESCRIPTION
The filter parameter contains the filter specification for this subscription to become active.
This parameter can be passed back via the CMPIBrokerFT.deliverIndication() call enabling
optimization of Indication delivery processing.

RETURN VALUE
This function returns nothing.

ERRORS
None.

EXAMPLES
None.

APPLICATION USAGE
None.

SEE ALSO
CMPIBrokerFT.deliverIndication()

CHANGE HISTORY
None.
CMPIIndicationMIFT.authorizeFilter()

NAME

CMPIIndicationMIFT.authorizeFilter – ????

SYNOPSIS

CMPIStatus CMPIIndicationMIFT.authorizeFilter(
    CMPIIndicationMI* thisMI,
    CMPIContext* ctx,
    CMPIResult* rslt,
    CMPISelectExp* filter,
    Char *eventType,
    CMPIObjectPath* classPath,
    char *owner
);

DESCRIPTION

None.

RETURN VALUE

This function returns CMPIBoolean using returnData().

ERRORS

None.

EXAMPLES

None.

APPLICATION USAGE

None.

SEE ALSO

returnData()

CHANGE HISTORY

None.
CMPIIndicationMIFT.deActivateFilter()

NAME

CMPIIndicationMIFT.deActivateFilter – ???

SYNOPSIS

CMPIStatus CMPIIndicationMIFT.deActivateFilter(
    CMPIIndicationMI* thisMI,
    CMPIContext* ctx,
    CMPIResult* rslt,
    CMPISelectExp* filter,
    char *eventType,
    CMPIObjectPath* classPath,
    int lastActivation
);  

DESCRIPTION

None.

RETURN VALUE

This function returns nothing.

ERRORS

None.

EXAMPLES

None.

APPLICATION USAGE

None.

SEE ALSO

None.

CHANGE HISTORY

None.
NAME

CMPIIndicationMIFT.mustPoll – ????

SYNOPSIS

CMPIStatus CMPIIndicationMIFT.mustPoll(
    CMPIIndicationMI* thisMI,
    CMPIContext* ctx,
    CMPIDataResult* rslt,
    CMPISelectExp* filter,
    char *eventType,
    CMPIObjectPath* classPath
);

DESCRIPTION

This function enables very simple MIs to support indications without providing a complete
indication support implementation. When true is returned, the MB will enumerate the
instances of this MI at regular intervals and apply indication filters.

RETURN VALUE

This function returns CMPIBoolean using returnData().

ERRORS

None.

EXAMPLES

None.

APPLICATION USAGE

None.

SEE ALSO

returnData()

CHANGE HISTORY

None.
5.7 CMPI Return Codes

The following return codes are recognized:

```c
typedef enum _CMPIrc {
    CMPI_RC_OK = 0,
    CMPI_RC_ERR_FAILED = 1,
    CMPI_RC_ERR_ACCESS_DENIED = 2,
    CMPI_RC_ERR_INVALID_NAMESPACE = 3,
    CMPI_RC_ERR_INVALID_PARAMETER = 4,
    CMPI_RC_ERR_INVALID_CLASS = 5,
    CMPI_RC_ERR_NOT_FOUND = 6,
    CMPI_RC_ERR_NOT_SUPPORTED = 7,
    CMPI_RC_ERR_CLASS_HAS_CHILDREN = 8,
    CMPI_RC_ERR_CLASS_HAS_INSTANCES = 9,
    CMPI_RC_ERR_INVALID_SUPERCLASS = 10,
    CMPI_RC_ERR_ALREADY_EXISTS = 11,
    CMPI_RC_ERR_NO_SUCH_PROPERTY = 12,
    CMPI_RC_ERR_TYPE_MISMATCH = 13,
    CMPI_RC_ERR_QUERY_LANGUAGE_NOT_SUPPORTED = 14,
    CMPI_RC_ERR_INVALID_QUERY = 15,
    CMPI_RC_ERR_METHOD_NOT_AVAILABLE = 16,
    CMPI_RC_ERR_METHOD_NOT_FOUND = 17,
} CMPIrc;
```

5.8 CMPI Result Data Support

CMPI follows the Pegasus Provider 2 concept of returning result data. All return data produced by MI functions must be returned using the `CMPIResultFT.returnData()` function. Enumerating calls must use a series of these calls. Returning data must always be terminated via the `CMPIResultFT.returnDone()` function.

An example for `getProperty()` (all examples use the convenience macros):

```c
CMPIStatus sample_getProperty(
    CMPIPropertyMI* thisMI,
    CMPIContext* ctx,
    CMPIResult* rslt,
    CMPIObjectPath* cop,
    char *name)
{
    int rv=4711;
    CMReturnData(rslt,&rv,CMPI_sint32);
    CMReturn(CMPI_RC_OK);
}
```

An example for `enumInstanceNames()` returning a `CMPIObjectPath` enumeration:

```c
CMPIBroker *broker; // Set in <mi-name>_CreateInstanceMI.
CMPIStatus sample_enumInstanceNames(
    CMPIPropertyMI* thisMI,
    CMPIContext* ctx,
    CMPIResult* rslt,
    CMPIObjectPath* cop,
    char* *properties)
{
    ```
CMPIObjectPath *p1;
CMPIString ns;

for (int i=0; i<3; i++) {
    ns=CMGetNameSpace(cop);
    p1=CMNewObjectPath(broker,CMGetCharPtr(ns),"myClass",&rc);
    CMAddKey(p1,"id",&i,CMPI_sint32);
    CMReturnObjectPath(rslt,p1);
}
CMReturnDone(rslt);
CMReturn (CMPI_RC_OK);
CMPIResultFT.returnData()

NAME
CMPIResultFT.returnData – return a value/type pair

SYNOPSIS
CMPIStatus CMPIResultFT.returnData(
    CMPIResult* rslt,
    CMPIValue *value,
    CMPIType type
);

DESCRIPTION
The CMPIResultFT.returnData() function shall return a value/type pair from a CMPIResult structure.

The rslt argument points to a CMPIResult structure containing results to be returned.
The value argument points to a CMPIValue structure to contain the returned value.
The type argument points to a CMPIType structure to contain the returned type.

RETURN VALUE
The CMPIResultFT.returnData() function shall return a CMPIStatus structure containing the service return status.

ERRORS
None.

EXAMPLES
None.

APPLICATION USAGE
None.

SEE ALSO
None.

CHANGE HISTORY
None.
CMPIResultFT.returnDone()

NAME
CMPIResultFT.returnDone – indicate no further data is to be returned

SYNOPSIS
CMPIStatus CMPIResultFT.returnDone(
    CMPIResult* rslt
);

DESCRIPTION
The CMPIResultFT.returnDone() function shall indicate no further data is to be returned.
The rslt argument points to a CMPIResult structure containing results to be returned.

{Ed: IS this function used to indicate to the run time system that the application has finished
with the CMPIResult structure, or by the program to test if there are no further values to
return?}

RETURN VALUE
The CMPIResultFT.returnData() function shall return a CMPIStatus structure containing
the service return status.

ERRORS
None.

EXAMPLES
None.

APPLICATION USAGE
None.

SEE ALSO
CMPIResultFT.returnData(), invokeMethod()

CHANGE HISTORY
None.
NAME
CMPIResultFT.returnInstance – return an Instance object

SYNOPSIS
CMPIStatus CMPIResultFT.returnInstance(
    CMPIResult *rslt,
    CMPIInstance* inst
);  

DESCRIPTION
The CMPIResultFT.returnInstance() function shall return an Instance object from a CMPIResult structure.
The rslt argument points to a CMPIResult structure containing results to be returned.
The inst argument points to a CMPIInstance structure to contain the returned value.

RETURN VALUE
The CMPIResultFT.returnInstance() function shall return a CMPIStatus structure containing the service return status.

ERRORS
None.

EXAMPLES
None.

APPLICATION USAGE
None.

SEE ALSO
None.

CHANGE HISTORY
None.
CMPIResultFT.returnObjectPath()

NAME
CMPIResultFT.returnObjectPath – return an ObjectPath object

SYNOPSIS
CMPIStatus CMPIResultFT.returnInstance(
       CMPIResult* rslt,
       CMPIObjectPath* ref
);

DESCRIPTION
The CMPIResultFT.returnObjectPath() function shall return an ObjectPath object from a
CMPIResult structure.
The rslt argument points to a CMPIResult structure containing results to be returned.
The ref argument points to a CMPIObjectPath structure to contain the returned value.

RETURN VALUE
The CMPIResultFT.returnInstance() function shall return a CMPIStatus structure
containing the service return status.

ERRORS
None.

EXAMPLES
None.

APPLICATION USAGE
None.

SEE ALSO
None.

CHANGE HISTORY
None.
5.9 Context Data Support

As defined by Pegasus, an MB can pass unstructured data to the MI. The data is organized as name-value pairs. Arrays are supported, but not for type `CMPI_ptr`. Context data support is an optional feature; requests can be terminated by returning `[CMPI_RC_ERR_NOT_SUPPORTED]`

`CMPIContext` can also be used by MBs to carry the MB internal security context. In case MIs use threading and issue CMPI calls from threads, then `CMPIBrokerFT.prepareAttachThread()` and `CMPIBrokerFT.attachThread()` must be issued. This enables the MB to set up the correct security context for foreign threads.

Currently, two `CMPIContext` entry names are defined:

<table>
<thead>
<tr>
<th>Entry Name</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>CMPIInitNameSpace</td>
<td>CMPI_string</td>
<td>Namespace for which the MI is started.</td>
</tr>
<tr>
<td>CMPIInvocationFlags</td>
<td>CMPI_uint32</td>
<td><code>CMPIFlags</code> – invocation flags as specified by the client.</td>
</tr>
</tbody>
</table>
CMPIContextFT.addEntry()

NAME
CMPIContextFT.addEntry – adds/replaces a named Context entry

SYNOPSIS
CMPIStatus CMPIContextFT.addEntry(
    CMPIContext* ctx,
    char* name;
    CMPIValue* data,
    CMPIType type
)

DESCRIPTION
The CMPIContextFT.addEntry() function shall add or replace a named Context entry. The ctx argument points to a CMPIContext structure. The name argument is a string containing the context entry name. The data argument points to a CMPIData structure containing the data to be assigned to the context entry. The type argument is a CMPIType that defines the type of the data.

RETURN VALUE
The CMPIContextFT.addEntry() function shall return a CMPIStatus structure containing the service return status.

ERRORS
None.

EXAMPLES
None.

APPLICATION USAGE
None.

SEE ALSO
None.

CHANGE HISTORY
None.
NAME
CMPIContextFT.getEntry – get a Context entry by name

SYNOPSIS
CMPIData CMPIContextFT.getEntry(
    CMPIContext* ctx,
    char *name,
    CMPIStatus* rc
);

DESCRIPTION
The CMPIContextFT.getEntry() function shall get a named Context entry.
The ctx argument points to a CMPIContext structure. The name argument is a string containing the context entry name.
The rc argument points to a CMPIStatus structure containing the service return status.

RETURN VALUE
The CMPIContextFT.getEntry() function shall return a CMPIData structure containing the Context entry value.

ERRORS
None.

EXAMPLES
None.

APPLICATION USAGE
None.

SEE ALSO
None.

CHANGE HISTORY
None.
CMPIContextFT.getEntryAt()

NAME

CMPIContextFT.getEntryAt – get a Context entry defined by its index

SYNOPSIS

```
CMPIData CMPIContextFT.getEntryAt(
    CMPIContext* ctx,
    unsigned int index,
    CMPIString** name,
    CMPIStatus* rc
);
```

DESCRIPTION

The `CMPIContextFT.getEntryAt()` function shall get a Context defined by its index.

The `ctx` argument points to a `CMPIContext` structure.

The `index` argument defines the position of the entry in the internal data array.

The `name` argument points to a `CMPIString` structure that is updated with the name of the returned Context entry.

The `rc` argument points to a `CMPIStatus` structure containing the service return status.

RETURN VALUE

The `CMPIContextFT.getEntryAt()` function shall return a `CMPIData` structure containing the Context entry value.

ERRORS

None.

EXAMPLES

None.

APPLICATION USAGE

None.

SEE ALSO

None.

CHANGE HISTORY

None.
NAME
CMPIContextFT.getEntryAt – get the number of entries contained in a context

SYNOPSIS
unsigned int CMPIContextFT.getEntryAt(
    CMPIContext* ctx,
);

DESCRIPTION
The CMPIContextFT.getEntryCount() function shall get a the number of entries contained in
a context.
The ctx argument points to a CMPIContext structure.
{ED: Should there not be a *CMPIStatus argument to return a status value in cases of error,
such as the ctx pointer being invalid.}

RETURN VALUE
The CMPIContextFT.getEntryCount() function shall return a count of the number of entries
in the Context.

ERRORS
None.

EXAMPLES
None.

APPLICATION USAGE
None.

SEE ALSO
None.

CHANGE HISTORY
None.
Data Type Manipulation support is used to encapsulate Management Broker (MB)-specific implementation details. Support is provided for the following data types:

- CMPIInstance
- CMPIObjectPath
- CMPIArgs
- CMPIEnumeration
- CMPISelectExp
- CMPISelectCond
- CMPISubCond
- CMPIPredicate

The factories for the CMPI data types are available via `CMPIBrokerEncFT`.

```c
struct _CMPIBrokerEncFT {
    int ftVersion;
    CMPIInstance* CMPIInstance enumerate
                  (CMPIBroker*, CMPIInstance*, CMPIStatus*);
    CMPIObjectPath* (*newObjectPath) (CMPIBroker*, char*, CMPIStatus*);
    CMPIArgs* (*newArgs) (CMPIBroker*, char*, CMPIStatus*);
    CMPIString* (*newString) (CMPIBroker*, char*, CMPIStatus*);
    CMPIArray* (*newArray) (CMPIBroker*, CMPICount, CMPIStatus*);
    CMPIDateTime* (*newDateTime) (CMPIBroker*, CMPIStatus*);
    CMPIDateTime* (*newDateTimeFromBinary) (CMPIBroker*, CMPIDateTime, CMPIStatus*);
    CMPIDateTime* (*newDateTimeFromChars) (CMPIBroker*, char*, CMPIStatus*);
    CMPISelectExp* (*newSelectExp) (CMPIBroker*, char*, char*, CMPIArray**, CMPIStatus*);
    CMPIBoolean (*classPathIsA) (CMPIBroker*, CMPIObjectPath*, char*, CMPIStatus*);
    CMPIString* (*toString) (CMPIBroker*, void*, CMPIStatus*);
    CMPIBoolean (*isOfType) (CMPIBroker*, void*, char*, CMPIStatus*);
    CMPIString* (*getType) (CMPIBroker*, void*, CMPIStatus*);
    CMPIStatus (*logMessage) (CMPIBroker*, int, int, char*);
};
```

// Debugging support

```c
CMPIInstance* (*newInstance) (CMPIBroker*, CMPIContext*, CMPIObjectPath*, CMPIStatus*);
CMPIObjectPath* (*newObjectPath) (CMPIBroker*, char*, char*, CMPIStatus*);
CMPIArgs* (*newArgs) (CMPIBroker*, char*, CMPIStatus*);
CMPIString* (*newString) (CMPIBroker*, char*, CMPIStatus*);
CMPIArray* (*newArray) (CMPIBroker*, CMPICount, CMPIStatus*);
CMPIDateTime* (*newDateTime) (CMPIBroker*, CMPIStatus*);
CMPIDateTime* (*newDateTimeFromBinary) (CMPIBroker*, CMPIDateTime, CMPIStatus*);
CMPIDateTime* (*newDateTimeFromChars) (CMPIBroker*, char*, CMPIStatus*);
CMPISelectExp* (*newSelectExp) (CMPIBroker*, char*, char*, CMPIArray**, CMPIStatus*);
CMPIBoolean (*classPathIsA) (CMPIBroker*, CMPIObjectPath*, char*, CMPIStatus*);
CMPIString* (*toString) (CMPIBroker*, void*, CMPIStatus*);
CMPIBoolean (*isOfType) (CMPIBroker*, void*, char*, CMPIStatus*);
CMPIString* (*getType) (CMPIBroker*, void*, CMPIStatus*);
CMPIStatus (*logMessage) (CMPIBroker*, int, int, char*);
```
6.1 Miscellaneous Services

The following miscellaneous services are defined.
CMPIArgsFT.clone()

NAME
CMPIArgsFT.clone – create an independent copy of an Args object

SYNOPSIS
CMPIArgs* CMPIArgsFT.clone(
    CMPIArgs* as,
    CMPIStatus* rc
);

DESCRIPTION
The CMPIArgsFT.clone() function shall create an independent copy of an Args object.
The as argument points to the CMPIArgs structure to be copied.
The rc argument points to a CMPIStatus structure containing the service return status.
The resulting Args object must be explicitly released.

RETURN VALUE
The CMPIArgsFT.clone() function shall return a pointer to the copied Args object.

ERRORS
None.

EXAMPLES
None.

APPLICATION USAGE
None.

SEE ALSO
None.

CHANGE HISTORY
None.
NAME
CMPIArgsFT.release – the Args object will not be used any further and may be freed by the
CMPI run time system

SYNOPSIS
CMPIStatus CMPIArgsFT.release(
    CMPIArgs* as
);

DESCRIPTION
The CMPIArgsFT.release() function shall indicate to that an Args object will not be used
any further, and may be freed by the CMPI run time system.

The as argument points to a CMPIArgs structure.

RETURN VALUE
The CMPIArgsFT.release() function shall return a CMPIStatus structure containing the
service return status.

ERRORS
None.

EXAMPLES
None.

APPLICATION USAGE
None.

SEE ALSO
None.

CHANGE HISTORY
None.
**NAME**
CMPIArrayFT.clone – create an independent copy of an Array object

**SYNOPSIS**
```c
CMPIArray* CMPIArrayFT.clone(
    CMPIArray* ar,
    CMPIStatus* rc
);
```

**DESCRIPTION**
The `CMPIArrayFT.clone()` function shall create an independent copy of an Array object.
The `ar` argument points to the `CMPIArray` structure to be copied.
The `rc` argument points to a `CMPIStatus` structure containing the service return status.
The resulting Array object must be explicitly released.

**RETURN VALUE**
The `CMPIArrayFT.clone()` function shall return a pointer to the copied Array object.

**ERRORS**
None.

**EXAMPLES**
None.

**APPLICATION USAGE**
None.

**SEE ALSO**
None.

**CHANGE HISTORY**
None.
CMPIArrayFT.release()

NAME

CMPIArrayFT.release – the Array object will not be used any further and may be freed by the CMPI run time system

SYNOPSIS

CMPIStatus CMPIArrayFT.release(
    CMPIArray* ar
);

DESCRIPTION

The CMPIArrayFT.release() function shall indicate to that an Array object will not be used any further, and may be freed by the CMPI run time system.

The ar argument points to a CMPIArray structure.

RETURN VALUE

The CMPIArrayFT.release() function shall return the service return status.

ERRORS

None.

EXAMPLES

None.

APPLICATION USAGE

None.

SEE ALSO

None.

CHANGE HISTORY

None.
CMPIBrokerEncFT.classPathIsA()

NAME
CMPIBrokerFT.classpathIsA – determines whether a CIM class is of a specified type or any of its subclasses

SYNOPSIS
CMPIBoolean CMPIBrokerEncFT.classPathIsA(
    CMPIBroker* mb,
    CMPIObjectPath* op,
    char* type,
    CMPIStatus* rc
);

DESCRIPTION
The CMPIBrokerEncFT.classPathIsA() function shall determine whether a CIM class is of a specified type or any of that type’s subclasses.

The mb argument points to a CMPIBroker structure. The op argument points to a CMPIObjectPath structure. The type argument points to the type to be tested for.

The rc argument points to a CMPIStatus structure containing the service return status.

RETURN VALUE
The CMPIBrokerEncFT.classPathIsA() function shall return true if the test is successful.

ERRORS
None.

EXAMPLES
None.

APPLICATION USAGE
None.

SEE ALSO
None.

CHANGE HISTORY
None.
CMPIBrokerEncFT.getType()

NAME
CMPIBrokerEncFT.getType – retrieves the CMPI type of an object

SYNOPSIS
CMPIString* CMPIBrokerEncFT.getType(
    CMPIBroker* mb,
    void* object,
    CMPIStatus* rc
);

DESCRIPTION
The CMPIBrokerEncFT.getType() function shall return the CMPI type of an object.

The mb argument points to a CMPIBroker structure. The object argument is a pointer to a
valid CMPI object.

The rc argument points to a CMPIStatus structure containing the service return status.

This function is intended for debugging purposes only.

RETURN VALUE
The CMPIBrokerEncFT.getType() function shall return a pointer to a CMPIString structure
containing the encapsulated object type.

ERRORS
None.

EXAMPLES
None.

APPLICATION USAGE
None.

SEE ALSO
None.

CHANGE HISTORY
None.
CMPIBrokerEncFT.isOfType()

NAME
CMPIBrokerFT.isOfType – verify whether an object is of a specified CMPI type.

CMPIBoolean *CMPIBrokerEncFT.isOfType(
    CMPIBroker* mb,
    void* object,
    char* type,
    CMPIStatus* rc
);

DESCRIPTION
The CMPIBrokerEncFT.isOfType() function shall return whether an object is of a specified CMPI type.

The mb argument points to a CMPIBroker structure. The object argument is a pointer to a valid CMPI object. The type argument points to a string specifying a valid CMPI object type to be tested for.

The rc argument points to a CMPIStatus structure containing the service return status.

This function is intended for debugging purposes only.

RETURN VALUE
The CMPIBrokerEncFT.isOfType() function shall return true if the test is successful.

ERRORS
None.

EXAMPLES
None.

APPLICATION USAGE
None.

SEE ALSO
None.

CHANGE HISTORY
None.
**NAME**

CMPIBrokerEncFT.toString – attempts to transform a CMPI object into an implementation-specific string representation

**SYNOPSIS**

```c
CMPIString* CMPIBrokerEncFT.toString(
    CMPIBroker* broker,
    void* object,
    CMPIStatus* rc
);
```

**DESCRIPTION**

The `CMPIBrokerEncFT.toString()` function attempts to transform a string into an implementation-specific string representation.

The `mb` argument is a pointer to a `CMPIBroker` structure. The `object` argument is a pointer to a valid CMPI object.

The `rc` argument points to a `CMPIStatus` structure used to return the service return code.

**RETURN VALUE**

The `CMPIBrokerEncFT.toString()` function returns a pointer to a `CMPIString` structure containing the string representation. Output will vary depending on the specific implementation.

**ERRORS**

None.

**EXAMPLES**

None.

**APPLICATION USAGE**

None.

**SEE ALSO**

None.

**CHANGE HISTORY**

None.
NAME
CMPIContextFT.clone – create an independent copy of an Context object

SYNOPSIS
CMPIContext* CMPIContextFT.clone(
   CMPIContext* ctx,
   CMPIStatus* rc
);

DESCRIPTION
The CMPIContextFT.clone() function shall create an independent copy of an Context object.
The ctx argument points to the CMPIContext structure to be copied.
The rc argument points to a CMPIStatus structure containing the service return status.
The resulting Context object must be explicitly released.

RETURN VALUE
The CMPIContextFT.clone() function shall return a pointer to the copied Context object.

ERRORS
None.

EXAMPLES
None.

APPLICATION USAGE
None.

SEE ALSO
None.

CHANGE HISTORY
None.
NAME
CMPIContextFT.release – the Context object will not be used any further and may be freed
by the CMPI run time system.

SYNOPSIS
CMPIStatus CMPIContextFT.release(
    CMPIContext* ctx
);

DESCRIPTION
The CMPIContextFT.release() function shall indicate to that an Context object will not be
used any further, and may be freed by the CMPI run time system.
The ctx argument points to a CMPIContext structure.

RETURN VALUE
The CMPIContextFT.release() function shall return a CMPIStatus structure containing the
service return status.

ERRORS
None.

EXAMPLES
None.

APPLICATION USAGE
None.

SEE ALSO
None.

CHANGE HISTORY
None.
NAME

CMPIDateTimeFT.clone – create an independent copy of a DateTime object

SYNOPSIS

CMPIDateTime* CMPIDateTimeFT.clone(
    CMPIDateTime* dt,
    CMPIStatus* rc
);

DESCRIPTION

The CMPIDateTimeFT.clone() function shall create an independent copy of a DateTime object.

The dt argument points to the CMPIDateTime structure to be copied. The rc argument points to a CMPIStatus structure containing the service return status.

The resulting DateTime object must be explicitly released.

RETURN VALUE

The CMPIDateTimeFT.clone() function shall return a pointer to the copied DateTime object.

ERRORS

None.

EXAMPLES

None.

APPLICATION USAGE

None.

SEE ALSO

None.

CHANGE HISTORY

None.
NAME
CMPIDateTimeFT.release – the DateTime object will not be used any further and may be freed by the CMPI run time system.

SYNOPSIS
CMPIStatus CMPIDateTimeFT.release(
    CMPIDateTime* dt
);

DESCRIPTION
The CMPIDateTimeFT.release() function shall indicate to that an DateTime object will not be used any further, and may be freed by the CMPI run time system.

The dt argument points to a CMPIDateTime structure.

RETURN VALUE
The CMPIDateTimeFT.release() function shall return a CMPIStatus structure containing the service return status.

ERRORS
None.

EXAMPLES
None.

APPLICATION USAGE
None.

SEE ALSO
None.

CHANGE HISTORY
None.
CMPIEnumerationFT.clone()

NAME

CMPIEnumerationFT.clone – create an independent copy of an Enumeration object

SYNOPSIS

CMPIEnumeration* CMPIEnumerationFT.clone(
    CMPIEnumeration* en,
    CMPIStatus* rc
);

DESCRIPTION

The CMPIEnumerationFT.clone() function shall create an independent copy of an Enumeration object.

The en argument points to the CMPIEnumeration structure to be copied. The rc argument points to a CMPIStatus structure containing the service return status.

The resulting Enumeration object must be explicitly released.

RETURN VALUE

The CMPIEnumerationFT.clone() function shall return a pointer to the copied Enumeration object.

ERRORS

None.

EXAMPLES

None.

APPLICATION USAGE

None.

SEE ALSO

None.

CHANGE HISTORY

None.
NAME  
CMPIEnumerationFT.release – the Enumeration object will not be used any further and my 
be freed by the CMPI run time system

SYNOPSIS  
CMPIStatus CMPIEnumerationFT.release( 
  CMPIEnumeration* en 
);  

DESCRIPTION  
The CMPIEnumerationFT.release() function shall indicate to that an Enumeration object 
will not be used any further, and may be freed by the CMPI run time system. 
The en argument points to a CMPIEnumeration structure.

RETURN VALUE  
The CMPIEnumerationFT.release() function shall return a CMPIStatus structure containing 
the service return status.

ERRORS  
None.

EXAMPLES  
None.

APPLICATION USAGE  
None.

SEE ALSO  
None.

CHANGE HISTORY  
None.
NAME
CMPIInstanceFT.clone – create an independent copy of an Instance object

SYNOPSIS
CMPIInstance* CMPIInstanceFT.clone(
    CMPIInstance* inst,
    CMPIStatus* rc
);

DESCRIPTION
The CMPIInstanceFT.clone() function shall create an independent copy of an Instance object.

The inst argument points to the CMPIInstance structure to be copied. The rc argument points to a CMPIStatus structure containing the service return status.

The resulting Instance object must be explicitly released.

RETURN VALUE
The CMPIInstanceFT.clone() function shall return a pointer to the copied Instance object.

ERRORS
None.

EXAMPLES
None.

APPLICATION USAGE
None.

SEE ALSO
None.

CHANGE HISTORY
None.
CMPIInstanceFT.release()

NAME
CMPIInstanceFT.release – the Instance object will not be used any further and may be freed by the CMPI run time system.

SYNOPSIS
CMPIStatus CMPIInstanceFT.release(
    CMPIInstance* inst
);

DESCRIPTION
The CMPIInstanceFT.release() function shall indicate to that an Instance object will not be used any further, and may be freed by the CMPI run time system.

The inst argument points to a CMPIInstance structure.

RETURN VALUE
The CMPIInstanceFT.release() function shall return a CMPIStatus structure containing the service return status.

ERRORS
None.

EXAMPLES
None.

APPLICATION USAGE
None.

SEE ALSO
None.

CHANGE HISTORY
None.
NAME
CMPIObjectPathFT.clone – create an independent copy of an ObjectPath object

SYNOPSIS
CMPIObjectPath* CMPIObjectPathFT.clone(
    CMPIObjectPath* op,
    CMPIStatus* rc
);

DESCRIPTION
The CMPIObjectPathFT.clone() function shall create an independent copy of an ObjectPath object.

The op argument points to the CMPIObjectPath structure to be copied. The rc argument points to a CMPIStatus structure containing the service return status.

The resulting ObjectPath object must be explicitly released.

RETURN VALUE
The CMPIObjectPathFT.clone() function shall return a pointer to the copied ObjectPath object.

ERRORS
None.

EXAMPLES
None.

APPLICATION USAGE
None.

SEE ALSO
None.

CHANGE HISTORY
None.
CMPIObjectPathFT.release()

NAME
CMPIObjectPathFT.release – the ObjectPath object will not be used any further and may be freed by the CMPI run time system

SYNOPSIS
CMPIStatus CMPIObjectPathFT.release(
    CMPIObjectPath* op
);

DESCRIPTION
The CMPIObjectPathFT.release() function shall indicate to that an ObjectPath object will not be used any further, and may be freed by the CMPI run time system.

The op argument points to a CMPIObjectPath structure.

RETURN VALUE
The CMPIObjectPathFT.release() function shall return a CMPIStatus structure containing the service return status.

ERRORS
None.

EXAMPLES
None.

APPLICATION USAGE
None.

SEE ALSO
None.

CHANGE HISTORY
None.
CMPIPredicateFT.clone()

NAME
CMPIPredicateFT.clone – create an independent copy of an Predicate object

SYNOPSIS
CMPIPredicate* CMPIPredicateFT.clone(
    CMPIPredicate* pr,
    CMPIStatus* rc
) ;

DESCRIPTION
The CMPIPredicateFT.clone() function shall create an independent copy of an Predicate object.

The pr argument points to the CMPIPredicate structure to be copied. The rc argument points to a CMPIStatus structure containing the service return status.

The resulting Predicate object must be explicitly released.

RETURN VALUE
The CMPIPredicateFT.clone() function shall return a pointer to the copied Predicate object.

ERRORS
None.

EXAMPLES
None.

APPLICATION USAGE
None.

SEE ALSO
None.

CHANGE HISTORY
None.
CMIPredicateFT.release()

NAME
CMIPredicateFT.release – the Predicate object will not be used any further and may be freed by the CMPI run time system.

SYNOPSIS
CMPIStatus CMIPredicateFT.release(CMIPredicate* pr);

DESCRIPTION
The CMIPredicateFT.release() function shall indicate to that an Predicate object will not be used any further, and may be freed by the CMPI run time system.

The pr argument points to a CMIPredicate structure.

RETURN VALUE
The CMIPredicateFT.release() function shall return a CMPIStatus structure containing the service return status.

ERRORS
None.

EXAMPLES
None.

APPLICATION USAGE
None.

SEE ALSO
None.

CHANGE HISTORY
None.
CMPIResultFT.clone()

NAME
CMPIResultFT.clone – create an independent copy of an Result object

SYNOPSIS
CMPIResult* CMPIResultFT.clone(
    CMPIResult* rslt,
    CMPIStatus* rc
);

DESCRIPTION
The CMPIResultFT.clone() function shall create an independent copy of an Result object. The rslt argument points to the CMPIResult structure to be copied. The rc argument points to a CMPIStatus structure containing the service return status. The resulting Result object must be explicitly released.

RETURN VALUE
The CMPIResultFT.clone() function shall return a pointer to the copied Result object.

ERRORS
None.

EXAMPLES
None.

APPLICATION USAGE
None.

SEE ALSO
None.

CHANGE HISTORY
None.
CMPIResultFT.release()

NAME

CMPIResultFT.release – the Result object will not be used any further and may be freed by the CMPI run time system.

SYNOPSIS

CMPIStatus CMPIResultFT.release(
    CMPIResult* rslt
);  

DESCRIPTION

The CMPIResultFT.release() function shall indicate to that an Result object will not be used any further, and may be freed by the CMPI run time system. The rslt argument points to a CMPIResult structure.

RETURN VALUE

The CMPIResultFT.release() function shall return a CMPIStatus structure containing the service return status.

ERRORS

None.

EXAMPLES

None.

APPLICATION USAGE

None.

SEE ALSO

None.

CHANGE HISTORY

None.
NAME

CMPISelectCondFT.clone – create an independent copy of an SelectCond object

SYNOPSIS

CMPISelectCond* CMPISelectCondFT.clone(
    CMPISelectCond* sc,
    CMPIStatus* rc
);

DESCRIPTION

The CMPISelectCondFT.clone() function shall create an independent copy of an SelectCond object. The sc argument points to the CMPISelectCond structure to be copied. The rc argument points to a CMPIStatus structure containing the service return status. The resulting SelectCond object must be explicitly released.

RETURN VALUE

The CMPISelectCondFT.clone() function shall return a pointer to the copied SelectCond object.

ERRORS

None.

EXAMPLES

None.

APPLICATION USAGE

None.

SEE ALSO

None.

CHANGE HISTORY

None.
CMPISelectCondFT.release()

NAME
CMPISelectCondFT.release – the SelectCond object will not be used any further and may be freed by the CMPI run time system.

SYNOPSIS
CMPIStatus CMPISelectCondFT.release(
    CMPISelectCond* sc
);

DESCRIPTION
The CMPISelectCondFT.release() function shall indicate to that an SelectCond object will not be used any further, and may be freed by the CMPI run time system.

The sc argument points to a CMPISelectCond structure.

RETURN VALUE
The CMPISelectCondFT.release() function shall return a CMPIStatus structure containing the service return status.

ERRORS
None.

EXAMPLES
None.

APPLICATION USAGE
None.

SEE ALSO
None.

CHANGE HISTORY
None.
CMPISelectExpFT.clone()

NAME
CMPISelectExpFT.clone – create an independent copy of an SelectExp object

SYNOPSIS
CMPISelectExp* CMPISelectExpFT.clone(
    CMPISelectExp* se,
    CMPIStatus* rc
);  

DESCRIPTION
The CMPISelectExpFT.clone() function shall create an independent copy of an SelectExp object.

The se argument points to the CMPISelectExp structure to be copied. The rc argument points to a CMPIStatus structure containing the service return status.

The resulting SelectExp object must be explicitly released.

RETURN VALUE
The CMPISelectExpFT.clone() function shall return a pointer to the copied SelectExp object.

ERRORS
None.

EXAMPLES
None.

APPLICATION USAGE
None.

SEE ALSO
None.

CHANGE HISTORY
None.
**CMPISelectExpFT.release()**

**NAME**
CMPISelectExpFT.release – the SelectExp object will not be used any further and may be freed by the CMPI run time system

**SYNOPSIS**
```c
CMPIStatus CMPISelectExpFT.release(
    CMPISelectExp* se
);
```

**DESCRIPTION**
The `CMPISelectExpFT.release()` function shall indicate to the SelectExp object that it will not be used any further, and may be freed by the CMPI run time system. The `se` argument points to a `CMPISelectExp` structure.

**RETURN VALUE**
The `CMPISelectExpFT.release()` function shall return a `CMPIStatus` structure containing the service return status.

**ERRORS**
None.

**EXAMPLES**
None.

**APPLICATION USAGE**
None.

**SEE ALSO**
None.

**CHANGE HISTORY**
None.
NAME
CMPIStringFT.clone – create an independent copy of an String object

SYNOPSIS
CMPIString* CMPIStringFT.clone(
    CMPIString* st,
    CMPIStatus* rc
);

DESCRIPTION
The CMPIStringFT.clone() function shall create an independent copy of an String object.
The st argument points to the CMPIString structure to be copied. The rc argument points to
a CMPIStatus structure containing the service return status.
The resulting String object must be explicitly released.

RETURN VALUE
The CMPIStringFT.clone() function shall return a pointer to the copied String object.

ERRORS
None.

EXAMPLES
None.

APPLICATION USAGE
None.

SEE ALSO
None.

CHANGE HISTORY
None.
NAME

CMPIStringFT.release – the String object will not be used any further and may be freed by
the CMPI run time system

SYNOPSIS

CMPIStatus CMPIStringFT.release(
    CMPIString* st
);

DESCRIPTION

The CMPIStringFT.release() function shall indicate to that an String object will not be used
any further, and may be freed by the CMPI run time system.

The st argument points to a CMPIString structure.

RETURN VALUE

The CMPIStringFT.release() function shall return a CMPIStatus structure containing the
service return status.

ERRORS

None.

EXAMPLES

None.

APPLICATION USAGE

None.

SEE ALSO

None.

CHANGE HISTORY

None.
NAME

CMPISubCondFT.clone – create an independent copy of an SubCond object

SYNOPSIS

CMPISubCond* CMPISubCondFT.clone(
    CMPISubCond* sc,
    CMPIStatus* rc
);

DESCRIPTION

The CMPISubCondFT.clone() function shall create an independent copy of an SubCond object.

The sc argument points to the CMPISubCond structure to be copied.

The rc argument points to a CMPIStatus structure containing the service return status.

The resulting SubCond object must be explicitly released.

RETURN VALUE

The CMPISubCondFT.clone() function shall return a pointer to the copied SubCond object.

ERRORS

None.

EXAMPLES

None.

APPLICATION USAGE

None.

SEE ALSO

None.

CHANGE HISTORY

None.
CMPISubCondFT.release()

NAME

CMPISubCondFT.release – the SubCond object will not be used any further and may be freed by the CMPI run time system.

SYNOPSIS

CMPIStatus CMPISubCondFT.release(
    CMPISubCond* sc
);  

DESCRIPTION

The `CMPISubCondFT.release()` function shall indicate to that an SubCond object will not be used any further, and may be freed by the CMPI run time system.

The `sc` argument points to a `CMPISubCond` structure.

RETURN VALUE

The `CMPISubCondFT.release()` function shall return a `CMPIStatus` structure containing the service return status.

ERRORS

None.

EXAMPLES

None.

APPLICATION USAGE

None.

SEE ALSO

None.

CHANGE HISTORY

None.
6.2 CMPIString Support

CMPIString support is provided by the following function.
CMPIMbrokerEncFT.newString()

NAME

CMPIbrokerEncFT.newString – create a new String object

SYNOPSIS

CMPIString* CMPIMbrokerEncFT.newString(
    CMPIPbroker* mb,
    char* data,
    CMPIStatus* rc
);

DESCRIPTION

The CMPIMbrokerEncFT.newString() function shall return a new CMPIString object.

The mb argument points to a CMPIPbroker object. The data argument is a pointer to the data
to initialize the new CMPIString structure.

The rc argument points to a CMPIStatus structure containing the service return status.

RETURN VALUE

The CMPIMbrokerEncFT.newString() function shall return a pointer to a new CMPIString
object.

ERRORS

None.

EXAMPLES

None.

APPLICATION USAGE

None.

SEE ALSO

None.

CHANGE HISTORY

None.
CMPIStringFT.getCharPtr()

NAME
CMPIStringFT.getCharPtr – get a pointer to a C language string representation of a String

SYNOPSIS
char* CMPIStringFT.getCharPtr(
    CMPIString* st,
    CMPIStatus* rc
);

DESCRIPTION
The CMPIStringFT.getCharPtr() function shall return a C language character pointer representation of the string contained in a CMPIString structure.

The st argument is a pointer to a CMPIString structure.

The rc argument points to a CMPIStatus structure containing the service return status.

RETURN VALUE
The CMPIStringFT.getCharPtr() function shall return a character pointer to a C language string.

ERRORS
None.

EXAMPLES
None.

APPLICATION USAGE
None.

SEE ALSO
None.

CHANGE HISTORY
None.
6.3 CMPIArray Support

CMPIArrays encapsulate arrays of values of the same base types; however, some of them can be CIM NULL values. Property retrieval operations can return CMPIArray objects. MIs can produce CMPIArrays and use them in setProperty() operations. CMPIArrays are produced using a broker factory CMPIBrokerFT.newArray() function. The CMPIArrayFT.getElementAt() and CMPIArrayFT.setElementAt() functions are used to retrieve and set individual array elements.

CMPIArray support is provided by the following functions.
CMPIArrayFT.getElementAt()

NAME
CMPIArrayFT.getElementAt – gets an element value defined by its index

SYNOPSIS

CMPIData CMPIArrayFT.getElementAt(
    CMPIArray* ar,
    CMPICount index,
    CMPIStatus* rc
);

DESCRIPTION

The CMPIArrayFT.getElementAt() function shall return an element of an array, which can
be a CIM NULL value.

The ar argument points to a CMPIArray structure. The index argument specifies the
position of the element in the internal data array.

RETURN VALUE

The CMPIArrayFT.getElementAt() function shall return a CMPIData structure containing
the value of the specified element.

ERRORS
None.

EXAMPLES
None.

APPLICATION USAGE
None.

SEE ALSO
None.

CHANGE HISTORY
None.
CMPIArrayFT.getSimpleType()

NAME
CMPIArrayFT.getSimpleType – gets an element type

SYNOPSIS
CMPIType CMPIArrayFT.getSimpleType(
    CMPIArray* ar,
    CMPIStatus* rc
);

DESCRIPTION
The CMPIArrayFT.getSimpleType() function shall return the type of the elements of an Array object.
The ar argument points to a CMPIArray structure.
The rc argument points to a CMPIStatus structure used to return the service return code.

RETURN VALUE
The CMPIArrayFT.getElementAt() function shall return a CMPIType structure containing the type of the Array elements.

ERRORS
None.

EXAMPLES
None.

APPLICATION USAGE
None.

SEE ALSO
None.

CHANGE HISTORY
None.
CMPIArrayFT.getSize()

NAME
CMPIArrayFT.getSize – gets the number of elements in an Array

SYNOPSIS
CMPICount CMPIArrayFT.getSize(
    CMPIArray* ar,
    CMPIStatus* rc
);

DESCRIPTION
The CMPIArrayFT.getSize() function shall return the number of elements in an Array object.
The ar argument points to a CMPIArray structure.
The rc argument points to a CMPIStatus structure used to return the service return code.

RETURN VALUE
The CMPIArrayFT.getSize() function shall return a CMPICount structure containing the
number of the Array elements.

ERRORS
None.

EXAMPLES
None.

APPLICATION USAGE
None.

SEE ALSO
None.

CHANGE HISTORY
None.
NAME

CMPIArrayFT.setElementAt – sets an element defined by its index

SYNOPSIS

```c
CMPIStatus CMPIArrayFT.setElementAt(
    CMPIArray* ar,
    CMPICount index,
    CMPIValue* value,
    CMPIType type
);
```

DESCRIPTION

The `CMPIArrayFT.setElementAt()` function sets an element of an Array, which can be a CIM NULL value.

The `ar` argument points to a `CMPIArray` structure. The `index` argument specifies the position of the element in the internal data array. The `value` argument points to a `CMPIValue` structure containing the value to be assigned to the element. The `type` argument must either be the base type of the array or `CMPI_null`.

RETURN VALUE

The `CMPIArrayFT.setElementAt()` function shall return a `CMPIStatus` structure containing the service return status.

ERRORS

None.

EXAMPLES

None.

APPLICATION USAGE

None.

SEE ALSO

None.

CHANGE HISTORY

None.
NAME
CMPIBrokerEncFT.newArray – create a new Array object

SYNOPSIS
CMPIArray* CMPIBrokerEncFT.newArray(
    CMPIBroker* mb,
    CMPICount max,
    CMPIType type,
    CMPIStructure* rc
);

DESCRIPTION
The CMPIBrokerEncFT.newArray() function shall return a new CMPIArray object.

The mb argument points to a CMPIBroker object. The max argument specifies the maximum number of elements. The type argument specifies the element type.

The rc argument points to a CMPIStatus structure containing the service return status.

RETURN VALUE
The CMPIBrokerEncFT.newArray() function shall return a pointer to a new CMPIArray object.

ERRORS
None.

EXAMPLES
None.

APPLICATION USAGE
None.

SEE ALSO
None.

CHANGE HISTORY
None.
6.4 **CMPIEnumeration Support**

**CMPIEnumerations** are not directly created by an MI. MIs indirectly create enumerations by using successive `CMPIResultFT.returnData()` calls during execution of one of the enumerating MI functions. MIs, however, can request the MB to generate enumerations of other objects. In that case a **CMPIEnumeration** is returned.

In general, this support allows iteration through a **CMPIEnumeration**.

**CMPIEnumeration** support is provided by the following functions.
CMPIEnumerationFT.getNext()

NAME
CMPIEnumerationFT.getNext – get the next element for an Enumeration

SYNOPSIS
CMPIData CMPIEnumerationFT.getNext(
    CMPIEnumeration* en,
    CMPIStatus* rc
);

DESCRIPTION
The CMPIEnumerationFT.getNext() function shall return the next element for an Enumeration.
The en argument points to the CMPIEnumeration structure.
The rc argument points to a CMPIStatus structure containing the service return status.

RETURN VALUE
The CMPIEnumerationFT.getNext() function shall return a CMPIData structure containing the element.

ERRORS
None.

EXAMPLES
None.

APPLICATION USAGE
None.

SEE ALSO
CMPIEnumerationFT.hasNext()

CHANGE HISTORY
None.
CMPIEnumerationFT.hasNext()

NAME
CMPIEnumerationFT.hasNext – test for any elements left in an Enumeration

SYNOPSIS
CMPIBoolean CMPIEnumerationFT.hasNext(
    CMPIEnumeration* en,
    CMPIStatus* rc
);

DESCRIPTION
The CMPIEnumerationFT.hasNext() function shall test for any elements remaining in an
Enumeration.
The en argument points to the CMPIEnumeration structure.
The rc argument points to a CMPIStatus structure containing the service return status.

RETURN VALUE
The CMPIEnumerationFT.hasNext() function shall return a CMPIBoolean structure which
is true if the Enumeration has more elements to retrieve.

ERRORS
None.

EXAMPLES
None.

APPLICATION USAGE
None.

SEE ALSO
None.

CHANGE HISTORY
None.
CMPIEnumerationFT.toArray()

NAME
CMPIEnumerationFT.toArray – converts an Enumeration to an Array

SYNOPSIS
CMPIArray* CMPIEnumerationFT.toArray(
    CMPIEnumeration* en,
    CMPIStatus* rc
);

DESCRIPTION
The CMPIEnumerationFT.toArray() function converts a CMPIEnumeration structure into a CMPIArray structure.

The en argument points to the CMPIEnumeration structure.

The rc argument points to a CMPIStatus structure containing the service return status.

RETURN VALUE
The CMPIEnumerationFT.toArray() function shall return a pointer to a CMPIArray structure containing the elements from the Enumeration.

ERRORS
None.

EXAMPLES
None.

APPLICATION USAGE
None.

SEE ALSO
None.

CHANGE HISTORY
None.
6.5 CMPIInstance Support

CMPIInstance support is provided by the following functions.
NAME
CMPIBrokerEncFT.newInstance – create a new Instance object

SYNOPSIS
CMPIInstance* CMPIBrokerEncFT.newInstance(
    CMPIBroker* mb,
    CMPIObjectPath* op,
    CMPIStatus* rc
);

DESCRIPTION
The CMPIBrokerEncFT.newInstance() function shall return a new CMPIInstance object.
The mb argument points to a CMPIBroker object. The op argument points to a CMPIObjectPath structure containing the namespace and classname.
The rc argument points to a CMPIStatus structure containing the service return status.

RETURN VALUE
The CMPIBrokerEncFT.newInstance() function shall return a pointer to a new CMPIInstance object.

ERRORS
None.

EXAMPLES
None.

APPLICATION USAGE
None.

SEE ALSO
None.

CHANGE HISTORY
None.
CMPIInstanceFT.getObjectPath()

NAME
CMPIInstanceFT.getObjectPath – generate an ObjectPath from the namespace, classname, and key properties of an Instance

SYNOPSIS
CMPIObjectPath* CMPIInstanceFT.getObjectPath(
    CMPIInstance* inst,
    CMPIStatus* rc
);

DESCRIPTION
The CMPIInstanceFT.getObjectPath() function shall return an ObjectPath generated from the namespace, classname, and key properties of an Instance.

The inst argument is a pointer to the CMPIInstance structure.

The rc argument points to a CMPIStatus structure containing the service return status.

RETURN VALUE
The CMPIInstanceFT.getObjectPath() function shall return a pointer to a CMPIObjectPath structure.

ERRORS
None.

EXAMPLES
None.

APPLICATION USAGE
None.

SEE ALSO
None.

CHANGE HISTORY
None.
CMPIInstanceFT.getProperty()

NAME
CMPIInstanceFT.getProperty – get a named Property value

SYNOPSIS
CMPIData CMPIInstanceFT.getProperty(
    CMPIInstance* inst,
    char* name,
    CMPIStatus* rc
);

DESCRIPTION
The CMPIInstanceFT.getProperty() function shall return named Property value.
The inst argument is a pointer to the CMPIInstance structure.
The name argument is a string containing the property name.
The rc argument points to a CMPIStatus structure containing the service return status.

RETURN VALUE
The CMPIInstanceFT.getProperty() function shall return a CMPIData structure containing
the named Property value.

ERRORS
None.

EXAMPLES
None.

APPLICATION USAGE
None.

SEE ALSO
None.

CHANGE HISTORY
None.
NAME

CMPIInstanceFT.getPropertyAt – get a Property value defined by its index

SYNOPSIS

CMPIData CMPIInstanceFT.getPropertyAt(
    CMPIInstance* inst,
    unsigned int index,
    CMPIString** name,
    CMPIStatus* rc
);

DESCRIPTION

The CMPIInstanceFT.getPropertyAt() function shall return a Property value defined by its index.

The inst argument is a pointer to the CMPIInstance structure.

The index argument contains the index number of the Property in the internal data array.

The name argument is a pointer to a pointer to a CMPIString structure which is updated to contain the property name.

The rc argument points to a CMPIStatus structure containing the service return status.

RETURN VALUE

The CMPIInstanceFT.getPropertyAt() function shall return a CMPIData structure containing the Property value.

ERRORS

None.

EXAMPLES

None.

APPLICATION USAGE

None.

SEE ALSO

None.

CHANGE HISTORY

None.
CMPIInstanceFT.getPropertyCount()

NAME
CMPIInstanceFT.getPropertyCount – get the count of Properties contained in an Instance

SYNOPSIS
unsigned int CMPIInstanceFT.getPropertyCount(
    CMPIInstance* inst,
    CMPIStatus* rc
);

DESCRIPTION
The CMPIInstanceFT.getPropertyAt() function shall return the count of Properties contained in an Instance.
The inst argument is a pointer to the CMPIInstance structure.
The rc argument points to a CMPIStatus structure containing the service return status.

RETURN VALUE
The CMPIInstanceFT.getPropertyAt() function shall return the count of Properties in the Instance.

ERRORS
None.

EXAMPLES
None.

APPLICATION USAGE
None.

SEE ALSO
None.

CHANGE HISTORY
None.
**CMPIInstanceFT.setProperty()**

**NAME**

CMPIInstanceFT.setProperty – add/replace a named Property

**SYNOPSIS**

```c
CMPIStatus CMPIInstanceFT.setProperty(
    CMPIInstance* inst,
    char *name,
    CMPIValue* value,
    CMPIType type
);
```

**DESCRIPTION**

The **CMPIInstanceFT.setProperty()** function shall add or replace a named property within an Instance.

The **inst** argument is a pointer to the **CMPIInstance** structure.

The **name** argument is a string containing the Property name.

The **value** argument points to a **CMPIValue** structure containing the value to be assigned to the Property.

The **type** argument is a **CMPIType** structure defining the type of the value.

**RETURN VALUE**

The **CMPIInstanceFT.setProperty()** function shall return a **CMPIStatus** structure containing the service return status.

**ERRORS**

None.

**EXAMPLES**

None.

**APPLICATION USAGE**

None.

**SEE ALSO**

None.

**CHANGE HISTORY**

None.
6.6 CMPIObjectPath Support

CMPIObjectPath support is provided by the following functions.
CMPIBrokerEncFT.newObjectPath()

NAME

CMPIBrokerEncFT.newObjectPath – create a new ObjectPath object

SYNOPSIS

CMPIObjectPath* CMPIBrokerEncFT.newObjectPath(
    CMPIBroker* mb,
    char* ns,
    char* cn,
    CMPIStatus* rc
);

DESCRIPTION

The CMPIBrokerEncFT.newObjectPath() function shall return a new CMPIObjectPath object.

The mb argument points to a CMPIBroker object. The ns argument points to a string containing the namespace. The cn argument points to a string containing the classname.

The rc argument points to a CMPIStatus structure containing the service return status.

RETURN VALUE

The CMPIBrokerEncFT.newObjectPath() function shall return a pointer to a new CMPIObjectPath object.

ERRORS

None.

EXAMPLES

None.

APPLICATION USAGE

None.

SEE ALSO

None.

CHANGE HISTORY

None.
CMPObjecPathFT.addKey()

NAME
CMPObjecPathFT.addKey – add/replace a named key property

SYNOPSIS
CMPIStatus CMPObjecPathFT.addKey(
    CMPObjecPath* op,
    char* key,
    CMPIValue *value,
    CMPIType type
);

DESCRIPTION
The CMPObjecPathFT.addKey() function shall add/replace a named key property.
The op argument points to a CMPObjecPath structure.
The key argument points to a string containing the key property name.
The value argument is a pointer to a CMPIValue structure containing the value to be assigned to the key property.
The type argument is a CMPIType structure defining the type of the value to be assigned.

RETURN VALUE
The CMPObjecPathFT.addKey() function shall return a CMPIStatus structure containing the service return status.

ERRORS
None.

EXAMPLES
None.

APPLICATION USAGE
None.

SEE ALSO
None.

CHANGE HISTORY
None.
CMPIObjectPathFT.getClassName()

NAME

CMPIObjectPathFT.getClassName – get the classname component

SYNOPSIS

CMPIString* CMPIObjectPathFT.getClassName(
    CMPIObjectPath *op,
    CMPIStatus* rc
);

DESCRIPTION

The CMPIObjectPathFT.getClassName() function shall get the classname component of an ObjectPath.

The op argument points to a CMPIObjectPath structure.

The rc argument points to a CMPIStatus structure used to return the service return code.

RETURN VALUE

The CMPIObjectPathFT.getClassName() function shall return a pointer to a CMPIString structure containing the classname component.

ERRORS

None.

EXAMPLES

None.

APPLICATION USAGE

None.

SEE ALSO

None.

CHANGE HISTORY

None.
CMPIObjectPathFT.getHostname()

NAME
CMPIObjectPathFT.getHostname – get the hostname component

SYNOPSIS
CMPIString* CMPIObjectPathFT.getHostname(
    CMPIObjectPath *op,
    CMPIStatus* rc
);

DESCRIPTION
The CMPIObjectPathFT.getHostname() function shall get the hostname component of an
ObjectPath.
The op argument points to a CMPIObjectPath structure.
The rc argument points to a CMPIStatus structure used to return the service return code.

RETURN VALUE
The CMPIObjectPathFT.getHostname() function shall return a pointer to a CMPIString
structure containing the hostname component.

ERRORS
None.

EXAMPLES
None.

APPLICATION USAGE
None.

SEE ALSO
None.

CHANGE HISTORY
None.
CMPIObjectPathFT.getKey()

NAME
CMPIObjectPathFT.getKey – get a named key property value

SYNOPSIS
CMPIData CMPIObjectPathFT.getKey(
    CMPIObjectPath* op,
    char* key,
    CMPIStatus* rc
);

DESCRIPTION
The CMPIObjectPathFT.getKey() function shall get a named key property value.

The op argument points to a CMPIObjectPath structure.

The key argument points to a string containing the key name.

The rc argument points to a CMPIStatus structure used to return the service return code.

CIMXML does not prescribe transportation of precise key value types. There are only four
types that can be returned by this operation. They are part of the CMPIType definition and
are as follows:

```
#define CMPI_keyInteger   (CMPI_sint64)
#define CMPI_keyString    (CMPI_string)
#define CMPI_keyBoolean   (CMPI_boolean)
#define CMPI_keyRef       (CMPI_ref)
```

In addition, the CMPI_keyValue flag of CMPIValueState is set to indicate that the value is
emanating from a CMPIObjectPath.

RETURN VALUE
The CMPIObjectPathFT.getKey() function shall return a CMPIData structure containing
the named key value.

ERRORS
None.

EXAMPLES
None.

APPLICATION USAGE
None.

SEE ALSO
None.

CHANGE HISTORY
None.
CMPIObjectPathFT.getKeyAt()

NAME
CMPIObjectPathFT.getKeyAt – gets a key property value defined by its index

SYNOPSIS
CMPIData CMPIObjectPathFT.getKeyAt(
   CMPIObjectPath* copThis,
   unsigned int *index,
   CMPIString** name,
   CMPIStatus* rc
);

DESCRIPTION
The CMPIObjectPathFT.getKeyAt() function shall get a key property value defined by its index.

The op argument points to a CMPIObjectPath structure.

The index argument specifies the index of the key property value within the internal data array.

The name argument is a pointer to a CMPIString structure which is updated with the name of the key property.

The rc argument points to a CMPIStatus structure used to return the service return code.

CIMXML does not prescribe transportation of precise key value types. There are only four types that can be returned by this operation. They are part of the CMPIType definition and are as follows:
#define CMPI_keyInteger   (CMPI_sint64)
#define CMPI_keyString    (CMPI_string)
#define CMPI_keyBoolean   (CMPI_boolean)
#define CMPI_keyRef       (CMPI_ref)

In addition, the CMPI_keyValue flag of CMPIValueState is set to indicate that the value is emanating from a CMPIObjectPath.

RETURN VALUE
The CMPIObjectPathFT.getKeyAt() function shall return a CMPIData structure containing the named key value.

ERRORS
None.

EXAMPLES
None.

APPLICATION USAGE
None.

SEE ALSO
None.

CHANGE HISTORY
None.
CMPIObjectPathFT.getKeyCount()  

NAME  
CMPIObjectPathFT.getKeyCount – get the number of key properties

SYNOPSIS  
unsigned int CMPIObjectPathFT.getKeyCount(
   CMPIObjectPath* op,
   CMPIStatus* rc
);

DESCRIPTION  
The CMPIObjectPathFT.getKeyCount() function shall get the the number of key properties contained in an ObjectPath. The op argument points to a CMPIObjectPath structure. The rc argument points to a CMPIStatus structure used to return the service return code.

RETURN VALUE  
The CMPIObjectPathFT.getKeyCount() function shall return the number of key properties.

ERRORS  
None.

EXAMPLES  
None.

APPLICATION USAGE  
None.

SEE ALSO  
None.

CHANGE HISTORY  
None.
CMPIObjectPathFT.getNameSpace()

NAME

CMPIObjectPathFT.getNameSpace – get the namespace component

SYNOPSIS

CMPIString* CMPIObjectPathFT.getNameSpace(
    CMPIObjectPath* op,
    CMPIStatus* rc
);  

DESCRIPTION

The CMPIObjectPathFT.getNameSpace() function shall get the namespace component of an ObjectPath.

The op argument points to a CMPIObjectPath structure.

The rc argument points to a CMPIStatus structure used to return the service return code.

RETURN VALUE

The CMPIObjectPathFT.getNameSpace() function shall return a pointer to a CMPIString structure containing the namespace component.

ERRORS

None.

EXAMPLES

None.

APPLICATION USAGE

None.

SEE ALSO

None.

CHANGE HISTORY

None.
CMPIObjectPathFT::setHostname()

NAME
CMPIObjectPathFT::setHostname – set/replace the hostname component

SYNOPSIS
CMPIStatus CMPIObjectPathFT::setNameSpace(
    CMPIObjectPath* op,
    char *hn
);

DESCRIPTION
The CMPIObjectPathFT::getNameSpace() function shall get the namespace component of an
ObjectPath.
The op argument points to a CMPIObjectPath structure.
The hn argument is a string containing the new hostname.

RETURN VALUE
The CMPIObjectPathFT::setHostname() function shall return a CMPIStatus structure
containing the service return status.

ERRORS
None.

EXAMPLES
None.

APPLICATION USAGE
None.

SEE ALSO
None.

CHANGE HISTORY
None.
CMPIObjectPathFT.setNameSpace()

NAME
CMPIObjectPathFT.setNameSpace – set the namespace component

SYNOPSIS
CMPIStatus CMPIObjectPathFT.setNameSpace(
    CMPIObjectPath* op,
    char *ns
);

DESCRIPTION
The CMPIObjectPathFT.setNameSpace() function shall get the namespace component of an
ObjectPath.
The op argument points to a CMPIObjectPath structure.
The ns argument is a string containing the new namespace.

RETURN VALUE
The CMPIObjectPathFT.setNameSpace() function shall return a CMPIStatus structure
containing the service return status.

ERRORS
None.

EXAMPLES
None.

APPLICATION USAGE
None.

SEE ALSO
None.

CHANGE HISTORY
None.
CMPIObjectPathFT.setNameSpaceFromObjectPath()

NAME
CMPIObjectPathFT.setNameSpaceFromObjectPath – set/replace the namespace and classname components from an ObjectPath

SYNOPSIS
CMPIStatus CMPIObjectPathFT.setNameSpaceFromObjectPath(
    CMPIObjectPath* op,
    CMPIObjectPath* src
);

DESCRIPTION
The CMPIObjectPathFT.setNameSpaceFromObjectPath() function shall set/replace the namespace and classname components from an ObjectPath.

The op argument points to a CMPIObjectPath structure to be modified.

The src argument points to a CMPIObjectPath structure used as the source for the new namespace and classname components.

RETURN VALUE
The CMPIObjectPathFT.setNameSpaceFromObjectPath() function shall return a CMPIStatus structure containing the service return status.

ERRORS
None.

EXAMPLES
None.

APPLICATION USAGE
None.

SEE ALSO
None.

CHANGE HISTORY
None.
CMPIObjectPathFT.setHostAndNameSpaceFromObjectPath()

NAME
CMPIObjectPathFT.setHostAndNameSpaceFromObjectPath – set/replace the hostname, namespace, and classname components from an ObjectPath

SYNOPSIS
CMPIStatus CMPIObjectPathFT.setHostAndNameSpaceFromObjectPath(
    CMPIObjectPath* op,
    CMPIObjectPath* src
);

DESCRIPTION
The CMPIObjectPathFT.setHostAndNameSpaceFromObjectPath() function shall set/replace the namespace and classname components from an ObjectPath.

The op argument points to a CMPIObjectPath structure to be modified.

The src argument points to a CMPIObjectPath structure used as the source for the new hostname, namespace, and classname components.

RETURN VALUE
The CMPIObjectPathFT.setHostAndNameSpaceFromObjectPath() function shall return a CMPIStatus structure containing the service return status.

ERRORS
None.

EXAMPLES
None.

APPLICATION USAGE
None.

SEE ALSO
None.

CHANGE HISTORY
None.
6.7 CMPIArgs Support

CMPIArgs is a container used to capture arguments for `invokeMethod()` functions.

CMPIArgs support is provided by the following functions:
CMPIArgsFT.addArg()

NAME
CMPIArgsFT.addArg – Adds/replaces a named argument

SYNOPSIS

```c
CMPIStatus CMPIArgsFT.addArg(
    CMPIArgs* as,
    char *name,
    CMPIValue *value,
    CMPIType type
);
```

DESCRIPTION
The `CMPIArgsFT.addArg()` function shall add or replace a named argument within an Args structure.

The `as` argument points to an `CMPIArgs` structure. The `name` argument is a string containing the name of the argument to be added or replaced. The `value` argument points to a `CMPIValue` structure containing to be assigned to the argument. The `type` argument identifies the type of the argument. The following types are supported:

- `CMPIBoolean`
- `CMPIChar16`
- `CMPIUint8`
- `CMPIUint16`
- `CMPIUint32`
- `CMPIUint64`
- `CMPIInt8`
- `CMPIInt16`
- `CMPIInt32`
- `CMPIInt64`
- `CMPIReal32`
- `CMPIReal64`
- `CMPIString*`
- `CMPIObjectPath*`

RETURN VALUE
The `CMPIArgsFT.addArg()` function shall return a `CMPIStatus` structure containing the service return status.

ERRORS
None.

EXAMPLES
None.

APPLICATION USAGE
None.

SEE ALSO
None.

CHANGE HISTORY
None.
CMPIArgsFT.getArg()

NAME
CMPIArgsFT.getArg – gets a named argument value

SYNOPSIS
CMPIData CMPIArgsFT.getArg(
    CMPIArgs* as,
    char *name,
    CMPIStatus* rc
);

DESCRIPTION
The CMPIArgsFT.getArg() function shall return the value of a named argument.
The as argument points to a CMPIArgs structure. The name argument is a string containing the name of the argument to be retrieved.
The rc argument points to a CMPIStatus structure used to return the service return code.

RETURN VALUE
The CMPIArgsFT.getArg() function shall return a CMPIData structure containing the value of the named argument.

ERRORS
None.

EXAMPLES
None.

APPLICATION USAGE
None.

SEE ALSO
None.

CHANGE HISTORY
None.
CMPIArgsFT.getArgAt()

NAME
CMPIArgsFT.getArgAt – gets an argument value defined by its index

SYNOPSIS
CMPIData CMPIArgsFT.getArgAt(
    CMPIArgs* as,
    unsigned int index,
    CMPIString** name,
    CMPIStatus* rc
);

DESCRIPTION
The CMPIArgsFT.getArgAt() function shall return the value of an argument defined by its index in the internal data array.

The as argument points to a CMPIArgs structure. The index argument specifies the index in the internal data array.

The name argument points to a pointer to a CMPIString structure used to return the argument name.

The rc argument points to a CMPIStatus structure used to return the service return code.

RETURN VALUE
The CMPIArgsFT.getArgAt() function shall return a CMPIData structure containing the value of the specified argument.

ERRORS
None.

EXAMPLES
None.

APPLICATION USAGE
None.

SEE ALSO
None.

CHANGE HISTORY
None.
CMPIArgsFT.getArgCount()

NAME
CMPIArgsFT.getArgCount – gets the number of arguments contained in an Args structure

SYNOPSIS
unsigned int CMPIArgsFT.getArgCount(
    CMPIArgs* as,
    CMPIStatus* rc
);

DESCRIPTION
The CMPIArgsFT.getArgCount() function gets the number of arguments contained in an
Args structure.
The as argument points to a CMPIArgs structure.
The rc argument points to a CMPIStatus structure used to return the service return code.

RETURN VALUE
The CMPIArgsFT.getArgCount() function shall return the number of arguments contained in
the Args structure.

ERRORS
None.

EXAMPLES
None.

APPLICATION USAGE
None.

SEE ALSO
None.

CHANGE HISTORY
None.
CMPIBrokerEncFT.newArgs()

NAME
CMPIBrokerEncFT.newArgs – create a new Args object

SYNOPSIS

CMPIArgs* CMPIBrokerEncFT.newArgs(
    CMPIBroker* mb,
    CMPIStatus* rc
);

DESCRIPTION
The CMPIBrokerEncFT.newArgs() function shall return a new CMPIArgs object.
The mb argument points to a CMPIBroker object.
The rc argument points to a CMPIStatus structure containing the service return status.

RETURN VALUE
The CMPIBrokerEncFT.newArgs() function shall return a pointer to a new CMPIArgs object.

ERRORS
None.

EXAMPLES
None.

APPLICATION USAGE
None.

SEE ALSO
None.

CHANGE HISTORY
None.
In order to be platform-independent, **CMPIDateTime** is implemented as an encapsulated type. It supports two ways of expressing time in binary form using a *long long* C type and as a **CMPIString** using CIM datetime fixed string format. Time can be set by asking for the current time of day or by using any of the two formats defined before as input. **CMPIDateTime** supports the UTC notion of interval *versus* time of date values.

**CMPIDateTime** support is provided by the following functions.
**NAME**

CMPIBrokerEncFT.newDateTime – create a new DateTime object initialized to the current date and time

**SYNOPSIS**

```c
CMPIDateTime* CMPIBrokerEncFT.newDateTime(
    CMPIBroker* mb,
    CMPIStatus* rc
);
```

**DESCRIPTION**

The `CMPIBrokerEncFT.newDateTime()` function shall return a new `CMPIDateTime` object initialized with the current date and time.

The `mb` argument points to a `CMPIBroker` object.

The `rc` argument points to a `CMPIStatus` structure containing the service return status.

**RETURN VALUE**

The `CMPIBrokerEncFT.newDateTime()` function shall return a pointer to a new `CMPIDateTime` object.

**ERRORS**

None.

**EXAMPLES**

None.

**APPLICATION USAGE**

None.

**SEE ALSO**

None.

**CHANGE HISTORY**

None.
**NAME**

CMPIBrokerEncFT.newDateTimeFromBinary – create a new DateTime object initialized to a specified value

**SYNOPSIS**

```c
CMPIDateTime* CMPIBrokerEncFT.newDateTimeFromBinary(
    CMPIBroker* mb,
    CMPIUint64 binTime,
    CMPIBoolean interval,
    CMPIStatus* rc
);
```

**DESCRIPTION**

The `CMPIBrokerEncFT.newDateTimeFromBinary()` function shall return a new `CMPIDateTime` object initialized with the specified date and time.

The `binTime` argument contains a Date/Time value expressed as a 64-bit unsigned integer in microseconds starting since 00:00:00 GMT, Jan 1, 1970.

The `mb` argument points to a `CMPIBroker` object.

The `interval` argument, if true, indicates that the `binTime` argument is considered to be a time interval in microseconds.

The `rc` argument points to a `CMPIStatus` structure containing the service return status.

{Ed: The original text seemed clearer. The definitions of `binTime` and `interval` are somewhat unclear. If `interval` is true, is `binTime` still measured from the epoch?}

**RETURN VALUE**

The `CMPIBrokerEncFT.newDateTimeFromBinary()` function shall return a pointer to a new `CMPIDateTime` object.

**ERRORS**

None.

**EXAMPLES**

None.

**APPLICATION USAGE**

None.

**SEE ALSO**

None.

**CHANGE HISTORY**

None.
CMPIBrokerEncFT.newDateTimeFromChars()

NAME
CMPIBrokerEncFT.newDateTimeFromChars – create a new DateTime object initialized to a specified value

SYNOPSIS
CMPIDateTime* CMPIBrokerEncFT.newDateTimeFromChars(
    CMPIBroker* mb,
    char* utcTime,
    CMPIStatus* rc
);

DESCRIPTION
The `CMPIBrokerEncFT.newDateTimeFromChars()` function shall return a new `CMPIDateTime` object initialized with the specified date and time.

The `utcTime` argument contains a Date/Time value expressed in UTC format. {Ed: Need a reference here.}

The `rc` argument points to a `CMPIStatus` structure containing the service return status.

RETURN VALUE
The `CMPIBrokerEncFT.newDateTimeFromChars()` function shall return a pointer to a new `CMPIDateTime` object.

ERRORS
None.

EXAMPLES
None.

APPLICATION USAGE
None.

SEE ALSO
None.

CHANGE HISTORY
None.
CMPIDateTimeFT.getBinaryFormat()

NAME

CMPIDateTimeFT.getBinaryFormat – get a DateTime value in binary format

SYNOPSIS

CMPIUint64 CMPIDateTimeFT.getBinaryFormat(
    CMPIDateTime* dt,
    CMPIStatus* rc
);

DESCRIPTION

The CMPIDateTimeFT.getBinaryFormat() function shall return the value of a DateTime object as a 64-bit unsigned integer in microseconds starting since 00:00:00 GMT, Jan 1, 1970, or as an interval, depending on how the CMPIDateTime object was created.

The dt argument is a pointer the CMPIDateTime structure.

The rc argument points to a CMPIStatus structure containing the service return status.

RETURN VALUE

The CMPIDateTimeFT.getBinaryFormat() function shall return a CMPIUint64 structure containing the DateTime value in binary format.

ERRORS

None.

EXAMPLES

None.

APPLICATION USAGE

None.

SEE ALSO

None.

CHANGE HISTORY

None.
CMPIDateTimeFT.getStringFormat()

NAME
CMPIDateTimeFT.getStringFormat – get a DateTime value in UTC string format

SYNOPSIS
CMPIString* CMPIDateTimeFT.getStringFormat(
        CMPIDateTime* dt,
        CMPIStatus* rc
    );

DESCRIPTION
The CMPIDateTimeFT.getStringFormat() function shall return the value of a DateTime as a string formatted using the UTC fixed string format: yyymmddhhmmss.mmmmsutc (year, month, day, hour, minute, second, microseconds, and signed UTC offset). Interval times are indicated by :000 for the signed UTC offset.

The dt argument is a pointer the CMPIDateTime structure.

The rc argument points to a CMPIStatus structure containing the service return status.

RETURN VALUE
The CMPIDateTimeFT.getStringFormat() function shall return a pointer to a CMPIString structure containing the DateTime value in UTC string format.

ERRORS
None.

EXAMPLES
None.

APPLICATION USAGE
None.

SEE ALSO
None.

CHANGE HISTORY
None.
CMPIDateTimeFT.isInterval()

NAME
CMPIDateTimeFT.isInterval – test whether a DateTime object is an interval value

SYNOPSIS
CMPIBoolean CMPIDateTimeFT.isInterval(
    CMPIDateTime* dt,
    CMPIStatus* rc
);

DESCRIPTION
The CMPIDateTimeFT.isInterval() function shall test whether a DateTime object is an interval value.

The dt argument is a pointer the CMPIDateTime structure.

The rc argument points to a CMPIStatus structure containing the service return status.

RETURN VALUE
The CMPIDateTimeFT.isInterval() function shall return a CMPIBoolean structure with the value true if the DateTime object is an interval value.

ERRORS
None.

EXAMPLES
None.

APPLICATION USAGE
None.

SEE ALSO
None.

CHANGE HISTORY
None.
6.9 CMPISelectExp Support

CMPISelectExp support is provided by the following functions.
NAME
CMPIBrokerEncFT.newSelectExp – create a new SelectExp object

SYNOPSIS
CMPISelectExp* CMPIBrokerEncFT.newSelectExp(
    CMPIBroker* mb,
    char* query,
    char* lang,
    CMPIArray** projection,
    CMPIStatus* rc
);

DESCRIPTION
The CMPIBrokerEncFT.newSelectExp() function shall return a new CMPISelectExp object.

The mb argument points to a CMPIBroker object. The query argument is a pointer to a string containing the select expression. The lang argument is a pointer to a string containing the query language.

The projection argument is a pointer to a pointer to a CMPIArray structure that is updated with the projection specification.

The rc argument points to a CMPIStatus structure containing the service return status.

RETURN VALUE
The CMPIBrokerEncFT.newSelectExp() function shall return a pointer to a new CMPISelectExp object.

ERRORS
None.

EXAMPLES
None.

APPLICATION USAGE
None.

SEE ALSO
None.

CHANGE HISTORY
None.
CMPISelectExpFT.evaluate()

NAME
CMPISelectExpFT.evaluate – evaluate an Instance using a Select Expression

SYNOPSIS
CMPIBoolean CMPISelectExpFT.evaluate(
    CMPISelectExp* se,
    CMPIInstance* inst,
    CMPIStatus* rc
);

DESCRIPTION
The CMPISelectExpFT.evaluate() function shall evaluate an Instance using a Select Expression.
The se argument is a pointer to a CMPISelectExp structure containing the Select Expression.
The inst argument is a pointer to a CMPIInstance structure containing the Instance to be evaluated.
The rc argument points to a CMPIStatus structure containing the service return status.

RETURN VALUE
The CMPISelectExpFT.evaluate() function shall return a CMPIBoolean structure indicating the result of the evaluation.

ERRORS
None.

EXAMPLES
None.

APPLICATION USAGE
None.

SEE ALSO
None.

CHANGE HISTORY
None.
NAME

CMPISelectExpFT.getCOD – return a selection as a conjunction of disjunctions

SYNOPSIS

CMPISelectCond* CMPISelectExpFT.getCOD(
    CMPISelectExp* se,
    CMPIStatus* rc
);

DESCRIPTION

The `CMPISelectExpFT.getCOD` function shall return the selection as a conjunction of disjunctions. This function transforms the filter’s `where` clause into a canonical Conjunction of Disjunctions form (AND or OR’ed comparison expressions). This enables handling of the `where` expression more easily than using a tree form.

The `se` argument is a pointer to a `CMPISelectExp` structure containing the Select Expression.

The `rc` argument points to a `CMPIStatus` structure containing the service return status.

Support for this function is optional. Availability of this support is indicated by the `CMPI_MB_Supports_QueryNormalization` flag in `CMPIBrokerFT.classification`.

RETURN VALUE

The `CMPISelectExpFT.getCOD` function shall return a pointer to a `CMPISelectCond` structure containing the transformed selection.

ERRORS

None.

EXAMPLES

None.

APPLICATION USAGE

None.

SEE ALSO

None.

CHANGE HISTORY

None.
CMPISelectExpFT.getDOC()

NAME
CMPISelectExpFT.getDOC – return a selection as a disjunction of conjunctions

SYNOPSIS
CMPISelectCond* CMPISelectExpFT.getDOC(
    CMPISelectExp* se,
    CMPIStatus* rc
);

DESCRIPTION
The CMPISelectExpFT.getDOC() function shall return the selection as a conjunction of disjunctions. This function transforms the filter’s where clause into a canonical Disjunction of Conjunctions form (OR or AND’ed comparison expressions). This enables handling of the where expression more easily than using a tree form.

The se argument is a pointer to a CMPISelectExp structure containing the Select Expression.

The rc argument points to a CMPIStatus structure containing the service return status.

Support for this function is optional. Availability of this support is indicated by the CMPI_MB_Supports_QueryNormalization flag in CMPIBrokerFT.classification.

RETURN VALUE
The CMPISelectExpFT.getDOC() function shall return a pointer to a CMPISelectCond structure containing the transformed selection.

ERRORS
None.

EXAMPLES
None.

APPLICATION USAGE
None.

SEE ALSO
None.

CHANGE HISTORY
None.
CMPISelectExpFT.getString()

NAME
CMPISelectExpFT.getString – return a Select Expression in string format

SYNOPSIS
CMPIString* CMPISelectExpFT.getString(
    CMPISelectExp* se,
    CMPIStatus* rc
);

DESCRIPTION
The CMPISelectExpFT.getString() function shall return the selection as a string.

The se argument is a pointer to a CMPISelectExp structure containing the Select Expression.

The rc argument points to a CMPIStatus structure containing the service return status.

RETURN VALUE
The CMPISelectExpFT.getString() function shall return a pointer to a CMPIString structure containing the Select Expression in string format.

ERRORS
None.

EXAMPLES
None.

APPLICATION USAGE
None.

SEE ALSO
None.

CHANGE HISTORY
None.
6.10 CMPISelectCond Support

CMPISelectCond support is optional. Availability of this support is indicated by the CMPI_MB_Supports_QueryNormalization flag in CMPIBrokerFT.classification().

CMPISelectCondSupport is provided by the following functions.
CMPISelectCondFT.getCountAndType()

NAME
CMPISelectCondFT.getCountAndType – return the number and type of subconditions that
are part of a SelectCond object

SYNOPSIS
CMPICount CMPISelectCondFT.getCountAndType(
    CMPISelectCond* sc,
    int *type,
    CMPIStatus* rc
); 

DESCRIPTION
The CMPISelectCondFT.getCountAndType() function shall return the count and type of the
subconditions that are part of a SelectCond object.

The sc argument points to a CMPISelectCond structure.

The type argument is a pointer to an integer which is updated with the SelectCond type. A
value of 0 indicates a DOC type, and a value of 1 indicates a COD type. If type is NULL no
type information is returned.

The rc argument points to a CMPIStatus structure containing the service return status.

RETURN VALUE
The CMPISelectCondFT.getCountAndType() function shall return a pointer to a
CMPICount structure containing the number of subconditions

ERRORS
None.

EXAMPLES
None.

APPLICATION USAGE
None.

SEE ALSO
None.

CHANGE HISTORY
None.
CMPISelectCondFT.getExpAt()

NAME
CMPISelectCondFT.getExpAt – return a SubCond element based on its index

SYNOPSIS
CMPSUBCOND* CMPISelectCondFT.getExpAt(
    CMPISelectCond* sc,
    unsigned int index,
    CMPIStatus* rc
);

DESCRIPTION
The CMPISelectCondFT.getExpAt() function shall return the count and type of the
subconditions that are part of a SelectCond object.

The sc argument points to a CMPISelectCond structure.

The index argument is an integer specifying the position of the SubCOnd element in the
internal data array.

The rc argument points to a CMPIStatus structure containing the service return status.

RETURN VALUE
The CMPISelectCondFT.getExpAt() function shall return a pointer to a CMPSUBCOND
structure containing the SubCond element.

ERRORS
None.

EXAMPLES
None.

APPLICATION USAGE
None.

SEE ALSO
None.

CHANGE HISTORY
None.
6.11 **CMPICond Support**

CMPICond support is optional. Availability of this support is indicated by the CMPI_MB_Supports_QueryNormalization flag in CMPIBrokerFT.classification.

CMPICond support is provided by the following functions.
CMPISubCondFT.getCount()

NAME

CMPISubCondFT.getCount – return the number of predicates that are part of a subcondition.

SYNOPSIS

CMPICount CMPISubCondFT.getCount(
    CMPISubCondFT* sc,
    CMPISubCondFT* rc
);

DESCRIPTION

The CMPISubCondFT.getCount() function shall return the number of predicates that are part
of a subcondition.

The sc argument is a pointer to a CMPISubCondFT structure containing the subcondition.

The rc argument points to a CMPISubCondFT structure containing the service return status.

RETURN VALUE

The CMPISubCondFT.getCount() function shall return a pointer to a CMPICount structure
containing the number of predicates.

ERRORS

None.

EXAMPLES

None.

APPLICATION USAGE

None.

SEE ALSO

None.

CHANGE HISTORY

None.
CMPISubCondFT.getPredicate()

NAME

CMPISubCondFT.getPredicate – return a named predicate element

SYNOPSIS

```c
CMPIPredicate* CMPISubCondFT.getPredicate(
   CMPISubCond* sc,
    char* name,
    CMPIStatus* rc
);
```

DESCRIPTION

The CMPISubCondFT.getPredicate() function shall return the a named predicate from a
subcondition. This function treats the list of predicates as a list of named entries and enables
getting a particular predicate by name.

The sc argument is a pointer to a CMPISubCondFT structure containing the subcondition.

The name argument is a string containing the predicate name. The name is the left-hand side
of the predicate.

The rc argument points to a CMPIStatus structure containing the service return status.

RETURN VALUE

The CMPISubCondFT.getPredicate() function shall return a pointer to a CMPIPredicate
structure containing the named predicate.

ERRORS

None.

EXAMPLES

None.

APPLICATION USAGE

None.

SEE ALSO

None.

CHANGE HISTORY

None.
CMPISubCondFT.getPredicateAt()

NAME

CMPISubCondFT.getPredicateAt – return a predicate based on its index

SYNOPSIS

CMPIPredicate* CMPISubCondFT.getPredicateAt(
    CMPISubCond* sc,
    int index,
    CMPIStatus* rc
);

DESCRIPTION

The CMPISubCondFT.getPredicateAt() function shall return the a predicate from a subcondition based on its index. The sc argument is a pointer to a CMPISubCondFT structure containing the subcondition.

The index argument is an integer specifying the index of the predicate in the internal data array.

The rc argument points to a CMPIStatus structure containing the service return status.

The name is the left-hand side of the predicate.

RETURN VALUE

The CMPISubCondFT.getPredicate() function shall return a pointer to a CMPIPredicate structure containing the predicate.

ERRORS

None.

EXAMPLES

None.

APPLICATION USAGE

None.

SEE ALSO

None.

CHANGE HISTORY

None.
6.12 CMPIPredicate Support

CMPIPredicate support is optional. Availability of this support is indicated by the
CMPI_MB_Supports_QueryNormalization flag in CMPIBrokerFT.classification.

CMPIPredicate support is provided by the following functions.
CMPIPredicateFT.evaluate()

NAME
CMPIPredicateFT.evaluate – evaluate a predicate using a specific value

SYNOPSIS

```c
int CMPIPredicateFT.evaluate(
    CMPIPredicate* pr,
    CMPIValue* value,
    CMPIType type,
    CMPIStatus* rc
);
```

DESCRIPTION
The `CMPIPredicateFT.evaluate()` function shall evaluate a predicate using a specific value.

The `pr` argument points to a `CMPIPredicate` structure to be evaluated.

{Ed: The web documentation had the `type` argument duplicated. I have deleted one copy. Is this correct?}

The `value` argument points to a `CMPIValue` structure containing the value to be used to evaluate the predicate.

The `type` argument is a `CMPIType` structure defining the type of the value.

The `rc` argument points to a `CMPIStatus` structure containing the service return status.

RETURN VALUE
The `CMPIPredicateFT.evaluate()` function shall return an integer containing the evaluation result.

{Ed: Are the only results `TRUE` or `FALSE`? If so, should this be returning a `CMPIBoolean`.}

ERRORS
None.

EXAMPLES
None.

APPLICATION USAGE
None.

SEE ALSO
None.

CHANGE HISTORY
None.
CMPIPredicateFT.getData()  

NAME  
CMPIPredicateFT.getData – get the predicate components  

SYNOPSIS  
CMPIStatus CMPIPredicateFT.getData(
   CMPIPredicate* pr,
   CMPIType* type,
   CMPIPredOp *op,
   CMPIString** lhs,
   CMPIString** rhs
);

DESCRIPTION  
The CMPIPredicateFT.getData() function shall get the predicate components.  
The pr argument points to a CMPIPredicate structure to be evaluated.  
The type argument points to a CMPIType structure defining the property type.  
The op argument points to a CMPIPredOp structure containing the predicate operation.  
The lhs argument points to a CMPIString structure that is updated with the left hand side of  
the predicate.  
The rhs argument points to a CMPIString structure that is updated with the right hand side  
of the predicate.

RETURN VALUE  
The CMPIPredicateFT.getData() function shall return a CMPIStatus structure containing  
the service return status.

ERRORS  
None.

EXAMPLES  
None.

APPLICATION USAGE  
None.

SEE ALSO  
None.

CHANGE HISTORY  
None.
Qualifier Support

Qualifier support is an optional feature of CMPI. Availability of this support is indicated by the `CMPI_MB_Supports_Qualifier` flag in `CMPIBrokerFT.classification`.

Qualifier support is a subset of full schema support. It entails read-only access to CIM type qualifiers of class definitions and its components. Qualifier support is an extension to the `CMPIObjectPath` encapsulated object. The model path portion to the `CMPIObjectPath` is used to identify the object.

Qualifier support is provided by the following functions.
CMPIObjectPathFT.getClassQualifier()

NAME
CMPIObjectPathFT.getClassQualifier – get the class qualifier value

SYNOPSIS
CMPIData CMPIObjectPathFT.getClassQualifier(
  CMPIObjectPath* op,
  char* qName,
  CMPIStatus *rc
);

DESCRIPTION
The CMPIObjectPathFT.getClassQualifier() function shall return the class qualifier value of
an ObjectPath.

The op argument points to a CMPIObjectPath structure.
The qName argument is a string containing the qualifier name.
The rc argument points to a CMPIStatus structure used to return the service return code.

RETURN VALUE
The CMPIObjectPathFT.getClassQualifier() function shall return a CMPIData structure
containing the qualifier value.

ERRORS
None.

EXAMPLES
None.

APPLICATION USAGE
None.

SEE ALSO
None.

CHANGE HISTORY
None.
CMPIObjectPathFT.getMethodQualifier()

NAME
CMPIObjectPathFT.getMethodQualifier – get the method qualifier value

SYNOPSIS
CMPIData CMPIObjectPathFT.getMethodQualifier(
    CMPIObjectPath* op,
    char* mName,
    char* qName,
    CMPIStatus *rc
);

DESCRIPTION
The CMPIObjectPathFT.getMethodQualifier() function shall return the class qualifier value of an ObjectPath.
The op argument points to a CMPIObjectPath structure.
The mName argument is a string containing the method name.
The qName argument is a string containing the qualifier name.
The rc argument points to a CMPIStatus structure used to return the service return code.

RETURN VALUE
The CMPIObjectPathFT.getMethodQualifier() function shall return a CMPIData structure containing the qualifier value.

ERRORS
None.

EXAMPLES
None.

APPLICATION USAGE
None.

SEE ALSO
None.

CHANGE HISTORY
None.
CMPIObjectPathFT.getParameterQualifier()

NAME
CMPIObjectPathFT.getParameterQualifier – get the method parameter qualifier value

SYNOPSIS
CMPIData CMPIObjectPathFT.getParameterQualifier(
    CMPIObjectPath* op,
    char* mName,
    char* pName,
    char* qName,
    CMPIStatus *rc
);  

DESCRIPTION
The CMPIObjectPathFT.getParameterQualifier() function shall return the class qualifier value of an ObjectPath.

The op argument points to a CMPIObjectPath structure.

The mName argument is a string containing the method name.

The pName argument is a string containing the parameter name.

The qName argument is a string containing the qualifier name.

The rc argument points to a CMPIStatus structure used to return the service return code.

RETURN VALUE
The CMPIObjectPathFT.getParameterQualifier() function shall return a CMPIData structure containing the qualifier value.

ERRORS
None.

EXAMPLES
None.

APPLICATION USAGE
None.

SEE ALSO
None.

CHANGE HISTORY
None.
CMPIObjectPathFT.getPropertyQualifier()

NAME
CMPIObjectPathFT.getPropertyQualifier – ????

SYNOPSIS
CMPIData CMPIObjectPathFT.getPropertyQualifier(
    CMPIObjectPath* op,
    char* pName,
    char* qName,
    CMPIStatus *rc
);

DESCRIPTION
The CMPIObjectPathFT.getPropertyQualifier() function shall return the class qualifier value of an ObjectPath.

The op argument points to a CMPIObjectPath structure.

The pName argument is a string containing the property name.

The qName argument is a string containing the qualifier name.

The rc argument points to a CMPIStatus structure used to return the service return code.

RETURN VALUE
The CMPIObjectPathFT.getPropertyQualifier() function shall return a CMPIData structure containing the qualifier value.

ERRORS
None.

EXAMPLES
None.

APPLICATION USAGE
None.

SEE ALSO
None.

CHANGE HISTORY
None.
Schema support is an optional feature of CMPI. Availability of this support is indicated by the `CMPI_MB_Supports_Schema` flag in `CMPIBrokerFT.classification()`.

Schema support enables full support for creation, update, and access to class definitions and its components.

TO BE COMPLETED.
9 MB Services

Management Broker (MB) Services are part of CMPI support. In general, the services correspond with most of the services either available via a CIMClient interface or via various CIMOMProviderHandle implementations.

9.1 Minimal Services

In theory, MBs do not need to support some or all of the services; after all, it is acceptable for CIM clients to get a [CIM_ERR_NOT_SUPPORTED] exception for client calls. Still, writing Management Instrumentation (MI) in such unpredictable environments is a hassle.

We should consider making a few services mandatory. The following services have proven to be useful in the past and should be looked at more closely:

- Enumerate Instances/InstanceNames
- Enumerate Associators/AssociatorNames
- Create/Get/Set/DeleteInstance
- InvokeMethod
- DeliverIndication
- Get/SetProperty

There is some redundancy in these services, meaning with a smaller list most services are still available somehow. When we ignore DeliverIndication, then Enumerate InstanceNames() and GetInstance() should be the absolute minimal set.

9.2 MB Classification and Optional Feature Support

MBs not supporting these minimal services are called Class 0 MBs. MBs supporting the minimal set are called Class 1 MBs. The next higher level is Class 2, supporting all services. Notice that individual MIs (but not a Class 2 MB itself) still can reject a Get/SetProperty operation.

MIs can query the MB classification and decide to refuse to operate in a less-than-needed environment, or curtail its functionality. This can be done by the MI factory by returning [CMPI_RC_ERR_NOT_SUPPORTED].

---

7 This list is almost as complete as the usual CIMClient repertoire, except for Qualifier services.
8 Could be done using Get/SetInstance.
In addition, the MB exposes which optional features of CMPI are supported. The following are valid values for `CMPIBrokerFT.classification()`:

```c
#define CMPI_MB_Class_0 0x00000001
#define CMPI_MB_Class_1 0x00000003
#define CMPI_MB_Class_2 0x00000007
#define CMPI_MB_Supports_PropertyMI          0x00000100
#define CMPI_MB_Supports_IndicationMI         0x00000200
#define CMPI_MB_Supports_IndicationPolling    0x00000400
#define CMPI_MB_Supports_QueryNormalization   0x00000800
#define CMPI_MB_Supports_Qualifier            0x00001000
#define CMPI_MB_Supports_Schema               0x00003000
```

### 9.3 Class 0 Services

Class 0 services are provided by the following functions.
CMPIBrokerFT.deliverIndication()

NAME

CMPIBrokerFT.deliverIndication – request delivery of an Indication

SYNOPSIS

CMPIStatus CMPIBrokerFT.deliverIndication(
    CMPIBroker* mb,
    CMPIContext* ctx,
    char* ns,
    CMPIInstance* ind
);

DESCRIPTION

The `CMPIBrokerFT.deliverIndication()` requests the delivery of an Indication. The CIMOM will locate pertinent subscribers and notify them about the event.

The `mb` argument points to a `CMPIBroker` structure. The `ctx` argument points to the context object.

The `ns` argument is a pointer to the Namespace.

The `ind` argument is a pointer to a `CMPIInstance` object containing the Indication.

RETURN VALUE

The `CMPIBrokerFT.deliverIndication()` function shall return a `CMPIStatus` structure containing the service return status.

ERRORS

None.

EXAMPLES

None.

APPLICATION USAGE

None.

SEE ALSO

None.

CHANGE HISTORY

None.
9.4 Class 1 Services

Class 1 services are provided by the following functions.
**CMPIBrokerFT.enumInstanceNames()**

**NAME**

CMPIBrokerFT.enumInstanceNames – enumerate the Instance Names of the class (and subclasses) defined by an ObjectPath

**SYNOPSIS**

```c
CMPIEnumeration* CMPIBrokerFT.enumInstanceNames(
    CMPIBroker* mb,
    CMPIContext *ctx,
    CMPIObjectPath* op,
    CMPIStatus* rc
);
```

**DESCRIPTION**

The `CMPIBrokerFT.enumerateInstanceNames()` function shall enumerate the Instance Names of the class (and subclasses) defined by the specified ObjectPath. Instance structure can be controled using the `CMPIInvocationFlags` entry in `ctx`.

The `mb` argument points to a `CMPIBroker` structure. The `ctx` argument points to the context object, and the `op` argument points to the source ObjectPath containing namespace and classname elements.

The `rc` argument points to a `CMPIStatus` structure used to return the service return code.

**RETURN VALUE**

The `CMPIBrokerFT.enumInstanceNames()` function shall return a pointer to a `CMPIEnumeration` structure containing the ObjectPaths.

**ERRORS**

None.

**EXAMPLES**

None.

**APPLICATION USAGE**

None.

**SEE ALSO**

None.

**CHANGE HISTORY**

None.
NAME
CMPIBrokerFT.getInstance – get an Instance using an ObjectPath as a reference

SYNOPSIS
CMPIInstance* CMPIBrokerFT.getInstance(
    CMPIBroker* mb,
    CMPIContext *ctx,
    CMPIObjectPath* op,
    char** properties,
    CMPIStatus* rc
);

DESCRIPTION
The CMPIBrokerFT.getInstance() function shall retrieve an Instance using an ObjectPath as
a reference.

The mb argument points to a CMPIBroker structure. The ctx argument points to the context
object, and the op argument points to the source ObjectPath containing namespace,
classname, and key components.

The properties argument, if not NULL, is an array of elements defining one or more
Property names. Each returned Object must not include elements for any Properties missing
from this list.

The rc argument points to a CMPIStatus structure used to return the service return code.

RETURN VALUE
The CMPIBrokerFT.getInstance() returns a pointer to a CMPIInstance structure.

ERRORS
None.

EXAMPLES
None.

APPLICATION USAGE
None.

SEE ALSO
None.

CHANGE HISTORY
None.
Class 2 services are provided by the following functions.
NAME

CMPIBrokerFT.associatorNames – enumerates ObjectPaths associated with an Instance

SYNOPSIS

```c
CMPIEnumeration* CMPIBrokerFT.associatorNames(
    CMPIBroker* mb,
    CMPIContext *ctx,
    CMPIObjectPath* op,
    char* assocClass,
    char* resultClass,
    char* role,
    char* resultRole,
    CMPIStatus* rc
);
```

DESCRIPTION

The `CMPIBrokerFT.associatorNames()` function shall enumerate ObjectPaths associated with an Instance.

The `mb` argument points to a `CMPIBroker` structure. The `ctx` argument points to the context object, and the `op` argument points to the source ObjectPath containing namespace, classname, and key components.

The `assocClass` argument, if not NULL, shall be a valid Association Class name. It acts as a filter on the returned set of Objects by mandating that each returned Object must be associated to the source Object via an Instance of this Class or one of its subclasses.

The `resultclass` argument, if not NULL, shall be a valid Class name. It acts as a filter on the returned set of Objects by mandating that each returned Object must be either an Instance of this Class (or one of its subclasses).

The `role` argument, if not NULL, shall be a valid Property name. It acts as a filter on the returned set of Objects by mandating that each returned Object must be associated to the source Object via an Association in which the source Object plays the specified role (i.e. the name of the Property in the Association Class that refers to the source Object must match the value of this parameter).

The `resultrole` argument, if not NULL, shall be a valid Property name. It acts as a filter on the returned set of Objects by mandating that each returned Object must be associated to the source Object via an Association in which the returned Object plays the specified role (i.e. the name of the Property in the Association Class that refers to the returned Object must match the value of this parameter).

RETURN VALUE

The `CMPIBrokerFT.associatorNames()` function shall return a pointer to a `CMPIEnumeration` structure containing the enumeration of the ObjectPaths.

ERRORS

None.

EXAMPLES

None.

APPLICATION USAGE

None.
SEE ALSO
None.

CHANGE HISTORY
None.
NAME
CMPIBrokerFT.associators – enumerates Instances associated with an Instance

SYNOPSIS
CMPIEnumeration* CMPIBrokerFT.associators(
    CMPIBroker* mb,
    CMPIContext *ctx,
    CMPIObjectPath* op,
    char* assocClass,
    char* resultClass,
    char* role,
    char* resultRole,
    char** properties,
    CMPIStatus* rc
);

DESCRIPTION
The CMPIBrokerFT.associators() function shall enumerate Instances associated with an Instance.

The mb argument points to a CMPIBroker structure. The ctx argument points to the context object, and the op argument points to the source ObjectPath containing namespace, classname, and key components.

The assocClass argument, if not NULL, shall be a valid Association Class name. It acts as a filter on the returned set of Objects by mandating that each returned Object must be associated to the source Object via an Instance of this Class or one of its subclasses.

The resultclass argument, if not NULL, shall be a valid Class name. It acts as a filter on the returned set of Objects by mandating that each returned Object must be either an Instance of this Class (or one of its subclasses).

The role argument, if not NULL, shall be a valid Property name. It acts as a filter on the returned set of Objects by mandating that each returned Object must be associated to the source Object via an Association in which the source Object plays the specified role (i.e. the name of the Property in the Association Class that refers to the source Object must match the value of this parameter).

The resultrole argument, if not NULL, shall be a valid Property name. It acts as a filter on the returned set of Objects by mandating that each returned Object must be associated to the source Object via an Association in which the returned Object plays the specified role (i.e. the name of the Property in the Association Class that refers to the returned Object must match the value of this parameter).

The properties argument, if not NULL, is an array of elements defining one or more Property names. Each returned Object must not include elements for any Properties missing from this list.

The rc argument points to a CMPIStatus structure used to return the service return code.

RETURN VALUE
The CMPIBrokerFT.associators() function shall return a pointer to a CMPIEnumeration structure containing the enumeration of the Instances.

ERRORS
None.
EXAMPLES
None.

APPLICATION USAGE
None.

SEE ALSO
None.

CHANGE HISTORY
None.
NAME
   CMPIBrokerFT.attachThread – inform the CMPI run time system that the current thread will
   begin using CMPI services

SYNOPSIS
   CMPIStatus CMPIBrokerFT.attachThread(
   
   
   

DESCRIPTION
   The CMPIBrokerFT.attachThread() function shall inform the CMPI run time system that the
current thread with the specified context will begin using CMPI services.
   The mb argument points to a CMPIBroker structure. The ctx argument points to the context
object.

RETURN VALUE
   The CMPIBrokerFT.attachThread() function shall return a CMPIStatus structure containing
the service return status.

ERRORS
   None.

EXAMPLES
   None.

APPLICATION USAGE
   None.

SEE ALSO
   CMPIBrokerFT.detachThread(), CMPIBrokerFT.prepareAttachThread()

CHANGE HISTORY
   None.
CMPIBrokerFT.createInstance()

NAME

CMPIBrokerFT.createInstance – create an Instance using a specified ObjectPath as a reference

SYNOPSIS

CMPIObjectPath* CMPIBrokerFT.createInstance(
    CMPIBroker* mb,
    CMPIContext* ctx,
    CMPIObjectPath* op,
    CMPIInstance* inst,
    CMPIStatus* rc
);

DESCRIPTION

The CMPIBrokerFT.createInstance() function shall create an Instance using a specified Object Path as a reference.

The mb argument points to a CMPIBroker structure. The ctx argument points to the context object, and the op argument points to the source ObjectPath containing namespace, classname, and key components.

The inst argument points to a CMPIInstance structure.

The rc argument points to a CMPIStatus structure used to return the service return code.

RETURN VALUE

The CMPIBrokerFT.createInstance() function shall return a pointer to the assigned Instance reference.

ERRORS

None.

EXAMPLES

None.

APPLICATION USAGE

None.

SEE ALSO

None.

CHANGE HISTORY

None.
CMPIBrokerFT.deleteInstance()

NAME

CMPIBrokerFT.deleteInstance – deletes an existing Instance using a specified ObjectPath as a reference.

SYNOPSIS

CMPIStatus CMPIBrokerFT.deleteInstance(
    CMPIBroker* mb,
    CMPIContext *ctx,
    CMPIObjectPath* op
);

DESCRIPTION

The CMPIBrokerFT.deleteInstance() function shall delete an existing Instance using a specified Object Path as a reference.

The mb argument points to a CMPIBroker structure. The ctx argument points to the context object, and the op argument points to the source ObjectPath containing namespace, classname, and key components.

RETURN VALUE

The CMPIBrokerFT.deleteInstance() function shall return a CMPIStatus structure containing the service return status.

ERRORS

None.

EXAMPLES

None.

APPLICATION USAGE

None.

SEE ALSO

None.

CHANGE HISTORY

None.
CMPIBrokerFT.detachThread()

NAME

CMPIBrokerFT.detachThread – inform the CMPI run time system that the current thread will no longer be using CMPI services

SYNOPSIS

CMPIStatus CMPIBrokerFT.detachThread(
    CMPIBroker* mb,
    CMPIContext* ctx
);

DESCRIPTION

The CMPIBrokerFTdetachThread() function shall inform the CMPI run time system that the current thread will no longer be using CMPI services.

The mb argument points to a CMPIBroker structure. The ctx argument points to the context object.

RETURN VALUE

The CMPIBrokerFTdetachThread() function shall return a CMPIStatus structure containing the service return status.

ERRORS

None.

EXAMPLES

None.

APPLICATION USAGE

None.

SEE ALSO

None.

CHANGE HISTORY

None.
**CMPIBrokerFT.enumInstances()**

**NAME**
CMPIBrokerFT.enumInstances – enumerate Instances of the class (and subclasses) defined by an ObjectPath

**SYNOPSIS**
```c
CMPIEnumeration* CMPIBrokerFT.enumInstances(
    CMPIBroker* mb,
    CMPIContext *ctx,
    CMPIObjectPath* op,
    char** properties;
    CMPIStatus* rc
);
```

**DESCRIPTION**
The `CMPIBrokerFT.enumInstances()` function shall enumerate the Instance Names of the class (and subclasses) defined by the specified ObjectPath. Instance structure can be controled using the `CMPIInvocationFlags` entry in `ctx`.

The `mb` argument points to a `CMPIBroker` structure. The `ctx` argument points to the context object, and the `op` argument points to the source ObjectPath containing namespace and classname elements.

The `properties` argument, if not NULL, is an array of elements defining one or more Property names. Each returned Object must not include elements for any Properties missing from this list.

The `rc` argument points to a `CMPIStatus` structure used to return the service return code.

**RETURN VALUE**
The `CMPIBrokerFT.enumInstances()` function shall return a pointer to a `CMPIEnumeration` structure containing the ObjectPaths.

**ERRORS**
None.

**EXAMPLES**
None.

**APPLICATION USAGE**
None.

**SEE ALSO**
None.

**CHANGE HISTORY**
None.
NAME
CMPIBrokerEncFT.execQuery – query the enumeration of instances of the class (and subclasses) defined by an ObjectPath using a query expression

SYNOPSIS
CMPIEnumeration* CMPIBrokerEncFT.execQuery(
    CMPIBroker* mb,
    CMPIContext* ctx,
    CMPIOBJECTPath* op,
    char* query,
    char* lang,
    CMPIStatus* rc
);

DESCRIPTION
The CMPIBrokerEncFT.execQuery() function shall return a new CMPISelectExp object. The mb argument points to a CMPIBroker object. The ctx argument points to the context object, and the op argument points to the source ObjectPath containing namespace and classname elements.

The query argument is a pointer to a string containing the select expression. The lang argument is a pointer to a string containing the query language.

The rc argument points to a CMPIStatus structure containing the service return status.

RETURN VALUE
The CMPIBrokerEncFT.execQuery() function shall return a pointer to a CMPIEnumeration object containing the Instances.

ERRORS
None.

EXAMPLES
None.

APPLICATION USAGE
None.

SEE ALSO
None.

CHANGE HISTORY
None.
CMPIBrokerFT.getProperty()

NAME
CMPIBrokerFT.getProperty – get the named property value of an Instance defined by an ObjectPath

SYNOPSIS
CMPIData CMPIBrokerFT.getProperty(
    CMPIBroker* mb,
    CMPIContext *ctx,
    CMPIObjectPath* op,
    char* name,
    CMPIStatus* rc
);

DESCRIPTION
The CMPIBrokerFT.getProperty() function shall get the named property value of an Instance defined by an ObjectPath. The mb argument points to a CMPIBroker object. The ctx argument points to the context object, and the op argument points to the source ObjectPath containing namespace, classname, and key elements. The name argument points to a string containing the property name. The rc argument points to a CMPIStatus structure containing the service return status.

RETURN VALUE
The CMPIBrokerFT.getProperty() shall return a CMPIData structure containing the requested property value.

ERRORS
None.

EXAMPLES
None.

APPLICATION USAGE
None.

SEE ALSO
None.

CHANGE HISTORY
None.
CMPIBrokerFT.invokeMethod()

NAME
CMPIBrokerFT.invokeMethod – invoke a named, extrinsic method of an Instance defined by an ObjectPath

SYNOPSIS
CMPIData CMPIBrokerFT.invokeMethod(
    CMPIBroker* mb,
    CMPIContext* ctx,
    CMPIObjectPath* op,
    char* method,
    CMPIArgs* in,
    CMPIArgs* out,
    CMPIStatus* rc
);

DESCRIPTION
The CMPIBrokerFT.invokeMethod() function shall invoke a named, extrinsic method of an instance defined by a specified ObjectPath.

The mb argument points to a CMPIBroker object. The ctx argument points to the context object, and the op argument points to the source ObjectPath containing namespace, classname, and key elements.

The method argument points to a string containing the method name. The in argument points to a CMPIArgs structure containing the input parameters. The out argument points to a CMPIArgs structure containing the output parameters.

The rc argument points to a CMPIStatus structure containing the service return status.

RETURN VALUE
The CMPIBrokerFT.invokeMethod() function shall return a CMPIData structure containing the method return value.

ERRORS
None.

EXAMPLES
None.

APPLICATION USAGE
None.

SEE ALSO
None.

CHANGE HISTORY
None.
CMPIBrokerFT.prepareAttachThread()

NAME
CMPIBrokerFT.prepareAttachThread – prepares the CMPI run time system to accept a thread that will be using CMPI services

SYNOPSIS
CMPIContext* CMPIBrokerFT.prepareAttachThread(
    CMPIBroker* mb,
    CMPIContext* ctx
);

DESCRIPTION
The CMPIBrokerFT.attachThread() function shall prepare the CMPI run time system to accept a thread that will be using CMPI services.

The mb argument points to a CMPIBroker structure. The ctx argument points to the context object.

The returned CMPIContext object must be used by the subsequent attachThread() and detachThread() invocations.

RETURN VALUE
The CMPIBrokerFT.attachThread() function shall return a pointer to a CMPIContext structure to be used by the thread to be attached.

ERRORS
None.

EXAMPLES
None.

APPLICATION USAGE
None.

SEE ALSO
CMPIBrokerFT.attachThread(), CMPIBrokerFT.detachThread()

CHANGE HISTORY
None.
CMPIBrokerFT.referenceNames()

NAME
CMPIBrokerFT.referenceNames – enumerates the association ObjectPaths that refer to an Instance defined by an ObjectPath

SYNOPSIS
CMPIEnumeration* CMPIBrokerFT.referenceNames(
    CMPIBroker* mb,
    CMPIContext *ctx,
    CMPIObjectPath* op,
    char* resultClass,
    char* role,
    CMPIStatus* rc
);

DESCRIPTION
The CMPIBrokerFT.referenceNames() function shall enumerate the association ObjectPaths that refer to an Instance defined by an ObjectPath. The mb argument points to a CMPIBroker structure. The ctx argument points to the context object, and the op argument points to the source ObjectPath containing namespace, classname, and key components.

The resultclass argument, if not NULL, shall be a valid Class name. It acts as a filter on the returned set of Objects by mandating that each returned Object must be either an Instance of this Class (or one of its subclasses).

The role argument, if not NULL, shall be a valid Property name. It acts as a filter on the returned set of Objects by mandating that each returned Object must be associated to the source Object via an Association in which the source Object plays the specified role (i.e. the name of the Property in the Association Class that refers to the source Object must match the value of this parameter).

The rc argument points to a CMPIStatus structure containing the service return status.

RETURN VALUE
The CMPIBrokerFT.referenceNames() function shall return a pointer to a CMPIEnumeration structure containing the enumeration of the ObjectPaths.

ERRORS
None.

EXAMPLES
None.

APPLICATION USAGE
None.

SEE ALSO
None.

CHANGE HISTORY
None.
CMPIBrokerFT.references()

NAME
CMPIBrokerFT.references – enumerates the association Instances that refer to an Instance defined by an ObjectPath

SYNOPSIS

CMPIEnumeration* CMPIBrokerFT.references(
    CMPIBroker* mb,
    CMPIContext *ctx,
    CMPIObjectPath* op,
    char* resultClass,
    char* role,
    char** properties,
    CMPIStatus* rc
);

DESCRIPTION
The CMPIBrokerFT.references() function shall enumerate the association Instances that refer to an Instance defined by an ObjectPath.

The mb argument points to a CMPIBroker structure. The ctx argument points to the context object, and the op argument points to the source ObjectPath containing namespace, classname, and key components.

The resultclass argument, if not NULL, shall be a valid Class name. It acts as a filter on the returned set of Objects by mandating that each returned Object must be either an Instance of this Class (or one of its subclasses).

The role argument, if not NULL, shall be a valid Property name. It acts as a filter on the returned set of Objects by mandating that each returned Object must be associated to the source Object via an Association in which the source Object plays the specified role (i.e. the name of the Property in the Association Class that refers to the source Object must match the value of this parameter).

The properties argument, if not NULL, is an array of elements defining one or more Property names. Each returned Object must not include elements for any Properties missing from this list.

The rc argument points to a CMPIStatus structure used to return the service return code.

RETURN VALUE
The CMPIBrokerFT.references() function shall return a pointer to a CMPIEnumeration structure containing the enumeration of the ObjectPaths.

ERRORS
None.

EXAMPLES
None.

APPLICATION USAGE
None.

SEE ALSO
None.
CHANGE HISTORY

None.
CMPIBrokerFT.setInstance()

NAME
CMPIBrokerFT.setInstance – replace an existing Instance using an Object Path as reference

SYNOPSIS
CMPIStatus* CMPIBrokerFT.setInstance(
    CMPIBroker* mb,
    CMPIContext* ctx,
    CMPIObjectPath* op,
    CMPIInstance* inst
);

DESCRIPTION
The CMPIBrokerFT.setInstance() function shall replace an existing Instance using an ObjectPath as a reference.

The mb argument points to a CMPIBroker structure. The ctx argument points to the context object, and the op argument points to the source ObjectPath containing namespace, classname, and key components.

The inst argument points to a CMPIInstance structure containing a complete instance.

RETURN VALUE
The CMPIBrokerFT.setInstance() function shall return a CMPIStatus structure containing the service return status.

ERRORS
None.

EXAMPLES
None.

APPLICATION USAGE
None.

SEE ALSO
None.

CHANGE HISTORY
None.
CMPIBrokerFT.setProperty()

NAME
CMPIBrokerFT.setProperty – set the named property value of an Instance defined by an ObjectPath

SYNOPSIS

```c
CMPIStatus CMPIBrokerFT.setProperty(
    CMPIBroker* mb,
    CMPIContext *context,
    CMPIObjectPath* op,
    char* name,
    CMPIValue *value,
    CMPIType type
);
```

DESCRIPTION
The CMPIBrokerFT.setProperty() function shall set the named property value of an Instance defined by an ObjectPath.

The mb argument points to a CMPIBroker structure. The context argument points to the context object, and the op argument points to the source ObjectPath containing namespace, classname, and key components.

The name argument points to a string containing the property name.

The value argument points to a CMPIValue structure containing the value to be assigned to the property.

The type argument points to a CMPIType structure which defines the type of value.

RETURN VALUE
The CMPIBrokerFT.setProperty() function shall return a CMPIStatus structure containing the service return status.

ERRORS
None.

EXAMPLES
None.

APPLICATION USAGE
None.

SEE ALSO
None.

CHANGE HISTORY
None.
9.6 Accessing MB Services

The MB service routines outlined above are made available by the MB to MIs via the CMPIBroker structure. CMPIBrokerFT.bft points to the _CMPIBrokerFT structure.

9.6.1 CMPIBrokerFT

Conceptually the structure is as follows:

```c
typedef struct _CMPIBrokerFT {
    int brokerClassification;
    int brokerVersion;
    char *brokerName;
    CMPIStatus (*deliverIndication)(CMPIBroker*, CMPIContext*, CMPIObjectPath*, char*, CMPIInstance*, CMPISelectExp*);
    CMPIEnumeration* (*enumInstanceNames)(CMPIBroker*, CMPIContext*, CMPIObjectPath*, CMPIInstance*);
    CMPIInstance* (*getInstance)(CMPIBroker*, CMPIContext*, CMPIObjectPath*, char**, CMPIStatus*);
    CMPIObjectPath* (*createInstance)(CMPIBroker*, CMPIContext*, CMPIObjectPath*, CMPIInstance*, CMPIStatus*);
    CMPIStatus (*setInstance)(CMPIBroker*, CMPIContext*, CMPIObjectPath*, CMPIInstance*);
    CMPIStatus (*deleteInstance)(CMPIBroker*, CMPIContext*, CMPIObjectPath*);
    CMPIEnumeration* (*execQuery)(CMPIBroker*, CMPIContext*, CMPIObjectPath*, char*, char*, CMPIStatus*);
    CMPIEnumeration* (*enumInstances)(CMPIBroker*, CMPIContext*, CMPIObjectPath*, char**, CMPIStatus*);
    CMPIEnumeration* (*associators)(CMPIBroker*, CMPIContext*, CMPIObjectPath*, char*, char*, char*, char*, char**, CMPIStatus*);
    CMPIEnumeration* (*associatorNames)(CMPIBroker*, CMPIContext*, CMPIObjectPath*, char*, char*, char*, char*, char*, CMPIStatus*);
    CMPIEnumeration* (*references)(CMPIBroker*, CMPIContext*, CMPIObjectPath*, char*, char*, char*, char**, CMPIStatus*);
    CMPIEnumeration* (*referenceNames)(CMPIBroker*, CMPIContext*, CMPIObjectPath*, char*, char*, CMPIStatus*);
    CMPIData (*invokeMethod)(CMPIBroker*, CMPIContext*, CMPIObjectPath*, char*, CMPIArgs*, CMPIArgs*, CMPIStatus*);
};
```
CMPIStatus (*setProperty)
(CMPIBroker*, CMPIContext*,
CMPObj ectPath*, char*, CMPIValue*, CMPIType);
CMPIStatus (*setProperty)
(CMPIBroker*, CMPIContext*,
CMPObj ectPath*, char, CMPIrc*);
} CMPIBrokerFT;

Convenience macros:

define CBDel iverIndication(mb, ctx, ns, ci, se) \
((mb)->eft->deliverIndication((mb), (ctx), \
(ns), (ci), (se)))
...
#define CBSetProperty(mb, ctx, cop, name, val, type) \
((mb)->eft->setProperty((mb), (ctx), (cop), \
(name), (val), (type)))
#define CBGetProperty(mb, ctx, cop, name, type, rc) \
((mb)->eft->getProperty((mb), (ctx), (cop), \
(val), (type), (rc)))
...
}
A Header Files

A.1 Data Types (<cmpidt.h>)

/*
 * cmpidt.h
 * Copyright (c) 2002, International Business Machines Corporation
 * THIS FILE IS PROVIDED UNDER THE TERMS OF THE COMMON PUBLIC
 * LICENSE ("AGREEMENT"). ANY USE, REPRODUCTION, OR DISTRIBUTION
 * OF THIS FILE CONSTITUTES THE RECIPIENT’S ACCEPTANCE OF THE
 * AGREEMENT.
 * A current copy of the Common Public License is available from:
 * Author: Adrian Schuur <schuur@de.ibm.com>
 * Description: CMPI data types
 */

#ifndef _CMPIDT_H_
#define _CMPIDT_H_

#include <stdio.h>

#ifdef __cplusplus
extern "C" {
#endif

#define CMPIVersion051 51     // 0.51
#define CMPICurrentVersion CMPIVersion051

struct _CMPIBroker;
struct _CMPIInstance;
struct _CMPIObjectPath;
struct _CMPIArgs;
struct _CMPISelectExp;
struct _CMPISelectCond;
struct _CMPISubCond;
struct _CMPIPredicate;
struct _CMPIEnumeration;
struct _CMPIArray;
struct _CMPIString;
struct _CMPIResult;
struct _CMPIContext;
struct _CMPIDateTime;

typedef struct _CMPIBroker    CMPIBroker;
typedef struct _CMPIInstance  CMPIInstance;
typedef struct _CMPIObjectPath CMPIObjectPath;
typedef struct _CMPIArgs     CMPIArgs;
typedef struct _CMPISelectExp CMPISelectExp;
typedef struct _CMPISelectCond CMPISelectCond;
typedef struct _CMPISubCond   CMPISubCond;
typedef struct _CMPIPredicate CMPIPredicate;

#endif
"C" {
#endif
typedef struct _CMPIEnumeration CMPIEnumeration;
typedef struct _CMPIArray CMPIArray;
typedef struct _CMPIString CMPIString;
typedef struct _CMPIResult CMPIResult;
typedef struct _CMPIContext CMPIContext;
typedef struct _CMPIDateTime CMPIDateTime;

struct _CMPIBrokerFT;
struct _CMPIBrokerEncFT;
struct _CMPIInstanceFT;
struct _CMPIObjectPathFT;
struct _CMPIArgsFT;
struct _CMPISelectExpFT;
struct _CMPISelectCondFT;
struct _CMPISelectCondDocFT;
struct _CMPISelectCondCodFT;
struct _CMPISubCondFT;
struct _CMPIPredicateFT;
struct _CMPIEnumerationFT;
struct _CMPIArrayFT;
struct _CMPIStringFT;
struct _CMPIResultFT;
struct _CMPIContextFT;
struct _CMPIDateTimeFT;

typedef struct _CMPIBrokerFT CMPIBrokerFT;
typedef struct _CMPIBrokerEncFT CMPIBrokerEncFT;
typedef struct _CMPIInstanceFT CMPIInstanceFT;
typedef struct _CMPIObjectPathFT CMPIObjectPathFT;
typedef struct _CMPIArgsFT CMPIArgsFT;
typedef struct _CMPISelectExpFT CMPISelectExpFT;
typedef struct _CMPISelectCondFT CMPISelectCondFT;
typedef struct _CMPISelectCondDocFT CMPISelectCondDocFT;
typedef struct _CMPISubCondFT CMPISubCondFT;
typedef struct _CMPIPredicateFT CMPIPredicateFT;
typedef struct _CMPIEnumerationFT CMPIEnumerationFT;
typedef struct _CMPIArrayFT CMPIArrayFT;
typedef struct _CMPIStringFT CMPIStringFT;
typedef struct _CMPIResultFT CMPIResultFT;
typedef struct _CMPIContextFT CMPIContextFT;
typedef struct _CMPIDateTimeFT CMPIDateTimeFT;

typedef unsigned char CMPIBoolean;
typedef unsigned short CMPIChar16;
typedef unsigned char CMPIUint8;
typedef unsigned short CMPIUint16;
typedef unsigned long CMPIUint32;
typedef unsigned long long CMPIUint64;
typedef signed char CMPIInt8;
typedef short CMPIInt16;
typedef long CMPIInt32;
typedef long long CMPIInt64;
typedef float CMPIReal32;
typedef double CMPIReal64;

typedef struct _CMPIValuePtr {
    void *ptr;
    unsigned int length;
} CMPIValuePtr;
typedef union _CMPIValue {
    CMPIBoolean    boolean;
    CMPIChar16    char16;
    CMPIUint8     uint8;
    CMPIUint16    uint16;
    CMPIUint32    uint32;
    CMPIUint64    uint64;
    CMPISint8     sint8;
    CMPISint16    sint16;
    CMPISint32    sint32;
    CMPISint64    sint64;
    CMPIReal32    real32;
    CMPIReal64    real64;
    CMPIInstance* inst;
    CMPIObjectPath* ref;
    CMPIArgs* args;
    CMPISelectExp* filter;
    CMPISerialization* Enum;
    CMPIInstance* array;
    CMPIString* string;
    char* chars;
    CMPIDateTime* dateTime;
    CMPIValuePtr dataPtr;
    CMPIInstance* Byte;
    CMPIInstance* Short;
    CMPIInstance* Int;
    CMPIInstance* Long;
    CMPIInstance* Float;
    CMPIInstance* Double;
} CMPIValue;

typedef unsigned short CMPIType;

#define CMPI_SIMPLE       (2)
#define CMPI_boolean      (2+0)
#define CMPI_char16       (2+1)

#define CMPI_REAL         ((2)<<2)
#define CMPI_real32       ((2+0)<<2)
#define CMPI_real64       ((2+1)<<2)

#define CMPI_UINT         ((8)<<4)
#define CMPI_uint8        ((8+0)<<4)
#define CMPI_uint16       ((8+1)<<4)
#define CMPI_uint32       ((8+2)<<4)
#define CMPI_uint64       ((8+3)<<4)
#define CMPI_SINT         ((8+4)<<4)
#define CMPI_sint8        ((8+4) <<4)
#define CMPI_sint16       ((8+5)<<4)
#define CMPI_sint32       ((8+6)<<4)
#define CMPI_sint64       ((8+7)<<4)
#define CMPI_INTEGER      ((CMPI_UINT | CMPI_SINT))

#define CMPI_ENC          ((16)<<8)
#define CMPI_instance     ((16+0)<<8)
#define CMPI_ref          ((16+1)<<8)
#define CMPI_args         ((16+2)<<8)
#define CMPI_class        ((16+3)<<8)
#define CMPI_filter       ((16+4)<<8)
#define CMPI_enumeration  ((16+5)<<8)
#define CMPI_string       ((16+6)<<8)
#define CMPI_chars        ((16+7)<<8)
#define CMPI_dateTime     ((16+8)<<8)
#define CMPI_ptr          ((16+9)<<8)

#define CMPI_ARRAY        ((1)<<13)
#define CMPI_SIMPLEA      (CMPI_ARRAY | CMPI_SIMPLE)
#define CMPI_booleanA     (CMPI_ARRAY | CMPI_boolean)
#define CMPI_char16A      (CMPI_ARRAY | CMPI_char16)

#define CMPI_REALA        (CMPI_ARRAY | CMPI_REAL)
#define CMPI_real32A      (CMPI_ARRAY | CMPI_real32)
#define CMPI_real64A      (CMPI_ARRAY | CMPI_real64)

#define CMPI_UINTA        (CMPI_ARRAY | CMPI_UINT)
#define CMPI_uint8A       (CMPI_ARRAY | CMPI_uint8)
#define CMPI_uint16A      (CMPI_ARRAY | CMPI_uint16)
#define CMPI_uint32A      (CMPI_ARRAY | CMPI_uint32)
#define CMPI_uint64A      (CMPI_ARRAY | CMPI_uint64)

#define CMPI_SINTA        (CMPI_ARRAY | CMPI_SINT)
#define CMPI_sint8A       (CMPI_ARRAY | CMPI_sint8)
#define CMPI_sint16A      (CMPI_ARRAY | CMPI_sint16)
#define CMPI_sint32A      (CMPI_ARRAY | CMPI_sint32)
#define CMPI_sint64A      (CMPI_ARRAY | CMPI_sint64)

#define CMPI_INTEGERA     (CMPI_ARRAY | CMPI_INTEGER)

#define CMPI_ENCA         (CMPI_ARRAY | CMPI_ENC)
#define CMPI_stringA      (CMPI_ARRAY | CMPI_string)
#define CMPI_charsA       (CMPI_ARRAY | CMPI_chars)
#define CMPI_dateTimeA    (CMPI_ARRAY | CMPI_dateTime)

// The following are CMPIObjectPath key-type synonyms and
// are valid only when CMPI_keyValue of CMPIValueState is set:
#define CMPI_keyInteger   (CMPI_sint64)
#define CMPI_keyString    (CMPI_string)
#define CMPI_keyBoolean   (CMPI_boolean)
#define CMPI_keyRef       (CMPI_ref)

// The following are predicate types only:
#define CMPI_charString      (CMPI_string)
#define CMPI_numericString   (CMPI_string | CMPI_sint64)
#define CMPI_booleanString   (CMPI_string | CMPI_boolean)
#define CMPI_dateTimeString  (CMPI_string | CMPI_dateTime)
#define CMPI_classNameString (CMPI_string | CMPI_class)

typedef unsigned short CMPIValueState;
#define CMPI_nullValue (1<<8)
#define CMPI_keyValue  (2<<8)
#define CMPI_badValue  (0x80<<8)

typedef struct _CMPIData {
  CMPIType type;
  CMPIValueState state;
  CMPIValue value;
}
```c

Systems Management: Common Management Programming Interface (CMPI)

1. } CMPIData;
2. #ifndef CMPI_NO_SYNONYM_SUPPORT
3. #define CMPI_Byte    CMPI_sint8
4. #define CMPI_Short   CMPI_sint16
5. #define CMPI_Int     CMPI_sint32
6. #define CMPI_Long    CMPI_sint64
7. #define CMPI_Float   CMPI_real32
8. #define CMPI_Double  CMPI_real64
9. #endif // CMPI_NO_SYNONYM_SUPPORT
10
11. #define CMPI_ByteA   CMPI_sint8A
12. #define CMPI_ShortA  CMPI_sint16A
13. #define CMPI_IntA    CMPI_sint32A
14. #define CMPI_LongA   CMPI_sint64A
15. #define CMPI_FloatA  CMPI_real32A
16. #define CMPI_DoubleA CMPI_real64A
17
18. typedef unsigned int CMPICount;
19
20. typedef unsigned int CMPIFlags;
21
22. #define CMPI_FLAG_LocalOnly          1
23. #define CMPI_FLAG_DeepInheritance    2
24. #define CMPI_FLAG_IncludeQualifiers  4
25. #define CMPI_FLAG_IncludeClassOrigin 8
26
27. #define CMPIInvocationFlags "CMPIInvocationFlags"
28
29. typedef enum _CMPIrc {
30.   CMPI_RC_OK                               =0,
31.   CMPI_RC_ERR_FAILED                       =1,
32.   CMPI_RC_ERR_ACCESS_DENIED               =2,
33.   CMPI_RC_ERR_INVALID_NAMESPACE           =3,
34.   CMPI_RC_ERR_INVALID_PARAMETER           =4,
35.   CMPI_RC_ERR_INVALID_CLASS               =5,
36.   CMPI_RC_ERR_NOT_FOUND                   =6,
37.   CMPI_RC_ERR_NOT_SUPPORTED               =7,
38.   CMPI_RC_ERR_CLASS_HAS_CHILDREN         =8,
39.   CMPI_RC_ERR_CLASS_HAS_INSTANCES         =9,
40.   CMPI_RC_ERR_INVALID_SUPERCLASS          =10,
41.   CMPI_RC_ERR_ALREADY_EXISTS              =11,
42.   CMPI_RC_ERR_NO_SUCH_PROPERTY            =12,
43.   CMPI_RC_ERR_TYPE_MISMATCH               =13,
44.   CMPI_RC_ERR_QUERY_LANGUAGE_NOT_SUPPORTED =14,
45.   CMPI_RC_ERR_INVALID_QUERY               =15,
46.   CMPI_RC_ERR_METHOD_NOT_AVAILABLE        =16,
47.   CMPI_RC_ERR_METHOD_NOT_FOUND            =17,
48.   CMPI_RC_ERROR_SYSTEM                    =100,
49.   CMPI_RC_ERROR                           =200,
50. } CMPIrc;
51
52. typedef struct _CMPIStatus {
53.   CMPIrc rc;
54.   CMPICstring *msg;
55. } CMPIStatus;
56
57```

/ * Management Instrumentation type */

#define CMPI_MIType_Instance 1
#define CMPI_MIType_Association 2
#define CMPI_MIType_Method 4
#define CMPI_MIType_Property 8
#define CMPI_MIType_Indication 16

/* Management Broker classification and feature support */

#define CMPI_MB_Class_0 0x00000001
#define CMPI_MB_Class_1 0x00000003
#define CMPI_MB_Class_2 0x00000007

#define CMPI_MB_Supports_PropertyMI 0x00000100
#define CMPI_MB_Supports_IndicationMI 0x00000200
#define CMPI_MB_Supports_IndicationPolling 0x00000400
#define CMPI_MB_Supports_QueryNormalization 0x00000800
#define CMPI_MB_Supports_Qualifier 0x00001000
#define CMPI_MB_Supports_Schema 0x00003000

/* Query Predicate operations */

typedef enum _CMPIPredOp {
  CMPI_PredOp_Equals =1,
  CMPI_PredOp_NotEquals =2,
  CMPI_PredOp_LessThan =3,
  CMPI_PredOp_GreaterThanOrEquals =4,
  CMPI_PredOp_GreaterThan =5,
  CMPI_PredOp_LessThanOrEquals =6,
  CMPI_PredOp_Isa =7,
  CMPI_PredOp_NotIsa =8,
  CMPI_PredOp_Like =9,
  CMPI_PredOp_NotLike =10,
} CMPIPredOp;

#ifdef __cplusplus
};
#endif

#endif // __cplusplus

#ifdef __cplusplus
#endif

#endif // _CMPIDT_H_
A.2 Function Tables (<cmpift.h>)

/*
* cmpift.h
* Copyright \(\text{co} 2002, \text{International Business Machines Corporation}
* THIS FILE IS PROVIDED UNDER THE TERMS OF THE COMMON PUBLIC LICENSE ("AGREEMENT"). ANY USE, REPRODUCTION, OR DISTRIBUTION
* OF THIS FILE CONSTITUTES THE RECIPIENT’S ACCEPTANCE OF THE AGREEMENT.
* A current copy of the Common Public License is available from:
* Author: Adrian Schuur <schuur@de.ibm.com>
* Description: CMPI encapsulated types function tables
*/

#ifndef _CMPIFT_H_
#define _CMPIFT_H_

#include <cmpidt.h>
#include <cmpimacs.h>

#ifdef __cplusplus
extern "C" {
#endif

struct _CMPIBrokerEncFT {
    int ftVersion;
    CMPInstance* (*newInstance)(CMPIBroker*,CMPIObjectPath*,CMPIStatus*);
    CMPObjectPath* (*newObjectPath)(CMPIBroker*,char*,char*,CMPIStatus*);
    CMPArgs* (*newArgs)(CMPIBroker*,CMPIStatus*);
    CMPIString* (*newString)(CMPIBroker*,char*,CMPIStatus*);
    CMPIArray* (*newArray)(CMPIBroker*,CMPICount,CMPIType,CMPIStatus*);
    CMPIDateTime* (*newDateTime)(CMPIBroker*,CMPIStatus*);
    CMPIDateTime* (*newDateTimeFromBinary)(CMPIBroker*,CMPUint64,CMPIBool,CMPIStatus*);
    CMPIDateTime* (*newDateTimeFromChars)(CMPIBroker*,char*,CMPIStatus*);
    CMPISelectExp* (*newSelectExp)(CMPIBroker*,char*,char*, CMPIArr**,CMPIStatus*);
    CMPIString* (*toString)(CMPIBroker*,void*,CMPIStatus*);
    CMPIBool (*isOfType)(CMPIBroker*,void*,char*,CMPIStatus*);
    CMPIString* (*getType)(CMPIBroker*,void*,CMPIStatus*);
    CMPIString (*logMessage)(CMPIBroker*,int,int,char*);
};
struct _CMPIBrokerFT {
  unsigned long brokerClassification;
  int brokerVersion;
  char *brokerName;
  CMPIContext* (*prepareAttachThread)(CMPIBroker*,CMPIContext*);
  CMPIStatus (*attachThread)(CMPIBroker*,CMPIContext*);
  CMPIStatus (*detachThread)(CMPIBroker*,CMPIContext*);

  // class 0 services
  CMPIStatus (*deliverIndication)(CMPIBroker*,CMPIContext*,
      char*,CMPIInstance*,CMPISelectExp*);
  // class 1 services
  CMPIEnumeration* (*enumInstanceNames)(CMPIBroker*,CMPIContext*, CMPIObjectPath*,CMPIStatus*);
  CMPIInstance* (*getInstance)(CMPIBroker*,CMPIContext*,
      CMPIObjectPath*,char**,CMPIStatus*);
  // class 2 services
  CMPIObjectPath* (*createInstance)(CMPIBroker*,CMPIContext*, CMPIInstance*,CMPIStatus*);
  CMPIStatus (*setInstance)(CMPIBroker*,CMPIContext*,CMPIInstance*);
  CMPIStatus (*deleteInstance)(CMPIBroker*,CMPIContext*, CMPIObjectPath*);
  CMPIEnumeration* (*execQuery)(CMPIBroker*,CMPIContext*,
      CMPIObjectPath*,char*,char*,CMPIStatus*);
  CMPIEnumeration* (*enumInstances)(CMPIBroker*,CMPIContext*,
      CMPIObjectPath*,char**,CMPIStatus*);
  CMPIEnumeration* (*associators)(CMPIBroker*,CMPIContext*,
      CMPIObjectPath*,char*,char*,char*,char*,CMPIStatus*);
  CMPIEnumeration* (*associatorNames)(CMPIBroker*,CMPIContext*,
      CMPIObjectPath*,char*,char*,char*,char*,char*,CMPIStatus*);
  CMPIEnumeration* (*references)(CMPIBroker*,CMPIContext*,
      CMPIObjectPath*,char*,char*,char*,CMPIStatus*);
  CMPIEnumeration* (*referenceNames)(CMPIBroker*,CMPIContext*,
      CMPIObjectPath*,char*,char*,CMPIStatus*);
  CMPIData (*invokeMethod)(CMPIBroker*,CMPIContext*, CMPIObjectPath*,
      char*,char*,CMPIStatus*);
  CMPIStatus (*setProperty)(CMPIBroker*,CMPIContext*, CMPIObjectPath*,
      char*,CMPIValue*,CMPIType);
  CMPIData (*getProperty)(CMPIBroker*,CMPIContext*,
      CMPIObjectPath*,char*,CMPIStatus*);
};
```c
struct _CMPIBroker {
    void *hdl;
    CMPIBrokerFT *bft;
    CMPIBrokerEncFT *eft;
};

struct _CMPIContextFT {
    int ftVersion;
    CMPIStatus (*release)(CMPIContext*);
    CMPIContext* (*clone)(CMPIContext*, CMPIStatus*);
    void* (*reserved1)(CMPIContext*, CMPIStatus*);
    CMPIData (*getEntry)(CMPIContext*, char*, CMPIStatus*);
    CMPIData (*getEntryAt)(CMPIContext*, unsigned int, CMPIString**, CMPIStatus*);
    unsigned int (*getEntryCount)(CMPIContext*, CMPIStatus*);
    CMPIStatus (*addEntry)(CMPIContext*, char*, CMPIValue*, CMPIType);
};

struct _CMPIContext {
    void *hdl;
    CMPIContextFT *ft;
};

struct _CMPIResult {
    void *hdl;
    CMPIResultFT *ft;
};

struct _CMPIResultFT {
    int ftVersion;
    CMPIStatus (*release)(CMPIResult*);
    CMPIResult* (*clone)(CMPIResult*, CMPIStatus*);
    void* (*reserved1)(CMPIResult*, CMPIStatus*);
    CMPIStatus (*returnData)(CMPIResult*, CMPIValue*, CMPIType);
    CMPIStatus (*returnInstance)(CMPIResult*, CMPIInstance*);
    CMPIStatus (*returnObjectPath)(CMPIResult* eRes, CMPIObjectPath* eRef);
    CMPIStatus (*returnDone)(CMPIResult*);
};

struct _CMPIInstance {
    void *hdl;
    CMPIInstanceFT* ft;
};

struct _CMPIInstanceFT {
    int ftVersion;
};
```
CMPIStatus (*release)  
  (CMPIInstance*);  
CMPIInstance* (*clone)  
  (CMPIInstance*,CMPIStatus*);  
void* (*reserved1)  
  (CMPIInstance*,CMPIStatus*);  
CMPIData (*getProperty)  
  (CMPIInstance*,char*,CMPIStatus*);  
CMPIData (*getPropertyAt)  
  (CMPIInstance*,unsigned int,CMPIString**,CMPIStatus*);  
unsigned int (*getPropertyCount)  
  (CMPIInstance*,CMPIStatus*);  
CMPIStatus (*setProperty)  
  (CMPIInstance*,char*, CMPIValue*,CMPIType);  
CMPIObjectPath* (*getObjectPath)  
  (CMPIInstance*,CMPIStatus*);  
CMPIBoolean (*classIsA)  
  (CMPIInstance*,char*,CMPIStatus*);  
};  

struct _CMPIObjectPath {  
  void *hdl;  
  CMPIObjectPathFT* ft;  
#ifdef __cplusplus  
  #endif  
};  

struct _CMPIObjectPathFT {  
  int ftVersion;  
  CMPIStatus (*release) (CMPIObjectPath*);  
CMPIObjectPath* (*clone) (CMPIObjectPath*,CMPIStatus*);  
void* (*reserved1) (CMPIObjectPath*,CMPIStatus*);  
CMPIStatus (*setNameSpace) (CMPIObjectPath*,char*);  
CMPIString* (*getNameSpace) (CMPIObjectPath*,CMPIStatus*);  
CMPIStatus (*setClassName) (CMPIObjectPath*,char*);  
CMPIString* (*getClassQualifier) /* optional qualifier support */  
CMPIObjectPath* opThis, char *qualifierName,  
CMPIStatus *rc);  
CMPIData (*getPropertyQualifier) (CMPIObjectPath* opThis,  
char *propertyName, CMPIStatus *rc);  
CMPIData (*getMethodQualifier) (CMPIObjectPath* opThis,  
char *methodName, CMPIStatus *rc);  
CMPIData (*getParameterQualifier) (CMPIObjectPath* opThis,  
char *methodName, char *parameterName, CMPIStatus *rc);  
};
struct __CMPISelectExp {
    void *hdl;
    CMPISelectExpFT* ft;
};

struct __CMPISelectExpFT {
    int ftVersion;
    CMPIStatus (*release)(CMPISelectExp*);
    CMPISelectExp* (*clone)(CMPISelectExp*, CMPIStatus*);
    void* (*reserved1)(CMPISelectExp*, CMPIStatus*);
    CMPIBoolean (*evaluate)(CMPISelectExp*, CMPIInstance*, CMPIStatus*);
    CMPIString* (*getString)(CMPISelectExp*, CMPIStatus*);
    CMPISelectCond* (*getDOC)(CMPISelectExp*, CMPIStatus*);
    CMPISelectCond* (*getCOD)(CMPISelectExp*, CMPIStatus*);
};

struct __CMPISelectCond {
    void *hdl;
    CMPISelectCondFT* ft;
};

struct __CMPISelectCondFT {
    int ftVersion;
    CMPIStatus (*release)(CMPISelectCond*);
    CMPISelectCond* (*clone)(CMPISelectCond*, CMPIStatus*);
    void* (*reserved1)(CMPISelectCond*, CMPIStatus*);
    CMPICount (*getCountAndType)(CMPISelectCond*, int*, CMPIStatus*);
    CMPISubCond* (*getSubCondAt)(CMPISelectCond*, unsigned int, CMPIStatus*);
};

struct __CMPISubCond {
    void *hdl;
    CMPISubCondFT* ft;
};

struct __CMPISubCondFT {
    int ftVersion;
    CMPIStatus (*release)(CMPISubCond*);
    CMPISubCond* (*clone)(CMPISubCond*, CMPIStatus*);
    void* (*reserved1)(CMPISubCond*, CMPIStatus*);
    CMPICount (*getCount)(CMPISubCond*, CMPIStatus*);
    CMPIPredicate* (*getPredicateAt)(CMPISubCond*, unsigned int, CMPIStatus*);
CMPIPredicate* (*getPredicate)
  (CMPISubCond*,char*,CMPIStatus*);
);

struct _CMPIPredicate {
  void *hdl;
  CMPIPredicateFT* ft;
};

struct _CMPIPredicateFT {
  int ftVersion;
  CMPIStatus (*release) (CMPIPredicate*);
  CMPIPredicate* (*clone) (CMPIPredicate*,CMPIStatus*);
  void* (*reserved1) (CMPIPredicate*,CMPIStatus*);
  CMPIStatus (*getData) (CMPIPredicate*,CMPIType*,
    CMPISredOp*,CMPIString**,CMPIString**);
  int (*evaluate) (CMPIPredicate*,CMPIValue*,
    CMPIType,CMPIStatus*);
};

struct _CMPIArgs {
  void *hdl;
  CMPIArgsFT* ft;
};

struct _CMPIArgsFT{
  int ftVersion;
  CMPIStatus (*release) (CMPIArgs*);
  CMPIArgs* (*clone) (CMPIArgs*,CMPIStatus*);
  void* (*reserved1) (CMPIArgs*,CMPIStatus*);
  CMPIStatus (*addArg)
    (CMPIArgs*,char*,CMPIValue*, CMPIType);
  CMPIData (*getArg) (CMPIArgs*,char*,CMPIStatus*);
  CMPIData (*getArgAt)
    (CMPIArgs*,unsigned int,CMPIString**,CMPIStatus*);
  unsigned int (*getArgCount) (CMPIArgs*,CMPIStatus*);
};

struct _CMPIString {
  void *hdl;
  CMPIStringFT* ft;
};

struct _CMPIStringFT {
  int ftVersion;
  CMPIStatus (*release) (CMPIString*);
  CMPIString* (*clone) (CMPIString*,CMPIStatus*);
  void* (*reserved1) (CMPIString*,CMPIStatus*);
  char* (*getCharPtr) (CMPIString*,CMPIStatus*);
};

struct _CMPIArray {
  void *hdl;
  CMPIArrayFT* ft;
};

struct _CMPIArrayFT {
  int ftVersion;
  CMPIStatus (*release) (CMPIArray*);
CMPIArray* (*clone) (CMPIArray*,CMPIStratus*);
void* (*reserved1) (CMPIArray*,CMPIStratus*);
CMPICount (*getSize) (CMPIArray*,CMPIStratus*);
CMPIType (*getSimpleType) (CMPIArray*,CMPIStratus*);
CMPIData (*getElementAt)
(CMPIArray*,CMPICount,CMPIStratus*);
CMPIStatus (*setElementAt)
(CMPIArray*,CMPICount,CMPIData,CMPIValue*,CMPIType);

struct _CMPIEnumeration {
    void *hdl;
    CMPIStratusFT* ft;
};

struct _CMPIEnumerationFT {
    int ftVersion;
    CMPIStratus (*release) (CMPIEnumeration*);
    CMPIEnumeration* (*clone) (CMPIEnumeration*,CMPIStratus*);
    CMPIStratus (*reserved1)
    (CMPIEnumeration*,CMPIEnumeration*,CMPIStratus*);
    CMPIData (*getElementAt)
    (CMPIEnumeration*,CMPIEnumeration*,CMPIStratus*);
    CMPIStratus (*setElementAt)
    (CMPIEnumeration*,CMPIEnumeration*,CMPIStratus*);
    CMPIArray* (*toArray) (CMPIEnumeration*,CMPIStratus*);
};

struct _CMPIDateTime {
    void *hdl;
    CMPIDateTimeFT *ft;
};

struct _CMPIDateTimeFT {
    int ftVersion;
    CMPIStratus (*release) (CMPIDateTime*);
    CMPIDateTime* (*clone) (CMPIDateTime*,CMPIStratus*);
    void* (*reserved1) (CMPIDateTime*,CMPIStratus*);
    CMPIStratus (*getStringFormat) (CMPIDateTime*,CMPIStratus*);
    CMPIStratus (*isInterval) (CMPIDateTime*,CMPIStratus*);
};

typedef struct _CMPIInstanceMIFT CMPIInstanceMIFT;
typedef struct _CMPIInstanceMI {
    void *hdl;
    CMPIInstanceMIFT *ft;
} CMPIInstanceMI;

struct _CMPIInstanceMIFT {
    int ftVersion;
    int miVersion;
    char *miName;
    CMPIStratus (*cleanup) (CMPIInstanceMI*,CMPIContext*);
    CMPIStratus (*enumInstanceNames)
    (CMPIInstanceMI*,CMPIContext*,CMPIResult*,
    CMPIObj ectPath*);
    CMPIStratus (*enumInstances)
    (CMPIInstanceMI*,CMPIContext*,CMPIResult*,
    CMPIObj ectPath*,char**);
    CMPIStratus (*getInstance)
(CMPIInstanceMI*,CMPIContext*,CMPIResult*,
  CMPObj ectPath*,char**);
CMPIstatus (*createInstance)
(CMPIInstanceMI*,CMPIContext*,CMPIResult*,
  CMPObj ectPath*,CMPIInstance*);
CMPIstatus (*setInstance)
(CMPIInstanceMI*,CMPIContext*,CMPIResult*,
  CMPObj ectPath*,CMPIInstance*,char**);
CMPIstatus (*deleteInstance)
(CMPIInstanceMI*,CMPIContext*,CMPIResult*,
  CMPObj ectPath*);
CMPIstatus (*execQuery)
(CMPIInstanceMI*,CMPIContext*,CMPIResult*,
  CMPObj ectPath*,char*,char*);
);

typedef struct _CMPIAssociationMIFT CMPIAssociationMIFT;
typedef struct _CMPIAssociationMI {
  void *hdl;
  CMPIAssociationMIFT *ft;
} CMPIAssociationMI;

struct _CMPIAssociationMIFT {
  int ftVersion;
  int miVersion;
  char *miName;
  CMPIstatus (*cleanup)
    (CMPIAssociationMI*,CMPIContext*);
  CMPIstatus (*associators)
    (CMPIAssociationMI*,CMPIContext*,CMPIResult*,
     CMPObj ectPath*,char*,char*,char*,char*,char*,char**);
  CMPIstatus (*associatorNames)
    (CMPIAssociationMI*,CMPIContext*,CMPIResult*,
     CMPObj ectPath*,char*,char*,char*,char*,char*);
  CMPIstatus (*references)
    (CMPIAssociationMI*,CMPIContext*,CMPIResult*,
     CMPObj ectPath*,char*,char*,char*,char*);
  CMPIstatus (*referenceNames)
    (CMPIAssociationMI*,CMPIContext*,CMPIResult*,
     CMPObj ectPath*,char*,char*);
};

typedef struct _CMPIMethodMIFT CMPIMethodMIFT;
typedef struct _CMPIMethodMI {
  void *hdl;
  CMPIMethodMIFT *ft;
} CMPIMethodMI;

struct _CMPIMethodMIFT {
  int ftVersion;
  int miVersion;
  char *miName;
  CMPIstatus (*cleanup)
    (CMPIMethodMI*,CMPIContext*);
  CMPIstatus (*invokeMethod)
    (CMPIMethodMI*,CMPIContext*,CMPIResult*,
     CMPObj ectPath*,char*,CMPIArgs*,CMPIArgs*);
typedef struct _CMPIPropertyMIFT CMPIPropertyMIFT;
typedef struct _CMPIPropertyMI {
    void *hdl;
    CMPIPropertyMIFT *ft;
} CMPIPropertyMI;

struct _CMPIPropertyMIFT {
    int ftVersion;
    int miVersion;
    char *miName;
    CMPIStatus (*cleanup)(CMPIPropertyMI*,CMPIContext*);
    CMPIStatus (*setProperty)(CMPIPropertyMI*,CMPIContext*,CMPIResult*,
                              CMPIObjectPath*,char*,CMPIData);
    CMPIStatus (*getProperty)(CMPIPropertyMI*,CMPIContext*,CMPIResult*,
                              CMPIObjectPath*,char*);
};

typedef struct _CMPIIndicationMIFT CMPIIndicationMIFT;
typedef struct _CMPIIndicationMI {
    void *hdl;
    CMPIIndicationMIFT *ft;
} CMPIIndicationMI;

struct _CMPIIndicationMIFT {
    int ftVersion;
    int miVersion;
    char *miName;
    CMPIStatus (*cleanup)(CMPIIndicationMI*,CMPIContext*);
    CMPIStatus (*authorizeFilter)(CMPIIndicationMI*,CMPIContext*,CMPIResult*,
                                   CMPISelectExp*,char*,CMPIObjectPath*,char*);
    CMPIStatus (*mustPoll)(CMPIIndicationMI*,CMPIContext*,CMPIResult*,
                           CMPISelectExp*,char*,CMPIObjectPath*);
    CMPIStatus (*activateFilter)(CMPIIndicationMI*,CMPIContext*,CMPIResult*,
                                CMPISelectExp*,char*,CMPIObjectPath*,CMPIBool);
    CMPIStatus (*deActivateFilter)(CMPIIndicationMI*,CMPIContext*,CMPIResult*,
                                   CMPISelectExp*,char*,CMPIObjectPath*,CMPIBool);
};

#ifdef __cplusplus
};
#endif // _CMPIFT_H_
B  MI Convenience Support

B.1  C++ Convenience Classes

THIS SECTION IS INCOMPLETE.

CmpiString

/*  
 *  CmpiString.h  
 *  THIS FILE IS PROVIDED UNDER THE TERMS OF THE COMMON PUBLIC  
 *  LICENSE ("AGREEMENT"). ANY USE, REPRODUCTION, OR DISTRIBUTION  
 *  OF THIS FILE CONSTITUTES THE RECIPIENT’S ACCEPTANCE OF THE  
 *  AGREEMENT.  
 *  A current copy of the Common Public License is available from:  
 *  Author: Adrian Schuur <schuur@de.ibm.com>  
 *  Description: CMPI string provider wrapper  
 */  

#ifndef _CmpiString_h_
#define _CmpiString_h_

#include "cmpisrv.h"

class CmpiContext;
class CmpiResult;
class CmpiObjectPath;
class CmpiInstance;
class CmpiBroker;
class CmpiString;
class CmpiResult;
class CmpiContext;

class CmpiString {
    friend class CmpiBroker;
    friend class CmpiData;
    protected:
        CMPIString *enc;
    private:
        public:
            void makeGlobal(const CmpiString) {}  
            CmpiString(const CmpiString& s) { enc=s.enc->ft 
                ->clone(s.enc,NULL); }
            CmpiString() { enc=NULL; }
            CmpiString(CMPIString* c) { enc=c; }
            char* charPtr() { if (enc) return (char*)enc->hdl;
                else return NULL; }
        }
    #endif
CmpiData

/*
* CmpiData.h
* THIS FILE IS PROVIDED UNDER THE TERMS OF THE COMMON PUBLIC
* LICENSE ("AGREEMENT"). ANY USE, REPRODUCTION, OR DISTRIBUTION
* OF THIS FILE CONSTITUTES THE RECIPIENT’S ACCEPTANCE OF THE
* AGREEMENT.
* A current copy of the Common Public License is available from:
* Author: Adrian Schuur <schuur@de.ibm.com>
* Description: CMPI C++ helper class
*/

#ifndef _CmpiData_h_
#define _CmpiData_h_

#include "cmpidt.h"
#include "CmpiString.h"

class CmpiData {
    public:
        CMPIValue data;
        CMPIType type;
        CMPICount count;
    CmpiData() {}
    CmpiData(CmpiString& d) {data.chars=d.charPtr();
        type=CMPI_chars; count=-1;}
    CmpiData(char* d) {data.chars=d; type=CMPI_chars;
        count=-1;}
    CmpiData(CMPISint8 d) {data.sint8=d; type=CMPI_sint8;
        count=-1;}
    CmpiData(CMPISint16 d) {data.sint16=d; type=CMPI_sint16;
        count=-1;}
    CmpiData(CMPISint32 d) {data.sint32=d; type=CMPI_sint32;
        count=-1;}
    CmpiData(CMPISint64 d) {data.sint64=d; type=CMPI_sint64;
        count=-1;}
    CmpiData(CMPIUint8 d) {data.sint8=d; type=CMPI_sint8;
        count=-1;}
    CmpiData(CMPIUint16 d) {data.sint16=d; type=CMPI_sint16;
        count=-1;}
    CmpiData(CMPIUint32 d) {data.sint32=d; type=CMPI_sint32;
        count=-1;}
    CmpiData(CMPIUint64 d) {data.sint64=d; type=CMPI_sint64;
        count=-1;}
};

CmpiObjectPath

/*
* CmpiObjectPath.h
* THIS FILE IS PROVIDED UNDER THE TERMS OF THE COMMON PUBLIC
* LICENSE ("AGREEMENT"). ANY USE, REPRODUCTION OR DISTRIBUTION
* OF THIS FILE CONSTITUTES THE RECIPIENT’S ACCEPTANCE OF THE
* AGREEMENT.
* A current copy of the Common Public License is available from:
*/
* Author: Adrian Schuur <schuur@de.ibm.com>
* Description: CMPI C++ ObjectPath wrapper
*/

#ifndef _CmpiObjectPath_h_
#define _CmpiObjectPath_h_

#include "cmpisrv.h"

#include "CmpiData.h"

class CmpiObjectPath {
friend class CmpiBroker;
friend class CmpiResult;
protected:
CMPIObjectPath *enc;
private:
CmpiObjectPath() {}
public:
void makeGlobal() {};
CmpiObjectPath(CMPIObjectPath* c) { enc=c; }
CmpiString getNameSpace() {
    CMPIrc rc;
    CMPIString *s=enc->ft->getNameSpace(enc,&rc);
    if (rc!=CMPI_RC_OK) throw rc;
    return *new CmpiString(s);
}
void setNameSpace(CmpiString ns) { setNameSpace(ns.charPtr()); }
void setNameSpace(char* ns) {
    CMPIrc rc=enc->ft->setNameSpace(enc,ns);
    if (rc!=CMPI_RC_OK) throw rc;
}
CmpiString getClassName() {
    CMPIrc rc;
    CMPIString *s=enc->ft->getClassName(enc,&rc);
    if (rc!=CMPI_RC_OK) throw rc;
    return *new CmpiString(s);
}
CmpiData getnKey(char* name) {
    CmpiData d;
    CMPIrc rc;
    d.data=enc->ft->getKey(enc,name,&d.type,&rc);
    if (rc!=CMPI_RC_OK) throw rc;
    return d;
}
unsigned int getKeyCount() {
    CMPIrc rc;
    unsigned int c=enc->ft->getKeyCount(enc,&rc);
    if (rc!=CMPI_RC_OK) throw rc;
    return c;
}
CmpiData getKey(int pos, CmpiString *name=NULL) {
    CmpiData d;
    CMPIrc rc;
    CMPIString *s;
    d.data=enc->ft->getKeyAt(enc,pos,&s,&d.type,&rc);
    if (rc!=CMPI_RC_OK) throw rc;
    if (name) *name=*(new CmpiString(s));
    return d;
void addKey(char* name, CmpiData& data) {
    CMPIrc rc=enc->ft->
        addKey(enc,name,&data.data,data.type,data.count);
    if (rc!=CMPI_RC_OK) throw rc;
}

CmpiInstance

/*
* CmpiInstance.h
* THIS FILE IS PROVIDED UNDER THE TERMS OF THE COMMON PUBLIC
* LICENSE ("AGREEMENT"). ANY USE, REPRODUCTION, OR DISTRIBUTION
* OF THIS FILE CONSTITUTES THE RECIPIENT’S ACCEPTANCE OF THE
* AGREEMENT.
* A current copy of the Common Public License is available from:
* Author: Adrian Schuur <schuur@de.ibm.com>
* Description: CMPI C++ Instance wrapper
*/
#ifndef _CmpiInstance_h_
#define _CmpiInstance_h_
#include "cmpisrv.h"
#include "CmpiObjectPath.h"

class CmpiInstance {
    friend class CmpiBroker;
    friend class CmpiResult;
    protected:
        CmpiInstance *enc;
    private:
        CmpiInstance() {};
    public:
        void makeGlobal() {}
    CmpiInstance(CMPIInstance* enc) { this->enc=enc; }
    CmpiData getProperty(char* name) {
        CmpiData d;
        CMPIrc rc;
        d.data=enc->ft->getProperty(enc,name,&d.type,&rc);
        if (rc!=CMPI_RC_OK) throw rc;
        return d;
    }
    CmpiData getProperty(int pos, CmpiString *name=NULL) {
        CmpiData d;
        CMPIrc rc;
        CmpiString *s;
        d.data=enc->ft->getPropertyAt(enc,pos,&s,&d.type,&rc);
        if (rc!=CMPI_RC_OK) throw rc;
        if (name) *name=*(new CmpiString(s));
        return d;
    }
};
unsigned int getPropertyCount() {
    CMPIrc rc;
    unsigned int c = enc->ft->getPropertyCount(enc, &rc);
    if (rc!=CMPI_RC_OK) throw rc;
    return c;
}

void setProperty(char* name, CmpiData data) {
    CMPIrc rc = enc->ft->setProperty
    (enc, name, &data.data, data.type, data.count);
    if (rc!=CMPI_RC_OK) throw rc;
}

CmpiObjectPath getObjectPath() {
    CMPIrc rc;
    CmpiObjectPath cop = enc->ft->getObjectPath(enc, &rc);
    if (rc!=CMPI_RC_OK) throw rc;
    return cop;
};

#endif

CmpiSelectExp
TBD

CmpiSelectCond
TBD

CmpiSubCond
TBD

CmpiPredicate
TBD

CmpiArray
TBD

CmpiArgs
TBD

CmpiDateTime
TBD
#ifndef _CmpiResult_h_
#define _CmpiResult_h_

#include "cmpisrv.h"
#include "CmpiData.h"
#include "CmpiInstance.h"
#include "CmpiObjectPath.h"

class CmpiResult {
  protected:
    CMPRIResult *enc;
  private:
    CmpiResult() {}
  public:
    void makeGlobal() {}
    CmpiResult(CMPIResult* r) { enc=r; }
  
    void returnData(CmpIData d) {
      CMPRIrc rc=enc->ft->returnData(enc,&d.data,d.type,d.count);
      if (rc!=CMPI_RC_OK) throw rc;
    }
  
    void returnData(CmpiInstance d) {
      CMPRIrc rc=enc->ft->returnInstance(enc,d.enc);
      if (rc!=CMPI_RC_OK) throw rc;
    }
  
    void returnData(CmpiObjectPath d) {
      CMPRIrc rc=enc->ft->returnObjectPath(enc,d.enc);
      if (rc!=CMPI_RC_OK) throw rc;
    }
  
    void returnDone() {
      CMPRIrc rc=enc->ft->returnDone(enc);
      if (rc!=CMPI_RC_OK) throw rc;
    }

  #endif
CmpiContext

/*
 * CmpiContext.h
 * THIS FILE IS PROVIDED UNDER THE TERMS OF THE COMMON PUBLIC
 * LICENSE ("AGREEMENT"). ANY USE, REPRODUCTION, OR DISTRIBUTION
 * OF THIS FILE CONSTITUTES THE RECIPIENT’S ACCEPTANCE OF THE
 * AGREEMENT.
 * A current copy of the Common Public License is available from:
 * Author: Adrian Schuur <schuur@de.ibm.com>
 * Description: CMPI C++ context provider wrapper
 */

#ifndef _CmpiContext_h_
#define _CmpiContext_h_

#include "cmpisrv.h"
#include "CmpiString.h"
#include "CmpiData.h"
#include "CmpiObjectPath.h"
#include "CmpiInstance.h"

class CmpiContext {
   protected:
      CMPIContext *enc;
   private:
      CmpiContext() {}
   public:
      void makeGlobal() {}
      CmpiContext(CMPIContext* r) { enc=r; }
};
#endif

CmpiEnumeration

TBD
CmpiBroker

/*
 * CmpiBroker.h
 * THIS FILE IS PROVIDED UNDER THE TERMS OF THE COMMON PUBLIC
 * LICENSE ("AGREEMENT"). ANY USE, REPRODUCTION, OR DISTRIBUTION
 * OF THIS FILE CONSTITUTES THE RECIPIENT’S ACCEPTANCE OF THE
 * AGREEMENT.
 * A current copy of the Common Public License is available from:
 * Author: Adrian Schuur <schuur@de.ibm.com>
 * Description: CMPI C++ management broker wrapper
 */

#ifndef __CmpiBroker_h__
#define __CmpiBroker_h__

#include "cmpisrv.h"
#include "CmpiString.h"
#include "CmpiData.h"
#include "CmpiObjectPath.h"
#include "CmpiInstance.h"
#include "CmpiContext.h"

class CmpiBroker {
  protected:
    CmpiBroker *enc;
  private:
    CmpiBroker() {}
  public:
    CmpiBroker(CMPIBroker* b) { enc=b; }
    CmpiInstance newInstance(CmpiObjectPath cop) {
      CMPIrc rc;
      CmpiInstance *inst=enc->eft->newInstance(enc,cop.enc,&rc);
      if (rc!=CMPI_RC_OK) throw rc;
      return CmpiInstance(inst);
    }
    //
    // To be extended with remaining Management Broker methods.
    //
  }

#endif
CmpiInstanceMI

/*
 * CmpiInstanceMI.h
 * THIS FILE IS PROVIDED UNDER THE TERMS OF THE COMMON PUBLIC
 * LICENSE (*AGREEMENT*). ANY USE, REPRODUCTION OR DISTRIBUTION
 * OF THIS FILE CONSTITUTES THE RECIPIENT’S ACCEPTANCE OF THE
 * AGREEMENT.
 * A current copy of the Common Public License is available from:
 * Author: Adrian Schuur <schuur@de.ibm.com>
 * Description: CMPI C++ instance provider wrapper
 */

#ifndef _CmpiInstanceMI_h_
#define _CmpiInstanceMI_h_

#include "cmpisrv.h"
#include "CmpiString.h"
#include "CmpiData.h"
#include "CmpiObjectPath.h"
#include "CmpiInstance.h"
#include "CmpiResult.h"
#include "CmpiContext.h"
#include "CmpiBroker.h"

class CmpiInstanceMI {
protected:
  CmpiInstanceMI *mi;
  CmpiInstanceMI() {}
  CmpiInstanceMI(CmpiInstanceMI&) {}
  CmpiInstanceMI(CMPIInstanceMI* mi, CmpiBroker* mb);
private:
  CmpiBroker *mb;
  CmpiInstanceMI() {}
  CmpiInstanceMI(CMPIInstanceMI* mi, CmpiBroker* mb);
public:
  void initialize();
  void cleanup();
  virtual CMPIrc enumInstanceNames
    (CmpiContext& ctx, CmpiResult& rslt, CmpiObjectPath& cop)=0;
  virtual CMPIrc enumInstances
    (CmpiContext& ctx, CmpiResult& rslt, CmpiObjectPath& cop,
    char* *properties)=0;
  virtual CMPIrc getInstance
    (CmpiContext& ctx, CmpiResult& rslt, CmpiObjectPath& cop,
    char* *properties)=0;
  virtual CMPIrc createInstance
    (CmpiContext& ctx, CmpiResult& rslt, CmpiObjectPath& cop,
    CmpiInstance& inst)=0;
  virtual CMPIrc setInstance
    (CmpiContext& ctx, CmpiResult& rslt, CmpiObjectPath& cop,
    CmpiInstance& inst)=0;
  virtual CMPIrc deleteInstance
    (CmpiContext& ctx, CmpiResult& rslt, CmpiObjectPath& cop)=0;
  virtual CMPIrc execQuery
    (CmpiContext& ctx, CmpiResult& rslt, CmpiObjectPath& cop,
    char* language, char* query)=0;
};
extern "C" {

    CMPIrc driveCleanup
    (CMPIInstanceMI*);
    CMPIrc driveEnumInstanceNames
    (CMPIInstanceMI*, CMPIContext*, CMPIResult*, CMPIOpaqueObjectPath*);
    CMPIrc driveEnumInstances
    (CMPIInstanceMI*, CMPIContext*, CMPIResult*,
     CMPIOpaqueObjectPath*, char**);
    CMPIrc driveGetInstance
    (CMPIInstanceMI*, CMPIContext*, CMPIResult*,
     CMPIOpaqueObjectPath*, char**);
    CMPIrc driveCreateInstance
    (CMPIInstanceMI*, CMPIContext*, CMPIResult*, CMPIOpaqueObjectPath*,
     CMPIInstance*);
    CMPIrc driveSetInstance
    (CMPIInstanceMI*, CMPIContext*, CMPIResult*, CMPIOpaqueObjectPath*,
     CMPIInstance*);
    CMPIrc driveDeleteInstance
    (CMPIInstanceMI*, CMPIContext*, CMPIResult*, CMPIOpaqueObjectPath*);
    CMPIrc driveExecQuery
    (CMPIInstanceMI*, CMPIContext*, CMPIResult*, CMPIOpaqueObjectPath*,
     char*, char*);

};

#endif

CmpiAssociationMI
TBD

CmpiMethodMI
TBD

CmpiPropertyMI
TBD

CmpiIndicationMI
TBD
B.2 Convenience Macros

THIS SECTION IS INCOMPLETE.

/*
cmpimacs.h
Copyright (\(co\) 2002, International Business Machines
* THIS FILE IS PROVIDED UNDER THE TERMS OF THE COMMON PUBLIC
* LICENSE ("AGREEMENT"). ANY USE, REPRODUCTION OR DISTRIBUTION
* OF THIS FILE CONSTITUTES THE RECIPIENT’S ACCEPTANCE OF THE
* AGREEMENT.
* A current copy of the Common Public License is available from:
* Author: Adrian Schuur <schuur@de.ibm.com>
* Description: CMPI Encapsulated types function tables
*/

#ifndef _CMPIMACS_H_
#define _CMPIMACS_H_

// Various return helper macros
#define CMReturn(rc) 
  { CMPIStatus stat={(rc),NULL}; 
    return stat; }

#define CMReturnWithString(rc,str) 
  { CMPIStatus stat={(rc),(str)}; 
    return stat; }

#define CMReturnWithChars(b,rc,chars) 
  { CMPIStatus stat={(rc),NULL}; 
    stat.msg=(b)->eft->newString((b),(chars),NULL)) 
    return stat; }

#define CMSetStatus(st,rcc) 
  { (st)->rc=(rcc); (st)->msg=NULL; }

#define CMSetStatusWithString(st,rcc,str) 
  { (st)->rc=(rcc); (st)->msg=(str); }

#define CMSetStatusWithChars(b,st,rcc,chars) 
  { (st)->rc=(rcc); 
    (st)->msg=(b)->eft->newString((b),(chars),NULL); }

#ifndef CMPI_NO_CONVENIENCE_SUPPORT

#endif CMPI_NO_CONVENIENCE_SUPPORT
// Lifecycle macros

#define CMClone(o,rc) ((o) -> ft->clone((o),(rc)))
#define CMRelease(o) ((o) -> ft->release((o)))

// CMPIBroker factory macros

#define CMNewInstance(b,c,rc) ((b) -> eft->newInstance((b),(c),(rc)))
#define CMNewObjectPath(b,n,c,rc) ((b) -> eft->newObjectPath((b),(n),(c),(rc)))
#define CMNewString(b,s,rc) ((b) -> eft->newString((b),(s),(rc)))
#define CMNewDateTime(b,rc) ((b) -> eft->newDateTime((b),(rc)))
#define CMLoadMi(b,n,rc) ((b) -> eft->loadMI((b),(n),(rc)))
#define CMNewArray(b,c,t,rc) ((b) -> eft->newArray((b),(c),(t),(rc)))
#define CMNewArgs(b,rc) ((b) -> eft->newArgs((b),(rc)))
#define CMNewDateTimeFromBinary(b,d,i,rc) ((b) -> ft->newDateTimeFromBinary((b),(d),(i),(rc)))
#define CMNewDateTimeFromChars(b,d,rc) ((b) -> eft->newDateTimeFromChars((b),(d),(rc)))
#define CMNewSelectExp(b,l,x,rc) ((b) -> ft->*newSelectExp((b),(l),(x),(rc)))

// Debugging macros

#define CToString(b,o,rc) ((b) -> eft->toString((b),(void*)(o),(rc)))
#define CIsOfType(b,o,t,rc) ((b) -> eft->isOfType((b),(void*)(o),(t),(rc)))
#define CGetType(b,o,rc) ((b) -> eft->getTpye((b),(void*)(o),(rc)))

// CMPIInstance macros

#define CMGetProperty(i,n,rc) ((i) -> ft->getProperty((i),(n),(rc)))
#define CMSetProperty(i,n,v,t) ((i) -> ft->setProperty((i),(n),(CMPIValue*)(v),(t)))
#define CMGetPropertyCount(i,c) ((i) -> ft->setPropertyCount((i),(rc)))
#define CMGetObjectPath(i,rc) ((i) -> ft->getObjectPath((i),(rc)))

// CMPIObjectPath macros

#define CMSetNameSpace(p,n) ((p) -> ft->setNameSpace((p),(n)))
#define CMGetNameSpace(p,rc) ((p) -> ft->getNameSpace((p),(rc)))
#define CMSetClassName(p,n) ((p) -> ft->setClassName((p),(n)))
#define CMGetClassName(p,rc) ((p) -> ft->getClassName((p),(rc)))
#define CMAddKey(p,n,v,t) ((p) -> ft->addKey((p),(n),(CMPIValue*)(v),(t)))
#define CMGetKey(p,n,rc) ((p) -> ft->getKey((p),(n),(rc)))
#define CMGetKeyCount(p,rc) ((p) -> ft->getKeyCount((p),(rc)))
#define CMClassPathIsA(p,pn,rc) ((p) -> ft->classPathIsA((p),(pn),(rc)))

// CMPIArray macros

#define CMGetArrayCount(a,rc) ((a) -> ft->getSize((a),(rc)))
#define CMGetArrayType(a,rc) ((a) -> ft->getSimpleType((a),(rc)))
#define CMGetArrayElementAt(a,n,rc) ((a) -> ft->getArrayElementAt((a),(n),(rc)))
((a)->ft->getElementAt((a),(n),(rc)))
#define CMGetArrayElementAt(a,n,v,t) \((a)->ft->setElementAt((a),(n),(CMPIValue*)(v),(t)))

// CMPISelectExp macros

#define CMGetSelExpString(s,rc) ((s) ->ft->getString((s),(rc)))
#define CMEvaluateSelExp(s,i,r) ((s) ->ft->evaluate((s),(i),(r)))
#define CMGetDoc(s,rc) ((s) ->ft->getDOC((s),(rc)))
#define CMGetCod(s,rc) ((s) ->ft->getCOD((s),(rc)))

// CMPISelectCond macros

#define CMGetSubCondCount(c,rc) \((c)->ft->getCountAndType((c),NULL,(rc)))
#define CMGetSubCondCountAndType(c,t,rc) \((c)->ft->getCountAndType((c),(t),(rc)))
#define CMGetSubCondAt(c,p,rc) \((c)->ft->getSubCondAt((c),(p),(rc)))

// CMPISubCond macros

#define CMGetPredicateCount(s,rc) \((s)->ft->getCount((s),(rc)))
#define CMGetPredicateAt(s,p,rc) \((s)->ft->getPredicateAt((s),(p),(rc)))
#define CMGetPredicate(s,n,rc) \((s)->ft->getPredicate((s),(n),(rc)))

// CMPIPredicate macros

#define CMGetPredicateData(p,t,o,n,v) \((p)->ft->getData((p),(t),(o),(n),(v)))

// CMPIDateTime macros

#define CMGetStringFormat(d,rc) \((d)->ft->getStringFormat((d),(rc)))
#define CMGetBinaryFormat(d,rc) \((d)->ft->getBinaryFormat((d),(rc)))
#define CMIsInterval(d,rc) ((d) ->ft->isInterval((d),(rc)))

// CMPIARgs macros

#define CMAddArg(a,n,v,t) \((a)->ft->addArg((a),(n),(CMPIValue*)(v),(t)))
#define CMGetArg(a,n,rc) \((a)->ft->getArg((a),(n),(rc)))
#define CMGetArgAt(a,p,n,rc) \((a)->ft->getArgAt((a),(p),(n),(rc)))
#define CMGetArgCount(a,rc) \((a)->ft->getArgCount((a),(rc)))

// CMPIString macros

#define CMGetCharPtr(s) ((char*)s ->hdl)

// CMPIEnumeration macros

#define CMGetNext(n,rc) ((n) ->ft->getNext((n),(rc)))
#define CMHasNext(n,rc) ((n) ->ft->hasNext((n),(rc)))
// CMPIResult macros
#define CMReturnData(r,v,t)   
   ((r)->ft->returnData((r),(CMPIValue*)(v),(t)))
#define CMReturnInstance(r,i)  
   ((r)->ft->returnInstance((r),(i)))
#define CMReturnObjectPath(r,o)  
   ((r)->ft->returnObjectPath((r),(o)))
#define CMReturnDone(r)      
   ((r)->ft->returnDone((r)))

// CMPIContext macros
#define CMAddContextEntry(c,n,v,t)  
   ((c)->ft->addEntry((c),(n),(CMPIValue*)(v),(t)))
#define CMGetContextEntry(c,n,rc)    
   ((c)->ft->getEntry((c),(n),(rc)))
#define CMGetContextEntryAt(e,p,n,d,rc)    
   ((c)->ft->addEntry((c),(p),(n),(d),(rc)))
#define CMGetContextEntryCount(c,rc)    
   ((c)->ft->getEntryCount((c),(rc)))

// CMPIBroker macros
#define CBGetClassification(b)    
   ((b)->bft->brokerClassification)
#define CBBrokerVersion(b)        
   ((b)->bft->brokerVersion)
#define CBBrokerName(b)           
   ((b)->bft->brokerName)
#define CBPrepareAttachThread(b,c)  
   ((b)->bft->prepareAttachThread((b),(c)))
#define CBAttachThread(b,c)       
   ((b)->bft->attachThread((b),(c)))
#define CBDetachThread(b,c)       
   ((b)->bft->detachThread((b),(c)))
#define CDBdeliverIndication(b,c,n,i,s) 
   (b)->bft->deliverIndication((b),(c),(n),(i),(s))
#define CBEnumInstanceNames(b,c,p,rc)  
   ((b)->bft->enumInstanceNames((b),(c),(p),(rc)))
#define CBEnumInstances(b,c,p,pr,rc)  
   ((b)->bft->enumInstances((b),(c),(p),(pr),(rc)))
#define CBGetInstance(b,c,p,pr,rc)  
   ((b)->bft->getInstance((b),(c),(p),(pr),(rc)))
#define CBCreateInstance(b,p,i,rc)  
   ((b)->bft->createInstance((b),(p),(i),(rc)))
#define CBDeleteInstance(b,p)     
   ((b)->bft->deleteInstance((b),(p)))
#define CBexecQuery(b,p,q,l,c,rc) 
   ((b)->fb->execQuery((b),(q),(l),(c),(rc)))
#define CBAssociators(b,a,p,r,rr,pr,rc) 
   ((b)->bft->associators((b),(a),(p),(rr),(pr),(rc)))
#define CBAssociatorNames(b,a,p,r,rr,rc) 
   ((b)->bft->associatorNames((b),(a),(p),(rr),(rc)))
#define CBInvokeMethod(b,m,ai,ao,rc) 
   ((b)->bft->invokeMethod((b),(m),(ai),(ao),(rc)))
#define CBSetProperty(b,n,v,t) 
   ((b)->bft->setProperty((b),(n),(CMPIValue*)(v),(t)))
#define CBGetProperty(b,n,rc)    
   ((b)->bft->getProperty((b),(n),(rc)))

// MI factory stubs
#define CMNoHook if (brkr)
#define CMInstanceMIStub(cn,pfx,broker,hook) 
   static CMPIInstanceMIFT instMIFT__={ 
      CMPICurrentVersion, 
      (CMPIValue*)(v),(t))
#define CMReturnData(r,v,t)   
   ((r)->ft->returnData((r),(CMPIValue*)(v),(t)))
#define CMReturnInstance(r,i)  
   ((r)->ft->returnInstance((r),(i)))
#define CMReturnObjectPath(r,o)  
   ((r)->ft->returnObjectPath((r),(o)))
#define CMReturnDone(r)      
   ((r)->ft->returnDone((r)))

// CMPIContext macros
#define CMAddContextEntry(c,n,v,t)  
   ((c)->ft->addEntry((c),(n),(CMPIValue*)(v),(t)))
#define CMGetContextEntry(c,n,rc)    
   ((c)->ft->getEntry((c),(n),(rc)))
#define CMGetContextEntryAt(e,p,n,d,rc)   
   ((c)->ft->addEntry((c),(p),(n),(d),(rc)))
#define CMGetContextEntryCount(c,rc)    
   ((c)->ft->getEntryCount((c),(rc)))

// CMPIBroker macros
#define CBGetClassification(b)    
   ((b)->bft->brokerClassification)
#define CBBrokerVersion(b)        
   ((b)->bft->brokerVersion)
#define CBBrokerName(b)           
   ((b)->bft->brokerName)
#define CBPrepareAttachThread(b,c)  
   ((b)->bft->prepareAttachThread((b),(c)))
#define CBAttachThread(b,c)       
   ((b)->bft->attachThread((b),(c)))
#define CBDetachThread(b,c)       
   ((b)->bft->detachThread((b),(c)))
#define CDBdeliverIndication(b,c,n,i,s) 
   (b)->bft->deliverIndication((b),(c),(n),(i),(s))
#define CBEnumInstanceNames(b,c,p,rc)  
   ((b)->bft->enumInstanceNames((b),(c),(p),(rc)))
#define CBEnumInstances(b,c,p,pr,rc)  
   ((b)->bft->enumInstances((b),(c),(p),(pr),(rc)))
#define CBGetInstance(b,c,p,pr,rc)  
   ((b)->bft->getInstance((b),(c),(p),(pr),(rc)))
#define CBCreateInstance(b,p,i,rc)  
   ((b)->bft->createInstance((b),(p),(i),(rc)))
#define CBDeleteInstance(b,p)     
   ((b)->bft->deleteInstance((b),(p)))
#define CBexecQuery(b,p,q,l,c,rc) 
   ((b)->fb->execQuery((b),(q),(l),(c),(rc)))
#define CBAssociators(b,a,p,r,rr,pr,rc) 
   ((b)->bft->associators((b),(a),(p),(rr),(pr),(rc)))
#define CBAssociatorNames(b,a,p,r,rr,rc) 
   ((b)->bft->associatorNames((b),(a),(p),(rr),(rc)))
#define CBInvokeMethod(b,m,ai,ao,rc) 
   ((b)->bft->invokeMethod((b),(m),(ai),(ao),(rc)))
#define CBSetProperty(b,n,v,t) 
   ((b)->bft->setProperty((b),(n),(CMPIValue*)(v),(t)))
#define CBGetProperty(b,n,rc)    
   ((b)->bft->getProperty((b),(n),(rc)))

// MI factory stubs
#define CMNoHook if (brkr)
#define CMInstanceMIStub(cn,pfx,broker,hook) 
   static CMPIInstanceMIFT instMIFT__={ 
      CMPICurrentVersion, 
      (CMPIValue*)(v),(t))
#define CMReturnData(r,v,t)   
   ((r)->ft->returnData((r),(CMPIValue*)(v),(t)))
#define CMReturnInstance(r,i)  
   ((r)->ft->returnInstance((r),(i)))
#define CMReturnObjectPath(r,o)  
   ((r)->ft->returnObjectPath((r),(o)))
#define CMReturnDone(r)      
   ((r)->ft->returnDone((r)))
#define CMAssociationMIStub(cn,pfx,broker,hook) 
static CMPIAssociationMIFT assocMIFT__={ \n    CMPICurrentVersion, \n    CMPICurrentVersion, \n    "association" #cn, \n    pfx##AssociationCleanup, \n    pfx##Associators, \n    pfx##AssociatorNames, \n    pfx##References, \n    pfx##ReferenceNames, \n}; \nEXTERN_C 
CMPIAssociationMI* cn##_Create_AssociationMI(CMPIBroker* 
brkr,CMPIContext *ctx) { \n    static CMPIInstanceMI mi={ \n        NULL, \n        &instMIFT__, \n    }; \n    broker=brkr; \n    hook; \n    return &mi; \n} 
#define CMMethodMIStub(cn,pfx,broker,hook) 
static CMPIMethodMIFT methMIFT__={ \n    CMPICurrentVersion, \n    CMPICurrentVersion, \n    "method" #cn, \n    pfx##MethodCleanup, \n    pfx##InvokeMethod, \n}; \nEXTERN_C 
CMPIMethodMI* cn##_Create_MethodMI(CMPIBroker* 
brkr,CMPIContext *ctx) { \n    static CMPIMethodMI mi={ \n        NULL, \n    };

&methMIFT__, \
); \
broker=brkr; \
hook; \
return &mi; \
}

#define CMPropertyMIStub(cn,pfx,broker,hook) \
static CMIPropertyMIFT propMIFT__={ \
CMPICurrentVersion, \
CMPICurrentVersion, \
"property" #cn, \
pfx##PropertyCleanup, \
pfx##SetProperty, \
pfx##GetProperty, \
}; \
EXTERN_C \
CMIPropertyMI* cn##_Create_PropertyMI(CMPIBroker* \
brkr,CMPIContext *ctx) { \
static CMIPropertyMI mi={ \
NULL, \
 &propMIFT__, \
}; \
broker=brkr; \
hook; \
return &mi; \
}

#define CMIndicationMIStub(cn,pfx,broker,hook)  \
static CMPIIndicationMIFT indMIFT__={ \
CMPICurrentVersion, \
CMPICurrentVersion, \
"Indication" #cn, \
pfx##IndicationCleanup, \
pfx##AuthorizeFilter, \
pfx##MustPoll, \
pfx##ActivateFilter, \
pfx##DeActivateFilter, \
}; \
EXTERN_C \
CMPIIndicationMI* cn##_Create_IndicationMI(CMPIBroker* \
brkr,CMPIContext *ctx) { \
static CMPIIndicationMI mi={ \
NULL, \
 &indMIFT__, \
}; \
broker=brkr; \
hook; \
return &mi; \
}

#endif // CMPI_NO_CONVENIENCE_SUPPORT
#endif // _CMPIMACS_H_
C     Sample Instance MI

THIS SECTION IS INCOMPLETE.

/*
CmpiInstanceMI.cpp
THIS FILE IS PROVIDED UNDER THE TERMS OF THE COMMON PUBLIC
LICENSE ("AGREEMENT"). ANY USE, REPRODUCTION, OR DISTRIBUTION
OF THIS FILE CONSTITUTES THE RECIPIENT’S ACCEPTANCE
OF THE AGREEMENT.
A current copy of the Common Public License is available from:
oss.software.ibm.com/developerworks/opensource/license-cpl.html
* Author: Adrian Schuur <schuur@de.ibm.com>
* Description: CMPI sample MI method drivers
*/

#include "CmpiInstanceMI.h"

extern "C" {
    CMPIRc driveCleanup
    (CmpiInstanceMI* mi)
    {
        return CMPI_RC_OK;
    }

    CMPIRc driveEnumInstanceNames
    (CmpiInstanceMI* mi, CMPIContext* eCtx, CMPIResult* eRslt,
     CMPIObjectPath* eCop)
    {
        CmpiContext ctx(eCtx);
        CmpiResult rslt(eRslt);
        CmpiObjectPath cop(eCop);
        return((CmpiInstanceMI*)mi->hdl)->enumInstanceNames(ctx,rslt,cop);
    }

    CMPIRc driveEnumInstances
    (CmpiInstanceMI* mi, CMPIContext* eCtx, CMPIResult* eRslt,
     CMPIObjectPath* eCop, char* *properties)
    {
        CmpiContext ctx(eCtx);
        CmpiResult rslt(eRslt);
        CmpiObjectPath cop(eCop);
        return ((CmpiInstanceMI*)mi->hdl)->enumInstances
            (ctx,rslt,cop,properties);
    }

    CMPIRc driveGetInstance
    (CmpiInstanceMI* mi, CMPIContext* eCtx, CMPIResult* eRslt,
     CMPIObjectPath* eCop, char* *properties)
    {
        CmpiContext ctx(eCtx);
        CmpiResult rslt(eRslt);
        CmpiObjectPath cop(eCop);
        return ((CmpiInstanceMI*)mi->hdl)->getInstance
            (ctx,rslt,cop,properties);
    }
}
return ((CmpiInstanceMI*)mi->hdl)->getInstance(ctx,rslt,cop,properties);
}

CMPIrc driveCreateInstance
(CMPIInstanceMI* mi, CMPIContext* eCtx, CMPIResult* eRslt,
CMPIObjectPath* eCop, CMPIInstance* eInst)
{
CmpiContext ctx(eCtx);
CmpiResult rslt(eRslt);
CmpiObjectPath cop(eCop);
CmpiInstance inst(eInst);
return ((CmpiInstanceMI*)mi->hdl)->
createInstance(ctx,rslt,cop,inst);
}

CMPIrc driveSetInstance
(CMPIInstanceMI* mi, CMPIContext* eCtx, CMPIResult* eRslt,
CMPIObjectPath* eCop, CMPIInstance* eInst)
{
CmpiContext ctx(eCtx);
CmpiResult rslt(eRslt);
CmpiObjectPath cop(eCop);
CmpiInstance inst(eInst);
return ((CmpiInstanceMI*)mi->hdl)->
setInstance(ctx,rslt,cop,inst);
}

CMPIrc driveDeleteInstance
(CMPIInstanceMI* mi, CMPIContext* eCtx, CMPIResult* eRslt,
CMPIObjectPath* eCop)
{
CmpiContext ctx(eCtx);
CmpiResult rslt(eRslt);
CmpiObjectPath cop(eCop);
return ((CmpiInstanceMI*)mi->hdl)->
deleteInstance(ctx,rslt,cop);
}

CMPIrc driveExecQuery
(CMPIInstanceMI* mi, CMPIContext* eCtx, CMPIResult* eRslt,
CMPIObjectPath* eCop, char* language, char* query)
{
CmpiContext ctx(eCtx);
CmpiResult rslt(eRslt);
CmpiObjectPath cop(eCop);
return ((CmpiInstanceMI*)mi->hdl)->
execQuery(ctx,rslt,cop,language,query);
}
class CmpiTestProvider : public CmpiInstanceMI {
private:
    CmpiTestProvider() {}
    CmpiTestProvider(CmpiTestProvider&) {}
    ~CmpiTestProvider() {}
public:
    CmpiTestProvider(CMPIInstanceMI* ) { mi=p; }
    void initialize();
    void cleanup();
    CMPIrc enumInstanceNames
    (CmpiContext& ctx, CmpiResult& rslt, CmpiObjectPath& cop);
    CMPIrc enumInstances
    (CmpiContext& ctx, CmpiResult& rslt, CmpiObjectPath& cop,
     char* *properties);
    CMPIrc getInstance
    (CmpiContext& ctx, CmpiResult& rslt, CmpiObjectPath& cop,
     char* *properties);
    CMPIrc createInstance
    (CmpiContext& ctx, CmpiResult& rslt, CmpiObjectPath& cop,
     CmpiInstance& inst);
    CMPIrc setInstance
    (CmpiContext& ctx, CmpiResult& rslt, CmpiObjectPath& cop,
     CmpiInstance& inst);
    CMPIrc deleteInstance
    (CmpiContext& ctx, CmpiResult& rslt, CmpiObjectPath& cop);
    CMPIrc execQuery
    (CmpiContext& ctx, CmpiResult& rslt, CmpiObjectPath& cop,
     char* language, char* query);
};

extern "C" {
    static CMPIInstanceMIFT instMIFT={
        CMPIVersion050,
        CMPIVersion050,
        "TestProvider cmpiTestProvider",
        driveCleanup,
        driveEnumInstanceNames,
        driveEnumInstances,
        driveGetInstance,
        driveCreateInstance,
        driveSetInstance,
        driveDeleteInstance,
    };

    CMPIInstanceMI* TestProvider_Create_InstanceMI(CMPIBroker*
broker)
{
    static CMPIInstanceMI mi=
    {NULL,
     &instMI,
    }
    mi.hdl=new CmpiTestProvider(&mi);
    ((CMPIInstanceMI*)(mi.hdl))->mb=new CmpiBroker(broker);
    return &mi;
}

// Routines supporting a simple data store.
int addToStore(CmpiString *k, CmpiString *d);
int getFromStore(CmpiString *k, CmpiString *d),
int remFromStore(CmpiString *k);
CMPIrc CmpiTestProvider::enumInstanceNames
(CmpiContext& ctx, CmpiResult& rslt, CmpiObjectPath& cop)
{
    return CMPI_RC_ERR_NOT_SUPPORTED;
}
CMPIrc CmpiTestProvider::enumInstances
(CmpiContext& ctx, CmpiResult& rslt, CmpiObjectPath& cop,
char* *properties)
{
    return CMPI_RC_ERR_NOT_SUPPORTED;
}
CMPIrc CmpiTestProvider::getInstance
(CmpiContext& ctx, CmpiResult& rslt, CmpiObjectPath& cop,
char* *properties)
{
    CmpiString d,k;
    k=cop.getKey("Identifier").data.string;
    if (getFromStore(&k,&d)) {
        CmpiInstance inst=mb->newInstance(cop);
        inst.setProperty("Identifier",CmpiData(k));
        inst.setProperty("data",CmpiData(d));
        rslt.returnData(inst);
    } else {
        return CMPI_RC_ERR_NOT_FOUND;
    }
    rslt.returnDone();
    return CMPI_RC_OK;
}
CMPIrc CmpiTestProvider::createInstance
(CmpiContext& ctx, CmpiResult& rslt, CmpiObjectPath& cop,
CmpiInstance& inst)
{
    CmpiString k,d;
k=inst.getProperty("Identifier").data.string;

d=inst.getProperty("data").data.string;

if (addToStore(&k,&d)==0) return CMPI_RC_ERR_ALREADY_EXISTS;

rslt.returnData(cop);
return CMPI_RC_OK;

CMPIrc CmpiTestProvider::setInstance
(CmpiContext& ctx, CmpiResult& rslt, CmpiObjectPath& cop,
 CmpiInstance& inst) {
    return CMPI_RC_ERR_NOT_SUPPORTED;
}

CMPIrc CmpiTestProvider::deleteInstance
(CmpiContext& ctx, CmpiResult& rslt, CmpiObjectPath& cop) {
    return CMPI_RC_ERR_NOT_SUPPORTED;
}

CMPIrc CmpiTestProvider::execQuery
(CmpiContext& ctx, CmpiResult& rslt, CmpiObjectPath& cop,
 char* language, char* query) {
    return CMPI_RC_ERR_NOT_SUPPORTED;
}
# Glossary

<table>
<thead>
<tr>
<th>Term</th>
<th>Definition</th>
</tr>
</thead>
</table>


## Index

- `<E-Type>FT.clone` ........................................ 58
- `<E-Type>FT.release` ........................................ 59
- accessing MB services ....................................... 143
- activateFilter .................................................. 42
- array data items .................................................. 4
- association MI .................................................. 18
- association MI signatures ..................................... 31
- associatorNames .................................................. 33
- associators ...................................................... 32
- asymmetric properties ........................................... 5
- authorizeFilter .................................................. 43
- broker services .................................................. 6
- C++ class s ...................................................... 8
- C++ convenience classes ....................................... 160
- CIM ................................................................. 1
- CIM object manager ............................................. 1
- CIM provider ..................................................... 1
- CIMOM ............................................................. 1, 41
- CIMProperty ...................................................... 3
- CIMValue .......................................................... 3
- CMIXML ............................................................ 86, 87
- Class 0 services ................................................ 127
- Class 1 services ................................................ 130
- Class 2 services ................................................ 133
- CMPI ............................................................... 3
- CMPI encapsulated data types ................................ 11
- CMPI interface .................................................... 2
- CMPI miscellaneous data types ................................. 14
- CMPI result data support ....................................... 46
- CMPI return codes ............................................... 46
- CMPI simple data types ........................................... 12
- CMPI string data .................................................. 12
- CMPI types and values ........................................... 14
- CMPI_MIType_xxx ................................................ 19
- CmpiArgs .......................................................... 164
- CmpiArgs support ............................................... 92
- CmpiArgsFT.addButton ......................................... 93
- CmpiArgsFT.getArg .............................................. 94
- CmpiArgsFT.getArgAt ........................................... 95
- CmpiArgsFT.getArgCount ....................................... 96
- CmpiArray .......................................................... 164
- CmpiArray support ............................................... 67
- CmpiArrayFT.getElementAt ..................................... 68
- CmpiArrayFT.getSize ............................................. 69
- CmpiArrayFT.setElementAt ..................................... 70
- CmpiAssociationMI .............................................. 169
- CmpiAssociationMIFT .......................................... 20
- CmpiBroker .......................................................... 167
- CmpiBrokerEncFT.createSelectExp ............................ 106
- CmpiBrokerEncFT.newArgs ..................................... 97
- CmpiBrokerEncFT.newDateTime ................................ 99
- CmpiBrokerEncFT.newDateTimeFromBinary .................... 100
- CmpiBrokerEncFT.newDateTimeFromChars ..................... 101
- CmpiBrokerEncFT.newDateTimeFromDateTime .................. 102
- CmpiBrokerEncFT.newDateTimeFromInt ....................... 103
- CmpiBrokerFT ...................................................... 143
- CmpiBrokerFT.associatorNames ................................ 135
- CmpiBrokerFT.associators ...................................... 134
- CmpiBrokerFT.classification ................................... 128
- CmpiBrokerFT.classpathIsA ..................................... 60
- CmpiBrokerFT.createInstance .................................. 136
- CmpiBrokerFT.deleteInstance ................................... 137
- CmpiBrokerFT.deliverIndication ................................ 129
- CmpiBrokerFT.enumInstanceNames ............................... 131
- CmpiBrokerFT.enumInstances ................................... 138
- CmpiBrokerFT.getInstance ...................................... 132
- CmpiBrokerFT.getProperties .................................... 139
- CmpiBrokerFT.getType ............................................ 61
- CmpiBrokerFT.invokeMethod .................................... 140
- CmpiBrokerFT.isOfType .......................................... 62
- CmpiBrokerFT.logMessage ...................................... 63
- CmpiBrokerFT.setInstance ...................................... 141
- CmpiBrokerFT.setProperty ...................................... 142
- CmpiBrokerFT.toString ......................................... 64
- CmpiContext ..................................................... 166
- CmpiContextFT.addEntry ....................................... 53
- CmpiContextFT.getEntry ........................................ 54
- CmpiContextFT.getEntryAt ....................................... 55
- CmpiData .......................................................... 161
- CmpiData .......................................................... 14
- CmpiDateTime .................................................... 164
- CmpiDateTime.support ........................................... 98
- CmpiDateTime.getTime ............................................ 99
- CmpiDateTime.getTimeFromBinary ................................ 100
- CmpiDateTime.getTimeFromChars ................................ 101
- CmpiDateTime.getTimeFromDateTime ......................... 102
- CmpiDateTime.getTimeFromInt .................................. 103
- CmpiDateTime.isInterval ....................................... 104
- CmpiDateTime.getTimeFromNative ................................ 145
- CmpiEnumeration ............................................... 166
- CmpiEnumeration.support ....................................... 71
- CmpiEnumerationFT.getNext .................................... 72
- CmpiEnumerationFT.hasMore ................................... 73
- CmpiEnumerationFT.toArray ..................................... 74
- CmpiFlags .......................................................... 23
- cmptft.h .......................................................... 151
- CmpiIndicationMI ............................................... 169
- CmpiIndicationMIFT ............................................. 21
- CmpiInstance ..................................................... 163

Systems Management: Common Management Programming Interface (CMPI)

- CMPIType: 4, 14
- CMPIValue: 4, 16
- CMPIValueState: 4, 16
- CMPIxxxMIFT.cleanup: 22
- CoD: 9
- context data support: 52
- convenience macros: 170
- createInstance: 24
- data transfer: 3
- data type manipulation: 56
- data types: 11, 145
- deActivateFilter: 44
- deleteInstance: 25
- DoC: 10
- encapsulated data type: 3
- encapsulated types: 6
- encapsulation: 3
- enumeration: 5
- enumInstanceNames: 26
- enumInstances: 27
- error indication: 4
- execQuery: 28
- function tables: 151
- getElementAt: 4
- get Instance: 29
- getProperties: 37
- header files: 145
- helper class: 9
- indication filtering: 9
- indication MI: 18
- indication MI signatures: 41
- instance MI: 18
- instance MI signatures: 23
- invokeMethod: 40
- lifecycle operations: 3
- macro support: 7
- management broker: 1
- management instrumentation: 1
- MB services: 126
- memory ownership: 5
- method MI: 18
- method MI signatures: 39
- MI: 18
- MI convenience support: 160
- MI factory: 19
- miscellaneous services: 57
- mustPoll: 45
- null value specification: 17
- OpenCimom: 9, 41
- Pegasus: 5
- programming convenience support: 7
- property MI: 18
- property MI signatures: 36
- provider registration: 18
- qualifier support: 120
- query: 9
- referenceNames: 35
- references: 34
<table>
<thead>
<tr>
<th>Requirement</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>Requirements</td>
<td>1</td>
</tr>
<tr>
<td>Sample instance MI</td>
<td>176</td>
</tr>
<tr>
<td>Schema support</td>
<td>125</td>
</tr>
<tr>
<td>SetElementAt</td>
<td>4</td>
</tr>
<tr>
<td>SetInstance</td>
<td>30</td>
</tr>
<tr>
<td>SetProperty</td>
<td>38</td>
</tr>
<tr>
<td>Simple data items</td>
<td>4</td>
</tr>
<tr>
<td>String data types</td>
<td>12</td>
</tr>
<tr>
<td>Sun WBEM</td>
<td>9</td>
</tr>
<tr>
<td>Threading</td>
<td>5</td>
</tr>
<tr>
<td>Thread-safe</td>
<td>5</td>
</tr>
<tr>
<td>UTF-8</td>
<td>12</td>
</tr>
<tr>
<td>WBEMS Services</td>
<td>41</td>
</tr>
</tbody>
</table>