The Java™ API for Web Based Enterprise Management

JSR-48

Version Final

October 26, 2009

Abstract

Web Based Enterprise Management (WBEM) is a set of specifications that unify the management of enterprise computing environments. WBEM provides the ability for the industry to deliver a well-integrated set of standard-based management tools leveraging the emerging Web technologies. The Distributed Management Task Force (DMTF) has developed a core set of standards that make up WBEM.

This specification along with the Javadoc defines the Java™ WBEM API. This allows any Java™ VM to become WBEM enabled as well as allow any Java™ VM to manage any WBEM (SMI, SMASH, ...) enabled managed elements.

Table of Contents

Introduction

Java™ WBEM Services is an architecture and a set of programming interfaces based on CIM and WBEM standards. This specification deals specifically with the CIM, Client and Provider programming interfaces.

The Java™ WBEM API (javax.wbem and javax.cim) is based on CIM and WBEM standards and allows any Java™ platform to manage any WBEM enabled element as well as enables any Java™ platform to become WBEM enabled.

Web-Based Enterprise Management (WBEM) is an industry initiative. WBEM includes standards for managing systems, networks, users, and applications by using Internet standard technologies. WBEM provides a way for management applications to share management data independently of vendor, protocol, operating system, or management standard. WBEM consists of the following standards:

- The Common Information Model (CIM) is an object-oriented information model defined by the Distributed Management Task Force (DMTF) which provides a conceptual framework for describing management data.
- CIM-XML
- xmlCIM is the standard representation of CIM using XML.
- CIM Operations over HTTP defines the mapping that uses xmlCIM and HTTP

The DMTF is the leading industry organization for development, adoption and unification of management standards for distributed desktop, applications, network, enterprise and internet environments. Working with key technology vendors and affiliated standards groups, the DMTF is enabling a more integrated, cost-effective and less crisis-driven approach of management through interoperable management solutions. For information about DMTF standards and initiatives, see the DMTF web site at http://www.dmtf.org.

The complete Java WBEM API specification is composed of this document along with the WBEM API Javadoc.

Audience

The intended audience for this specification is:

- Java™ developers who want to manage WBEM enabled elements.
- Java™ VM, Operating System, and device providers that want to develop WBEM (SMASH, SMI) enabled products.
- The JSR-48 Expert Group.

Terminology

The key phrases and words MUST, MUST NOT, REQUIRED, SHALL, SHALL NOT, SHOULD, SHOULD NOT, RECOMMENDED, MAY and OPTIONAL in this document are to be interpreted as described in RFC 2119[1].

1 "Key words for use in RFCs to Indicate Requirement Levels", IETF RFC 2119, March 1997 (http://www.ietf.org/rfc/rfc2119.txt)

Requirements

Java Editions

JSR 48 is targeted at both the J2SE (Java 2 Standard Edition) and J2ME (Java 2 Micro Edition) platforms. Many management platform vendors, operating system providers, and device manufacturers are supporting WBEM as the management platform of choice. They require support for both J2SE and J2ME.

Standards

The following table lists the standards and versions that JSR48 supports.

Specification	Version	Organization
CIM Infrastructure Specification	2.5.0	DMTF
CIM Operations over HTTP	1.3.0	DMTF
CIM Representation using XML	2.3.0	DMTF
CIM XML DTD	2.3.0	DMTF

Architecture

Overview

This part of the specification is targeted towards developers who want to understand the mappings used to create the Java API from the CIM and WBEM specifications. This section does not repeat the information in the CIM specification, so the prerequisite is that the developer should already have an understanding of CIM and WBEM.

Java Packages

The Java WBEM API is broken down into the following packages:

`javax.cim`	Java representation of CIM provides classes and interfaces for handling the CIM (Common Information Model) Object Model and CIM Data Types as defined in the CIM Infrastructure Specification.
`Javax.wbem`	Common classes and interfaces across all WBEM APIs provide classes and interfaces that are common across all WBEM APIs.
`javax.wbem.client`	Mapping of CIM Operations provides classes and interfaces for writing WBEM Clients.
`javax.wbem.listener`	Listen for indications (events) provides classes and interfaces for writing WBEM Listeners.
`javax.wbem.provider`	Provider Interfaces provides classes and interfaces for writing WBEM Providers.

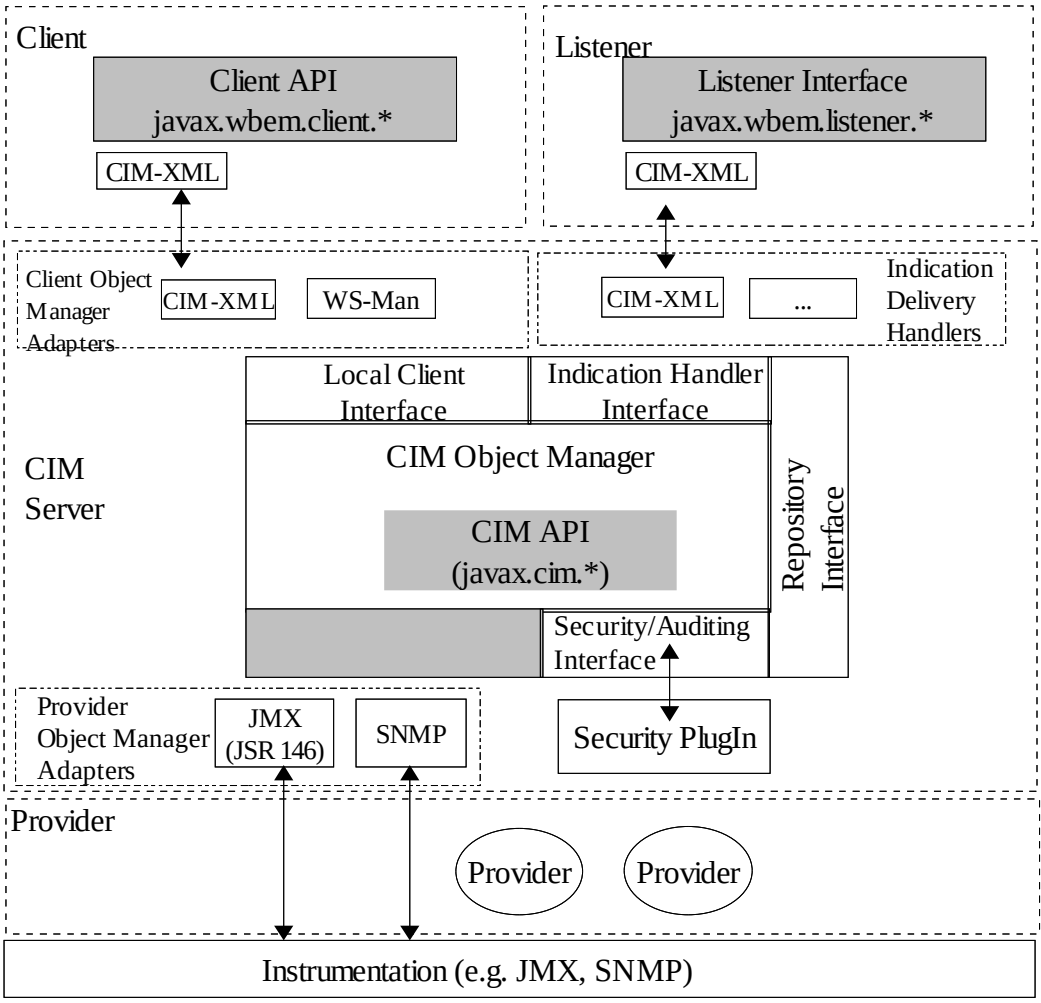
JSR48 Architecture

This section will describe the high level JSR48 architecture. It will include a high level architecture diagram, a brief description of each component, programming interface and protocol. For this release of the specification the CIM, Client, and Provider APIs are being standardized. Future versions of this specification may include other interfaces.

The JSR48 architecture is broken down into three layers:

1. Client - Consumer of WBEM information
2. WBEM Server - Agent Tier/Server
3. Provider - Supplier of WBEM information

The following is a high level architecture block diagram of all of components, interfaces and protocols. Note that the names/titles for protocol adapters and indication delivery handlers are examples only.



Clients

A JSR48 Client is an application or service that retrieves information originating from one or more managed elements using the Java WBEM Client API. Clients can vary from command line interfaces (CLI), Graphical User Interface (GUI) applications and web based consoles to automated services.

WBEM Server

The following sections describe the components that make up the JSR48 WBEM Server.

Client Object Manager Adapters

A *Client Object Manager Adapter* (COMA) is a pluggable module of the CIM Server that :-

1. Accepts incoming requests through a particular protocol
2. Translates these calls for the CIM Object Manager (CIMOM)
3. Accepts responses from the CIM Object Manager to send back to the client.

JSR48 COMA MUST support all of the intrinsic operations for basic read. JSR48 COMA MUST support the operations based on the functional profiles as defined in the CIM Operations over HTTP specification. This means that a COMA MUST support all operations of a functional profile or none of them.

JSR48 implementation MUST include a CIM-XML COMA. Any additional COMA MAY be included.

Indication Delivery Handlers

An *Indication Delivery Handler* (IH) is a pluggable module of the CIM Server that delivers indications through a specified means (e.g. e-mail, pager, etc.). Indication Delivery Handlers are responsible for receiving an indication from the CIM Object Manager and delivering it to the specified destination.

CIM Object Manager

A CIM Object Manager (CIMOM) is the central component of the JSR48 WBEM Server. It is responsible for the communication between the clients, providers and CIM repository. Since it communicates to clients and providers through object manager adapters, it is required to register, start, stop, and maintain both client and provider object manager adapters. Another responsibility of the CIM Object Manager is authentication, authorization and auditing through interaction with the Security/Auditing PlugIn.

CIM Repository

The *CIM Repository* is responsible for storing (persistent) schema information and delivery of schema modification indications. The repository can optionally also store (i.e., allow for storage of) instance information and handle delivery of class and instance indications.

The CIM Repository only interfaces with the CIM Object Manager, this means that clients, providers and other agent infrastructure modules can only access this information through the CIM Object Manager.

Security/Auditing PlugIn

The Security/Auditing PlugIn is responsible for authentication and auditing. When the CIM Object Manager receives a request for any of these, it forwards the requests to the Security/Auditing PlugIn and waits for a response. This PlugIn is responsible for handling the request and responding back to the CIM Object Manager. This PlugIn only interacts with the CIM Object Manager.

Provider Object Manager Adapter

A *Provider Object Manager Adapter* (POMA) is a pluggable module of the CIM Server that accepts requests from the CIM Object Manager and forwards them to the provider, receives the response from the provider and passes the response back to the CIM Object Manager.

Providers

A provider is an intermediary between the CIM Object Manager and one or more managed elements. It is responsible for accepting requests from the CIM Object Manager (through a POMA), getting the information from the managed element and responding with the result. All providers get a handle to the CIM Object Manager so that they can also act as a client.

JSR48 supports the following four provider types:

Provider Type	Purpose
Association Provider	for dynamic associations instances
Indication Provider	for dynamic indications (events)
Instance Provider	for dynamic instances of CIM classes
Method Provider	for extrinsic methods on a CIM class
Pull Association Provider	for pulled dynamic associations instances
Pull Instance Provider	for pulled dynamic instances of CIM classes

Design

The JSR48 API consists of a set of Java packages. Each package represents a different aspect of the JSR48 Architecture. Following packages have been defined:

Java Package Name	Purpose
javax.cim	represents APIs to use for CIM elements
javax.wbem	contains common interfaces for handling errors and operations
javax.wbem.client	represents APIs to use to perform client operations
javax.wbem.provider	represents APIs used to support interaction between the CIMOM and providers
javax.wbem.listener	represents APIs used to support indications

CIM (javax.cim)

The CIM API is a mapping to Java of the CIM elements described in the CIM Infrastructure Specification[2]. The CIM meta schema is the formal definition of the CIM Model. It defines terms used to express the model, usage and semantics. The CIM Infrastructure Specification also describes other CIM elements that are not part of the meta schema.

The following sections will describe Java representation for each of these elements. However, this section will not describe the API. For detailed information on the API, please consult the JSR48 Javadoc.

2 CIM Infrastructure Specification – http://www.dmtf.org/standards/published_documents/DSP0004_2_5.0.pdf

CIM Meta Elements

All meta elements in CIM are expressed as a descendant of Named Element. A Named Element has only one property – name, which means all meta elements must have a name. The name is case-insensitive.

In Java, the class CIMElement represents the Named Element. CIMElement is an abstract base class used for all Java CIM meta elements.

The following table lists the CIM Meta Schema Elements and its corresponding Java Class representation.

CIM Meta Schema Element	Java Class (javax.cim.*)
Class	CIMClass
Association	CIMClass
Indication	CIMClass
Property	CIMClassProperty
Qualifier Type	CIMQualifierType
Qualifier	CIMQualifier
Qualifier Flavor	CIMFlavor
Qualifier Scope	CIMScope
Reference	CIMObjectPath
Method	CIMMethod
Parameter	CIMParameter

Qualifier

A qualifier provides additional information about other meta schema elements. Depending upon its definition, a qualifier can be applied to classes, associations, indications, methods, parameters, properties or references.

The CIMQualifier class is the Java representation of a qualifier. Its value represents the value of the qualifier.

Qualifier Type

Before a qualifier can be used, its qualifier type must be defined in the target namespace. A qualifier type defines

- The qualifier data type (e.g., string, integer, etc.)
- The meta schema elements to which the qualifier can be applied
- Whether the qualifier is inherited by a derived class
- Whether the qualifier can be overridden in a derived class

Class

A class represents a blueprint or prototype of a managed object. Its definition must contain a name. Optionally, it can define

- A superclass
- One or more properties
- One or more methods
- One or more qualifiers

The CIMClass class is the Java representation for a class.

Association

An association is a specialized class that contains references to other classes.

The CIMClass class is the Java representation for an association.

Indication

An indication is a specialized class that represents an occurrence of an event.

The CIMClass class is the Java representation for an indication.

Class Property

A class property represents the definition of an attribute in a class definition. Its definition must contain a name and data type. Optionally, a property definition within a class may specify qualifiers. This default value is assigned to the property when the class is instantiated.

The CIMClassProperty class is the Java representation for a property definition within a class.

Reference

A reference is a specialized property that is used to refer to another class or instance. A fully populated reference consists of the following components:

- scheme

- host name
- port
- namespace
- class name
- keys

The CIMObjectPath class is the Java representation for a reference. When referred to an instance, keys must be populated.

Method

A method represents a specific operation that can be performed on an instance of the class in which the method is defined. Its definition must contain a name and data type. Optionally, a method definition within a class may specify qualifiers and parameters.

The CIMMethod class is the Java representation for a method definition within a class.

Parameter

A parameter represents the definition of an argument for a method in a class definition. A parameter definition must contain a name and data type. Optionally, it may also specify qualifiers.

The CIMMethod class is the Java representation for a parameter definition for a method within a class.

Qualifier Flavor

Qualifiers can be inherited from classes to derived classes. The rules for inheritance to derived classes are attached to each qualifier and encapsulated in the concept of qualifier flavor.

Additionally, qualifiers can be overridden in derived classes. The rules for overriding are also attached to each qualifier and encapsulated in the concept of the qualifier flavor.

The CIMFlavor class is the Java representation of a CIM Flavor.

Qualifier Scope

Qualifiers can be applied to all different CIM meta elements. Some qualifiers can be applied to ANY of the CIM meta elements, while others only make sense when they are applied to particular meta elements (for example, a property). This is called the scope in which the qualifier can be applied. When each qualifier type is defined, the scope for that qualifier type is also defined.

The CIMScope class is the Java representation for a CIM Scope.

Other CIM Elements

The following table lists other CIM Elements that are not part of the meta schema but have corresponding Java Class representation.

Other CIM Elements	Java Class (javax.cim.*)
Instance	CIMInstance
Instance Property	CIMProperty
Argument	CIMArgument

Instance

A class definition represents a blueprint or prototype of a managed object containing one or more properties. However, property definitions do not contain actual values.

In contrast, an instance represents an instantiation of a class where the properties may contain values.

The CIMInstance class is the Java representation for an instance.

Instance Property

A class definition represents a blueprint or prototype of a managed object that contains one or more property definitions. However, property definitions do not contain values.

When a class is instantiated, its properties defined in the class are also instantiated. An instance property represents an instantiation of a class property. In contrast to a class property, an instance property can contain a value.

The CIMProperty class is the Java representation for an instance property.

Argument

A method represents a specific operation that can be performed on an instance of the class in which the method is defined. A parameter represents the definition of an input and/or output variable for a method. A parameter definition does not contain a value.

An argument represents the instantiation of a parameter for a method. In contrast to parameter, an argument can contain a value.

A CIMArgument class is the Java representation of an argument for a method.

Data Types

The CIM Infrastructure Specification defines the following data types. The CIMDataType class is the Java representation for a CIM data type.

The following table shows the mapping of a CIM data types to its corresponding Java class. Many CIM data types can be represented by a standard Java class. For example, a signed 8-bit integer CIM data type can be represented by the Java Byte class.

However, some CIM data types required creating a new Java class as part of the JSR48 CIM API. These Java class names are shown using **bold** font.

CIM Data Type	Description	Java
uint8	Unsigned 8-bit integer	UnsignedInteger8
sint8	Signed 8-bit integer	Byte
uint16	Unsigned 16-bit integer	UnsignedInteger16
sint16	Signed 16-bit integer	Short
uint32	Unsigned 32-bit integer	UnsignedInteger32
sint32	Signed 32-bit integer	Integer
uint64	Unsigned 64-bit integer	UnsignedInteger64
sint64	Signed 64-bit integer	Long
string	UCS-2 string	String
boolean	Boolean	Boolean
real32	IEEE 4-byte floating-point	Float
real64	IEEE 8-byte floating-point	Double
datetime	A string containing a date-time	CIMDateTime CIMDateTimeAbsolute CIMDateTimeInterval
reference	Strongly typed reference	CIMObjectPath
char16	16-bit UCS-2 character	Character

CIMDateTime

The CIMDateTime class is the Java representation for the CIM datetime data type. This datatype has two forms. The CIMDateTimeAbsolute class is derived from CIMDateTime and is used when the datetime represents a time stamp. The CIMDateTimeInterval is also derived from CIMDateTime and is used when the datetime represents a time interval.

Common Java Elements (javax.wbem)

This section describes the common interfaces and classes used by the other APIs. For detailed information on these interfaces and classes, please consult the JSR48 Javadoc.

Java Class	Purpose
`CloseableIterator`	`Subclass of Iterator that adds support allowing the underlying implementation to serve up the CIM elements as they become available`
`WBEMException`	`Returned when there is a WBEM Operations error`

A CloseableIterator is used by other WBEM operation APIs to return a set of Java CIM objects. For operations that might return a large number of CIM objects, instead of returning all requested objects to the client at the same time, CloseableIterator is used to return them, a portion at a time, as they become available. The client can use the Iterator methods next() and hasNext() to control the flow of incoming objects.

Client (javax.wbem.client)

The Client API is a mapping of the operations listed in the CIM Operations over HTTP specification[3] by the DMTF. This API allows a client to send a request to perform a single CIM operation.

For each CIM operation specified in the CIM Operations over HTTP specification there is a corresponding Java method(s) in the WBEMClient class. Each section that follows shows the Java method(s) that map to the CIM Operations.

For detailed information about the behavior and the arguments defined for a CIM operation consult the CIM Operations over HTTP specification. For detailed information about the behavior and the arguments defined for the corresponding JSR48 API consult the JSR48 javadoc.

3 http://www.dmtf.org/standards/published_documents/DSP0200.html

Connecting to a CIM Server

A client that wishes to perform a CIM operation must first establish a connection to a target namespace on a server. Establishing a connection involves the following Java classes:

Java Class	Purpose
WBEMClientFactory	Factory for the WBEMClient implementation using a specified protocol
WBEMClient	Interface used by a client to perform CIM operations on a server
EnumerateResponse	A container that stores the information from a Pull request
UserPrincipal	Represents the user identity to be used for authentication when establishing a connection to a server
PasswordCredential	Represents the password based credential to be used for authentication when establishing a connection to a server
RolePrincipal	Represents the role name to be used for checking the authorization of a connected user
RoleCredential	Represents the password based credential to be used for checking the authorization of a connected user
WBEMCLientConstants	Defines the constants used for a WBEMClient configuration

The following code fragment illustrates their use in establishing a connection. For detailed information about this class consult the JSR48 javadoc.

```
Subject subject = new Subject();
subject.getPrincipals().add(new UserPrincipal(up));
subject.getPrivateCredentials().add(new PasswordCredential(pc));
subject.getPrincipals().add(new RolePrincipal(rp));
subject.getPrivateCredentials().add(new RoleCredential(rc));
cimClient = WBEMClientFactory.getClient("CIM-XML");
cimClient.initialize(new CIMObjectPath(name), s, null);
```

NOTE: The WBEMClientConstants class must be implemented. For

CIM Operations

The client API allows a client to perform a single CIM operation on metadata or data objects in a target namespace on a WBEM Server. They can be grouped into the following categories:

CIM Operations Categories	Purpose
Metadata	Allows a client to perform operations on classes and qualifier types
Data	Allows a client to perform operations on instances
Association Traversal	Allows a client to perform operations on association classes and to traverse between classes using associations.
Query	Allows a client to query for classes or instances
Extrinsic Method	Allows a client to perform a defined extrinsic method of a class

When a large number of objects might be returned, a different mechanism called Pull operations can be used to improve efficiency. For a Pull operation, an enumeration session is opened with certain parameters that control the return of data. Data is returned in one or more Pull operations. When all data has been returned, then the enumeration session is automatically closed by the server.

For detailed information about the behavior and the arguments defined for a Pull operation consult the CIM Operations over HTTP specification. For detailed information about the behavior and the arguments defined for the corresponding JSR48 API consult the JSR48 javadoc.

Metadata Operations

The metadata operations allow a client to manage metadata information relating to classes and qualifier types using the appropriate Java APIs.

CIM Operations	Java WBEM API (javax.wbem.client.WBEMClient)
CreateClass	createClass
GetClass	getClass
ModifyClass	setClass
DeleteClass	deleteClass
EnumerateClassNames	enumerateClassNames
EnumerateClasses	enumerateClasses
GetQualifier	getQualifierType
SetQualifier	setQualifierType
DeleteQualifier	deleteQualifierType
EnumerateQualifiers	enumQualifierTypes

The following table shows the mapping of argument names between the CIM Operations over HTTP specification and the JSR48 client API

CIM Operations over HTTP Specification	Java WBEM API (javax.wbem.client.WBEMClient)
ClassName	path or name
DeepInheritance	deep
LocalOnly	localOnly
IncludeQualifiers	includeQualifiers
IncludeClassOrigin	includeClassOrigin
QualifierName	path or name
QualifierDeclaration	qt
ModifiedClass or NewClass	cc

Data Operations

The data operations allow a client to manage data relating to instances using the appropriate Java APIs.

CIM Operations	Java WBEM API (javax.wbem.client.WBEMClient)
`CreateInstance`	`createInstance`
`GetInstance`	`getInstance`
`ModifyInstance`	`setInstance`
`DeleteInstance`	`deleteInstance`
`EnumerateInstanceNames`	`enumerateInstanceNames`
`EnumerateInstances`	`enumerateInstances`
`GetProperty`	`getProperty`
`SetProperty`	`setProperty`

Use the following methods when the data operations return data via the Pull mechanism

Pulled CIM Operations	Java WBEM API (javax.wbem.client.WBEMClient)
`OpenEnumerationInstances`	`enumerateInstances`
`OpenEnumerationInstancePaths`	`enumerateInstancePaths`
`PullInstancesWithPath`	`getInstancesWithPath`
`PullInstancePaths`	`getInstancePath`
`CloseEnumeration`	`closeEnumeration`
`EnumerationCount`	`enumerationCount`

The following table shows the mapping of argument names between the CIM Operations over HTTP specification and the JSR48 client API

CIM Operations over HTTP Specification	Java WBEM API (javax.wbem.client.WBEMClient)
ClassName	path
InstanceName	path or name
DeepInheritance	deep
LocalOnly	localOnly
IncludeQualifiers	includeQualifiers
IncludeClassOrigin	includeClassOrigin
PropertyList	propertyList
ModifiedInstance or NewInstance	ci

The following table shows the additional mapping of argument names between the CIM Operations over HTTP specification and the JSR48 client API when using Pull operation sessions

CIM Operations over HTTP Specification	Java WBEM API (javax.wbem.client.WBEMClient)
EnumerationContext	enumerationContext
MaxObjectCount	maxObjects
OperationTimeout	timeout
ContinueOnError	continueOnError
FilterQueryLanguage	filterQueryLanguage
FilterQuery	filterQuery
EndOfSequence	end

Association Traversal Operations

The association traversal operations allow a client to manage information relating to associations using the appropriate Java APIs.

Use the following operations, when the returned information can be metadata (i.e., classes) or data (i.e., instances).

CIM Operation	Java WBEM API (javax.wbem.client.WBEMClient)
AssociatorNames	associatorNames
Associators	associators
ReferenceNames	referenceNames
References	references

Use the following operations when the managed information can only be metadata; i.e., classes.

CIM Operation	Java WBEM API (javax.wbem.client.WBEMClient)
AssociatorNames	associatorNames
Associators	associatorClasses
ReferenceNames	referenceNames
References	referenceClasses

Use the following operations when the managed information can only be data; i.e., instances.

CIM Operation	Java WBEM API (javax.wbem.client.WBEMClient)
AssociatorNames	associatorNames
Associators	associatorInstances
ReferenceNames	referenceNames
References	referenceInstances

Use the following operations when the association traversal operations return data via the Pull mechanism

Pulled CIM Operations	**Java WBEM API (javax.wbem.client.WBEMClient)**
OpenAssociatorInstancePaths	**associatorPaths**
OpenAssociatorInstances	associators
OpenReferenceInstancePaths	referencePaths
OpenReferenceInstances	references
PullInstancesWithPath	getInstancesWithPath
PullInstancePaths	getInstancePaths
CloseEnumeration	closeEnumeration
EnumerationCount	enumerationCount

The following table shows the mapping of argument names between the CIM Operations over HTTP specification and the JSR48 client API

CIM Operations over HTTP Specification	Java WBEM API (javax.wbem.client.WBEMClient)
ObjectName	objectName
AssocClass	assocClass
ResultClass	resultClass
Role	role
ResultRole	resultRole
IncludeQualifiers	includeQualifiers
IncludeClassOrigin	includeClassOrigin
PropertyList	propertyList

The following table shows the additional mapping of argument names between the CIM Operations over HTTP specification and the JSR48 client API when using Pull operation sessions

Pulled CIM Operations	Java WBEM API (javax.wbem.client.WBEMClient)
EnumerationContext	enumerationContext
MaxObjectCount	maxObjects
OperationTimeout	timeout
ContinueOnError	continueOnError
FilterQueryLanguage	filterQueryLanguage
FilterQuery	filterQuery
EndOfSequence	end

Query

The query operation allows a client to retrieve information from a target namespace using the appropriate Java APIs.

Use the following operations, when the returned information can be metadata (i.e., classes) or data (i.e., instances).

CIM Operation	Java WBEM API (javax.wbem.client.WBEMClient)
`ExecQuery`	`execQuery`

Use the following operations when the Query operation returns data using the Pull mechanism

Pulled CIM Operations	Java WBEM API (javax.wbem.client.WBEMClient)
`OpenQueryInstances`	`execQueryInstances`
`PullInstances`	`getInstances`
`CloseEnumeration`	`closeEnumeration`
`EnumerationCount`	`enumerationCount`

The following table shows the mapping of argument names between the CIM Operations over HTTP specification and the JSR48 client API

CIM Operations over HTTP Specification	Java WBEM API (javax.wbem.client.WBEMClient)
QueryLanguage	qLanguage
Query	query

The following table shows the additional mapping of argument names between the CIM Operations over HTTP specification and the JSR48 client API when using Pull operation sessions

Pulled CIM Operations	Java WBEM API (javax.wbem.client.WBEMClient)
EnumerationContext	enumerationContext
MaxObjectCount	maxObjects
OperationTimeout	timeout
ContinueOnError	continueOnError
FilterQueryLanguage	filterQueryLanguage
FilterQuery	filterQuery
EndOfSequence	end

Extrinsic Method Invocation

Extrinsic operations allow a client to perform private operations defined for a class.

CIM Operation	Java WBEM API (javax.wbem.client.WBEMClient)
`Extrinsic Method Invocation`	`invokeMethod`

The actual behavior and list of input and output arguments are specified by the method definition in the class.

Listener (javax.wbem.listener)

The DMTF specification for indications supports subscription of indications and indication delivery. Clients perform event subscriptions by creating instances of filters, handlers and subscriptions. These instances specify indication conditions, handlers specify destination where generated indications are delivered and subscriptions associate filter and handler instances. The white paper on DMTF Events describes this in more detail.

Subscriptions are performed using intrinsic create instance calls, apart from that there are no specific methods for creating subscriptions on a remote CIM object manager. However, additional JSR48 APIs have been defined to support indications which are outside the scope of the DMTF events specification.

For detailed information about the behavior and the arguments defined for these JSR48 Listener APIs consult the JSR48 javadoc.

Providers (javax.wbem.provider)

In JSR48 architecture, a provider is an intermediary between the CIM Object Manager (CIMOM) and one or more managed elements. Metadata operations are handled entirely by the CIMOM. Association traversal operations for classes are also handled entirely by the CIMOM. No providers are invoked to handle such operations.

All other client operations (e.g., data, extrinsic methods, etc.) are handled by providers. A provider accepts the request from the CIMOM, retrieves the information from the managed object and responds to the CIMOM with the result.

The Provider class is the Java class representation for providers. Classes are derived from this base Provider class that represents various types of provider interfaces. Each of the following provider interfaces supports a certain set of client operations.

Provider Interfaces	Function
`InstanceProvider`	`Handles data operations`
`AssociatorProvider`	`Handles association traversal operations on instances`
`IndicationProvider`	`Handles the indication mechanism`
`MethodProvider`	`Handles the extrinsic method invocation operation on an instance`
`PullInstanceProvider`	`Handles pulled data operations`
`PullAssociatorProvider`	`Handles pulled association traversal operations`

For each CIM operation specified in the CIM Operations over HTTP specification that is handled by a provider, there is a corresponding Java method or methods. The following sections show the Java method(s) that map to the CIM Operations for each provider type.

For detailed information about the behavior and the arguments defined for a CIM operation consult the CIM Operations over HTTP specification. For detailed information about the behavior and the arguments defined for the corresponding JSR48 API consult the JSR48 javadoc.

Provider

The Provider class is the base Java class for other types of provider interfaces. It defines the following methods that are inherited by other provider interfaces:

Java WBEM API (javax.wbem.provider.*)	Function
`initialize`	`called by the CIMOM the first time the provider is invoked to perform any initialization steps`
`close`	`called by the CIMOM when the provider is unloaded`

InstanceProvider

The InstanceProvider interface is used to support the client data and query operations. Note that client queries about metadata will be handled by the CIMOM. Only client queries about data will be handled by a provider.

CIM Operations	Java WBEM API (javax.wbem.provider.*)
CreateInstance	createInstance
GetInstance	getInstance
ModifyInstance	modifyInstance
DeleteInstance	deleteInstance
EnumerateInstanceNames	enumerateInstanceNames
EnumerateInstances	enumerateInstances
GetProperty	getProperty
SetProperty	setProperty
ExecQuery	execQuery

The following table shows the mapping of argument names between the CIM Operations over HTTP specification and the JSR48 provider API.

The IncludeQualifiers argument is not mapped because instances do not have qualifiers and an InstanceProvider is only invoked by the CIMOM to return instance data.

The DeepInheritance argument is only used for the EnumerateInstances client operation. It controls whether properties of subclasses are included in returned instances. In JSR48 architecture, such filtering is performed by the CIMOM rather than by the provider.

CIM Operations over HTTP Specification	Java WBEM API (javax.wbem.provider.*)
ClassName	op
InstanceName	op
DeepInheritance	not mapped
LocalOnly	localOnly
IncludeQualifiers	not mapped
IncludeClassOrigin	includeClassOrigin
PropertyList	propertyList
ModifiedInstance or NewInstance	ci
QueryLanguage	ql
Query	query

AssociatorProvider

The AssociationProvider interface is used to support the client association traversal operations on instances. Only providers for association classes need to implement this interface.

CIM Operation	Java WBEM API (javax.wbem.provider.*)
AssociatorNames	associatorNames
Associators	associators
ReferenceNames	referenceNames
References	references

The following table shows the mapping of argument names between the CIM Operations over HTTP specification and the JSR48 client API.

The IncludeQualifiers argument is not mapped because instances do not have qualifiers and an AssociatorProvider is only invoked by the CIMOM to return instance data.

CIM Operations over HTTP Specification	Java WBEM API (javax.wbem.provider.*)
ObjectName	objectName
AssocClass	assocClass
ResultClass	resultClass
Role	role
ResultRole	resultRole
IncludeQualifiers	not mapped
IncludeClassOrigin	includeClassOrigin
PropertyList	propertyList

IndicationProvider

In JSR48 architecture, an indication allows a client to be notified when a particular event occurs. In order to be notified a client must first subscribe for the event of interest. The subscription process requires a client to specify

- which events are to be monitored using filters
- where to send the notification when the event occurs

Filters are query expressions that specify the classes to monitor and the conditions that must be satisfied before notification is sent to a listener. Conditions can be specified using a combination of property names, values, logical operators and arithmetic operators. For further information about query expression syntax, consult the Common Query Language (CQL) specification.

Metadata events (e.g., class creation, qualifier removal, etc.) are handled by the CIMOM. In contrast, data events (e.g., instance modification), are handled by providers.

Following methods must be supported unless provider for a class will not or cannot support indications:

Method	Purpose
`activateFilter`	`informs the provider to start monitoring the events specified by a filter`
`authorizeFilter`	`determines whether the provider supports a particular filter`
`deactivateFilter`	`informs the provider to stop monitoring the events specified by a filter`

MethodProvider

A provider need not implement this interface if no extrinsic methods defined for its class. Otherwise, its provider must implement this interface. If no methods are supported, return CIM_ERR_METHOD_NOT_AVAILABLE.

CIM Operation	Java WBEM API (javax.wbem.provider.*)
Extrinsic Method Invocation	invokeMethod

The CIM Operations over HTTP specification does not explicitly define a syntax for performing an extrinsic method operation. However, the JSR48 API defines the following arguments:

Argument	Purpose
op	address of the instance
methodName	name of the extrinsic method
inArgs	list of input arguments
outArgs	list of output arguments

PullInstanceProvider

The PullInstanceProvider interface is used to support the client data and query operations when data is returned using a Pull data operation. Note that queries from a client about metadata will be handled by the CIMOM. Only client queries about data will be handled by a provider.

CIM Operations	Java WBEM API (javax.wbem.provider.*)
CreateInstance	createInstance
GetInstance	getInstance
ModifyInstance	modifyInstance
DeleteInstance	deleteInstance
EnumerateInstanceNames	enumerateInstanceNames
EnumerateInstances	enumerateInstances
ExecQuery	execQuery

The following table shows the mapping of argument names between the CIM Operations over HTTP specification and the JSR48 provider API.

The IncludeQualifiers argument is not mapped because instances do not have qualifiers and PullInstanceProvider is only invoked by the CIMOM to return instance data.

The DeepInheritance argument is only used for the EnumerateInstances client operation. It controls whether properties of subclasses are included in returned instances. In JSR48 architecture, such filtering is performed by the CIMOM rather than by the provider.

CIM Operations over HTTP Specification	Java WBEM API (javax.wbem.provider.*)
ClassName	op
InstanceName	op
DeepInheritance	not mapped
LocalOnly	localOnly
IncludeQualifiers	not mapped
IncludeClassOrigin	includeClassOrigin
PropertyList	propertyList
ModifiedInstance or NewInstance	ci
FilterQueryLanguage	filterQueryLanguage
FilterQuery	filterQuery
QueryLanguage	ql
Query	query

CIM Operations over HTTP Specification	Java WBEM API (javax.wbem.provider.*)
`ContinueOnError`	`continueOnError`

PullAssociatorProvider

The AssociationProvider interface is used to support the client association traversal operations on instances. Only providers for association classes need to implement this interface.

CIM Operation	Java WBEM API (javax.wbem.provider.*)
AssociatorNames	associatorNames
Associators	associators
ReferenceNames	referenceNames
References	references

The following table shows the mapping of argument names between the CIM Operations over HTTP specification and the JSR48 client API.

The IncludeQualifiers argument is not mapped because instances do not have qualifiers and an AssociatorProvider is only invoked by the CIMOM to return instance data.

CIM Operations over HTTP Specification	Java WBEM API (javax.wbem.provider.*)
ObjectName	objectName
AssocClass	assocClass
ResultClass	resultClass
Role	role
ResultRole	resultRole
IncludeQualifiers	not mapped
IncludeClassOrigin	includeClassOrigin
PropertyList	propertyList
FilterQueryLanguage	filterQueryLanguage
FilterQuery	filterQuery
ContinueOnError	continueOnError

CIM Errors (javax.wbem)

The CIM Operations over HTTP specification defines a set of errors that can be returned for each CIM Operation. The WBEMException class is the Java representation of a CIM error. For detailed information on WBEMException, please consult the JSR48 Javadoc.

Asynchronous Method Invocations

The CIM Operations over HTTP specification defines the behavior and semantics for both intrinsic and extrinsic asynchronous methods. This allows instrumentation of "long running" management operations (e.g., format disk, create database, or backup/restore file system). For detailed information on asynchronous methods, please consult the CIM Operations over HTTP specification.

The CloseableIterator class can be used to provide support for asynchronous methods. For detailed information on CloseableIterator, please consult the JSR48 Javadoc.

Appendix A: Change Summary

The following table summarizes all changes to this document. The newest changes will be listed last.

Version	Date	Description
0.1	01/05/06	Initial Draft based on new API
0.2	06/05/06	Community Review Draft
0.3	07/25/06	Public Review Draft
0.4	12/30/06	Final Review Draft
1	10/26/09	Final

Appendix B: Futures

The following lists items that the JSR48 Expert Group is considering for future work.

- CIM Object Manager API
  - Client Object Manager Adapter Interface
  - Provider Object Manager Adapter Interface
  - Indication Handler Object Manager Adapter Interface
  - Repository Object Manager Adapter Interface
  - Security Object Manager Adapter Interface
- xmlCIM
  - May add an API to get the xmlCIM representation of CIM Qualifier Types, Classes and Instances.
- WS-CIM Support
  - May add an API to get the WS-CIM representation of CIM Qualifier Types, Classes and Instances.
- CLP Support
  - This may be just implementation of a client adapter to the client API, but the EG will verify no updates to the API are needed.
- WS-MANAGMENT Support
  - This may be just implementation of a client adapter to the client API, but the EG will verify no updates to the API are needed.
- WSDM Support
  - This may be just implementation of a client adapter to the client API, but the EG will verify no updates to the API are needed.
- CIM-XMI Mapping Specification

Appendix C: Contributors

WBEM Solutions, Inc.

- Jim Davis
- Paul Ferdinand
- Carl Chan (editor)

IBM.

- Ramandeep Arora
- Dave Blaschke