Pegasus CIM Object Broker Manual

Contents

Introduction

Welcome to PEGASUS

DRAFT FOR REVIEW

Pegasus is an open source implementation of the manageability environment defined by the DMTF WBEM standards.

This Document

This is a working document that is the temporary user and developers manual for the Pegasus CIM Server implementation which is being created by the OpenGroup Enterprise Management Program Group.

This manual serves as both a manual for installation and operation of the prototype version of Pegasus and a manual for developing components to attach to Pegasus. In addition to the manual, we keep current working information in a set of README files within the Pegasus distributions. Please consult these files also.

Within this manual you will find:

- Information on the installation and operation of the broker and additional components.
- A basic defintion of the architecture and function of Pegasus.
- The current definition of the programming interfaces that Pegasus allows. As these interfaces are

stabilized they will be moved from this document to an Open Group specificaton.

- A number of defintions of additional code that is available to interface with Pegasus as providers, consumers, and services.

ATTN: We need to give a description of the sections and what they accomplish with hotlinks to each.

This is not an internals manual for Pegasus. The Internals for Pegasus developers will be defined in a separate manual that will be available from The Open Group (http://www.opengroup.org/management).

NOTE: This a working document today. It is revised frequently as we learn more about the documentation system and stabilize the Pegasus interfaces. We apologize for the existing insonsistencies and errors in this working version of the document.

This version of the Pegasus User Manual was created at 05/05/2001 08:36:43 AM

Alphabetic index *Hierarchy of classes*

Pegasus Documentation

Introduction

Welcome to PEGASUS

DRAFT FOR REVIEW

Pegasus is an open source implementation of the manageability environment defined by the DMTF WBEM standards.

This Document

This is a working document that is the temporary user and developers manual for the Pegasus CIM Server implementation which is being created by the OpenGroup Enterprise Management Program Group.

This manual serves as both a manual for installation and operation of the prototype version of Pegasus and a manual for developing components to attach to Pegasus. In addition to the manual, we keep current working information in a set of README files within the Pegasus distributions. Please consult these files also.

Within this manual you will find:

- Information on the installation and operation of the broker and additional components.
- A basic defintion of the architecture and function of Pegasus.
- The current definition of the programming interfaces that Pegasus allows. As these interfaces are stabilized they will be moved from this document to an Open Group specificaton.
- A number of defintions of additional code that is available to interface with Pegasus as providers, consumers, and services.

ATTN: We need to give a description of the sections and what they accomplish with hotlinks to each.

This is not an internals manual for Pegasus. The Internals for Pegasus developers will be defined in a separate manual that will be available from The Open Group (http://www.opengroup.org/management).

NOTE: This a working document today. It is revised frequently as we learn more about the documentation system and stabilize the Pegasus interfaces. We apologize for the existing insonsistencies and errors in this working version of the document.

This version of the Pegasus User Manual was created at 05/05/2001 08:36:43 AM

Alphabetic index *Hierarchy of classes*

Table of Contents

General stuff

Namespaces

- CIMScope *The CIMQualifier Scopes are as follows: NONE, Class, ASSOCIATION,INDICATION,PROPERTY, REFERENCE, METHOD, PARAMETER, ANY*

Classes

- CIMOperations *The CIMOperations Class.*
- CIMClient *Class CIMClient - This class defines the client interfaces for Pegasus.*
- CIMRepository *This class derives from the CIMOperations class and provides a simple implementation of a CIM repository.*
- Array *Array Class.*
- CIMClass *The CIMClass class is used to represent CIM classes in Pegasus.*
- CIMConstClass *CIMConstClass - ATTN: define this.*
- CIMDateTime *The CIMDateTime class represents the CIM datetime data type as a C++ class CIMDateTime.*
- CIMInstance *Class CIMInstance - The CIMInstance class represents the instance of a CIM class in Pegasus.*
- CIMMethod *Class CIMMethod - This class defines the operations associated with manipulation of the Pegasus implementation of the CIM CIMMethod.*
- CIMName *The name class defines static methods for handling CIM names.*
- CIMProperty *CIMProperty Class - ATTN:*
- CIMQualifier *Class CIMQualifier - This class defines the Pegasus implementation of the CIM CIMQualifier QUALIFIER*
- CIMQualifierDecl *Class CIMQualifierDecl*
- CIMReference *The CIMReference class represents the value of a reference.*
- CIMType *The CIMType Class defines the CIMType enumeration which introduces symbolic constants for the CIM data types.*
- CIMValue *The CIMValue class represents a value of any of the CIM data types (see CIMTypeh for a list of valid CIM data types).*
- Char16 *The Char16 class represents a CIM sixteen bit character (char16).*
- KeyBinding *The KeyBinding class associates a key name, value, and type.*
- AddedReferenceToClass *ATTN:*
- AlreadyExists *ATTN:*
- AssertionFailureException *Class AssertionFailureException This is an Exception class tied to the definiton of an assert named PEGASUS_ASSERT.*
- BadQualifierOverride *ATTN:*

- BadQualifierScope *ATTN:*
- BadReference *ATTN:*
- CIMException *The CIMException defines the CIM exceptions that are formally defined in the CIM Operations over HTTP specification.*
- CannotCreateDirectory *ATTN:*
- CannotOpenFile *ATTN:*
- ClassAlreadyResolved *ATTN:*
- ClassNotResolved *ATTN:*
- Exception *Class Exception*
- ExpectedReferenceValue *ATTN:*
- FailedToRemoveDirectory *ATTN:*
- FailedToRemoveFile *ATTN:*
- IllegalName *ATTN:*
- IllegalTypeTag *ATTN:*
- InstanceAlreadyResolved *ATTN:*
- InstantiatedAbstractClass *ATTN:*
- InvalidMethodOverride *ATTN:*
- InvalidPropertyOverride *ATTN:*
- MissingReferenceClassName *ATTN:*
- NoSuchDirectory *ATTN:*
- NoSuchFile *ATTN:*
- NoSuchNameSpace *ATTN:*
- NoSuchProperty *ATTN:*
- NotImplemented *ATTN:*
- NullPointer *ATTN:*
- NullType *ATTN:*
- OutOfBounds *ATTN:*
- TruncatedCharacter *ATTN:*
- TypeMismatch *ATTN:*
- UndeclaredQualifier *ATTN:*
- UnitializedHandle *ATTN:*
- Stopwatch *Stopwatch - A class for measuring elapsed time Stopwatch is a class for measuring time intervals within the environment.*
- String *The Pegasus String C++ Class implements the CIM string type.*
- TimeValue *The TimeValue class represents time expressed in seconds plus microseconds*

Functions

- Pegasus Interfaces::CIM Operations over HTTP::CIMOperations::associatorNames *The* `associatorNames` *operation is used to enumerate the names of CIM Objects (Classes or Instances) that are associated to a particular source CIM Object.*

- Pegasus Interfaces::CIM Operations over HTTP::CIMOperations::associators *The `Associators` method enumerates CIM Objects (Classes or Instances) that are associated to a particular source CIM Object.*
- Pegasus Interfaces::CIM Operations over HTTP::CIMOperations::createClass *The `createClass` method creates a single CIM Class in the target Namespace.*
- Pegasus Interfaces::CIM Operations over HTTP::CIMOperations::createInstance *The `createInstance` method creates a single CIM Instance in the target Namespace.*
- Pegasus Interfaces::CIM Operations over HTTP::CIMOperations::deleteClass *The `DeleteClass` method deletes a single CIM Class from the target Namespace.*
- Pegasus Interfaces::CIM Operations over HTTP::CIMOperations::deleteInstance *The `DeleteInstance` operation deletes a single CIM Instance from the target Namespace.*
- Pegasus Interfaces::CIM Operations over HTTP::CIMOperations::deleteQualifier *The `deleteQualifier` operation deletes a single CIMQualifier declaration from the target Namespace.*
- Pegasus Interfaces::CIM Operations over HTTP::CIMOperations::enumerateClassNames *The `enumerateClassNames` operation is used to enumerate the names of subclasses of a CIM Class in the target Namespace.*
- Pegasus Interfaces::CIM Operations over HTTP::CIMOperations::enumerateClasses *The `enumerateClasses` method is used to enumerate subclasses of a CIM Class in the target Namespace.*
- Pegasus Interfaces::CIM Operations over HTTP::CIMOperations::enumerateInstanceNames *The `enumerateInstanceNames` operation enumerates the names (model paths) of the instances of a CIM Class in the target Namespace.*
- Pegasus Interfaces::CIM Operations over HTTP::CIMOperations::enumerateInstances *The `enumerateInstances` method enumerates instances of a CIM Class in the target Namespace.*
- Pegasus Interfaces::CIM Operations over HTTP::CIMOperations::enumerateQualifiers *The `enumerateQualifiers` operation is used to enumerate CIMQualifier declarations from the target Namespace.*
- Pegasus Interfaces::CIM Operations over HTTP::CIMOperations::execQuery *The `execQuery` is used to execute a query against the target Namespace.*
- Pegasus Interfaces::CIM Operations over HTTP::CIMOperations::getClass *The `GetClass` method returns a single CIM Class from the target Namespace where the ClassName input parameter defines the name of the class to be retrieved.*
- Pegasus Interfaces::CIM Operations over HTTP::CIMOperations::getInstance *The <`GetInstance` method returns a single CIM Instance from the target Namespace based on the InstanceName parameter provided.*
- Pegasus Interfaces::CIM Operations over HTTP::CIMOperations::getProperty *This operation is used to retrieve a single property value from a CIM Instance in the target Namespace.*
- Pegasus Interfaces::CIM Operations over HTTP::CIMOperations::getQualifier *The `getQualifier` operation retrieves a single CIMQualifier declaration from the target Namespace.*
- Pegasus Interfaces::CIM Operations over HTTP::CIMOperations::invokeMethod *Any CIM Server is assumed to support extrinsic methods.*
- Pegasus Interfaces::CIM Operations over HTTP::CIMOperations::modifyClass *The `modifyClass` method modifies an existing CIM Class in the target Namespace.*
- Pegasus Interfaces::CIM Operations over HTTP::CIMOperations::modifyInstance *The `modifyInstance` method is used to modify an existing CIM Instance in the target Namespace.*
- Pegasus Interfaces::CIM Operations over HTTP::CIMOperations::referenceNames *The `referenceNames` operation is used to enumerate the association objects that refer to a particular target CIM Object (Class or Instance).*
- Pegasus Interfaces::CIM Operations over HTTP::CIMOperations::references *The `references` operation enumerates the association objects that refer to a particular target CIM Object (Class or Instance).*

- Pegasus Interfaces::CIM Operations over HTTP::CIMOperations::setProperty *The* `setProperty` *operation sets a single property value in a CIM Instance in the target Namespace.*
- Pegasus Interfaces::CIM Operations over HTTP::CIMOperations::setQualifier *The* `setQualifier` *creates or update a single CIMQualifier declaration in the target Namespace.*
- Pegasus Interfaces::Pegasus Client Interfaces::CIMClient::CIMClient
- Pegasus Interfaces::Pegasus Client Interfaces::CIMClient::associatorNames
- Pegasus Interfaces::Pegasus Client Interfaces::CIMClient::associators
- Pegasus Interfaces::Pegasus Client Interfaces::CIMClient::connect
- Pegasus Interfaces::Pegasus Client Interfaces::CIMClient::createClass
- Pegasus Interfaces::Pegasus Client Interfaces::CIMClient::createInstance
- Pegasus Interfaces::Pegasus Client Interfaces::CIMClient::deleteClass
- Pegasus Interfaces::Pegasus Client Interfaces::CIMClient::deleteInstance
- Pegasus Interfaces::Pegasus Client Interfaces::CIMClient::deleteQualifier
- Pegasus Interfaces::Pegasus Client Interfaces::CIMClient::enumerateClassNames
- Pegasus Interfaces::Pegasus Client Interfaces::CIMClient::enumerateClasses
- Pegasus Interfaces::Pegasus Client Interfaces::CIMClient::enumerateInstanceNames
- Pegasus Interfaces::Pegasus Client Interfaces::CIMClient::enumerateInstances
- Pegasus Interfaces::Pegasus Client Interfaces::CIMClient::enumerateQualifiers
- Pegasus Interfaces::Pegasus Client Interfaces::CIMClient::execQuery
- Pegasus Interfaces::Pegasus Client Interfaces::CIMClient::get
- Pegasus Interfaces::Pegasus Client Interfaces::CIMClient::getClass
- Pegasus Interfaces::Pegasus Client Interfaces::CIMClient::getInstance
- Pegasus Interfaces::Pegasus Client Interfaces::CIMClient::getProperty
- Pegasus Interfaces::Pegasus Client Interfaces::CIMClient::getQualifier
- Pegasus Interfaces::Pegasus Client Interfaces::CIMClient::getTimeOut
- Pegasus Interfaces::Pegasus Client Interfaces::CIMClient::invokeMethod
- Pegasus Interfaces::Pegasus Client Interfaces::CIMClient::modifyClass
- Pegasus Interfaces::Pegasus Client Interfaces::CIMClient::modifyInstance
- Pegasus Interfaces::Pegasus Client Interfaces::CIMClient::referenceNames
- Pegasus Interfaces::Pegasus Client Interfaces::CIMClient::references
- Pegasus Interfaces::Pegasus Client Interfaces::CIMClient::runForever
- Pegasus Interfaces::Pegasus Client Interfaces::CIMClient::runOnce
- Pegasus Interfaces::Pegasus Client Interfaces::CIMClient::setQualifier
- Pegasus Interfaces::Pegasus Client Interfaces::CIMClient::setTimeOut
- Pegasus Interfaces::Pegasus Client Interfaces::CIMClient::~CIMClient
- Pegasus Interfaces::Repository Interfaces::CIMRepository::CIMRepository *Constructor*
- Pegasus Interfaces::Repository Interfaces::CIMRepository::associatorNames *associateNames*
- Pegasus Interfaces::Repository Interfaces::CIMRepository::associators *associators*
- Pegasus Interfaces::Repository Interfaces::CIMRepository::createClass *createClass*
- Pegasus Interfaces::Repository Interfaces::CIMRepository::createInstance *createInstance*

- Pegasus Interfaces::Repository Interfaces::CIMRepository::createNameSpace *CIMMethod createNameSpace - Creates a new namespace in the repository*
- Pegasus Interfaces::Repository Interfaces::CIMRepository::deleteClass *deleteClass*
- Pegasus Interfaces::Repository Interfaces::CIMRepository::deleteInstance *deleteInstance*
- Pegasus Interfaces::Repository Interfaces::CIMRepository::deleteNameSpace *CIMMethod deleteNameSpace - Deletes a namespace in the repository.*
- Pegasus Interfaces::Repository Interfaces::CIMRepository::deleteQualifier *virtual deleteQualifier*
- Pegasus Interfaces::Repository Interfaces::CIMRepository::enumerateClassNames *enumerateClassNames*
- Pegasus Interfaces::Repository Interfaces::CIMRepository::enumerateClasses *enumerateClasses*
- Pegasus Interfaces::Repository Interfaces::CIMRepository::enumerateInstanceNames *enumerateInstanceNames*
- Pegasus Interfaces::Repository Interfaces::CIMRepository::enumerateInstances *enumerateInstances*
- Pegasus Interfaces::Repository Interfaces::CIMRepository::enumerateNameSpaces *CIMMethod enumerateNameSpaces - Get all of the namespaces in the repository.*
- Pegasus Interfaces::Repository Interfaces::CIMRepository::enumerateQualifiers *enumerateQualifiers*
- Pegasus Interfaces::Repository Interfaces::CIMRepository::execQuery *execQuery*
- Pegasus Interfaces::Repository Interfaces::CIMRepository::getClass *virtual class CIMClass.*
- Pegasus Interfaces::Repository Interfaces::CIMRepository::getInstance *getInstance*
- Pegasus Interfaces::Repository Interfaces::CIMRepository::getProperty *getProperty*
- Pegasus Interfaces::Repository Interfaces::CIMRepository::getQualifier *getQualifier*
- Pegasus Interfaces::Repository Interfaces::CIMRepository::invokeMethod *invokeMethod*
- Pegasus Interfaces::Repository Interfaces::CIMRepository::modifyClass *modifyClass*
- Pegasus Interfaces::Repository Interfaces::CIMRepository::modifyInstance *modifyInstance*
- Pegasus Interfaces::Repository Interfaces::CIMRepository::referenceNames *referenceNames*
- Pegasus Interfaces::Repository Interfaces::CIMRepository::references *references*
- Pegasus Interfaces::Repository Interfaces::CIMRepository::setProperty *setProperty*
- Pegasus Interfaces::Repository Interfaces::CIMRepository::setQualifier *setQualifier*
- Pegasus Interfaces::Repository Interfaces::CIMRepository::~CIMRepository *Descructor*
- **Array:**
  - Programming Pegasus::Class Definitions::Array::Array *Constructs an array with size elements.*
  - Programming Pegasus::Class Definitions::Array::Array *Default constructor*
  - Programming Pegasus::Class Definitions::Array::Array *Copy Constructor*
  - Programming Pegasus::Class Definitions::Array::Array *Constructs an array with size elements.*
  - Programming Pegasus::Class Definitions::Array::Array *Constructs an array with size elements.*
- **append:**
  - Programming Pegasus::Class Definitions::Array::append *Appends size elements at x to the end of this array*
  - Programming Pegasus::Class Definitions::Array::append *Appends an element to the end of the array.*
- Programming Pegasus::Class Definitions::Array::appendArray *Appends one array to another.*
- Programming Pegasus::Class Definitions::Array::clear *Clears the contents of the array.*
- Programming Pegasus::Class Definitions::Array::getCapacity *Returns the capacity of the array*
- Programming Pegasus::Class Definitions::Array::getData *Returns a pointer to the first element of the array*

- Programming Pegasus::Class Definitions::Array::grow *Make the size of the array grow by size elements.*
- **insert:**
  - Programming Pegasus::Class Definitions::Array::insert *Inserts the element at the given index in the array.*
  - Programming Pegasus::Class Definitions::Array::insert *Inserts size elements at x into the array at the given position.*
- Programming Pegasus::Class Definitions::Array::operator= *Assignment operator*
- **operator[]:**
  - Programming Pegasus::Class Definitions::Array::operator[] *Returns the element at the index given by the pos argument.*
  - Programming Pegasus::Class Definitions::Array::operator[] *Same as the above method except that this is the version called on const arrays.*
- **prepend:**
  - Programming Pegasus::Class Definitions::Array::prepend *Appends one element to the beginning of the array.*
  - Programming Pegasus::Class Definitions::Array::prepend *Appends size elements to the array starting at the memory address given by x.*
- **remove:**
  - Programming Pegasus::Class Definitions::Array::remove *Removes the element at the given position from the array.*
  - Programming Pegasus::Class Definitions::Array::remove *Removes size elements starting at the given position.*
- Programming Pegasus::Class Definitions::Array::reserve *Reserves memory for capacity elements.*
- Programming Pegasus::Class Definitions::Array::size *Returns the number of elements in the array.*
- Programming Pegasus::Class Definitions::Array::swap *Swaps the contents of two arrays*
- Programming Pegasus::Class Definitions::Array::~Array *Destructs the objects, freeing any resources*
- **CIMClass:**
  - Programming Pegasus::Class Definitions::CIMClass::CIMClass *Constructor - Creates a Class from inputs of a classname and SuperClassName*
  - Programming Pegasus::Class Definitions::CIMClass::CIMClass *Constructor - Creates an uninitiated a new CIM object reprenting a CIM class.*
  - Programming Pegasus::Class Definitions::CIMClass::CIMClass *Constructor - Creates a class from a previous class*
- Programming Pegasus::Class Definitions::CIMClass::addMethod *CIMMethod addMethod - Adds the method object defined by the input parameter to the class and increments the count of the number of methods in the class*
- Programming Pegasus::Class Definitions::CIMClass::addProperty *CIMMethod addProperty - Adds the specified property object to the properties in the CIM class*
- Programming Pegasus::Class Definitions::CIMClass::addQualifier *CIMMethod addQualifier - Adds the specified qualifier to the class and increments the qualifier count.*
- Programming Pegasus::Class Definitions::CIMClass::clone *CIMMethod clone - ATTN:*
- Programming Pegasus::Class Definitions::CIMClass::findMethod *CIMMethod findMethod - Located the method object defined by the name input*
- Programming Pegasus::Class Definitions::CIMClass::findProperty *CIMMethod findProperty - Finds the property object with the name defined by the input parameter in the class.*
- Programming Pegasus::Class Definitions::CIMClass::findQualifier *CIMMethod findQualifier - Finds a qualifier with the specified input name if it exists in the class @param name CIMName of the qualifier to be found @return Position of the qualifier in the Class ATTN: Clarify the return.*

- Programming Pegasus::Class Definitions::CIMClass::getClassName *CIMMethod Gets the name of the class ATTN: COMMENT.*
- **getMethod:**
  - Programming Pegasus::Class Definitions::CIMClass::getMethod *CIMMethod getMethod - Gets the method object defined by the input parameter.*
  - Programming Pegasus::Class Definitions::CIMClass::getMethod *CIMMethod getMethod - ATTN:*
- Programming Pegasus::Class Definitions::CIMClass::getMethodCount *CIMMethod getMethodCount - Count of the number of methods in the class*
- **getProperty:**
  - Programming Pegasus::Class Definitions::CIMClass::getProperty *CIMMethod getProperty - Returns a property representing the property defined by the input parameter*
  - Programming Pegasus::Class Definitions::CIMClass::getProperty *CIMMethod getProperty - ATTN*
- Programming Pegasus::Class Definitions::CIMClass::getPropertyCount *CIMMethod getProperty - Gets the count of the number of properties defined in the class.*
- **getQualifier:**
  - Programming Pegasus::Class Definitions::CIMClass::getQualifier *CIMMethod getQualifier - Gets the CIMQualifier object defined by the input parameter*
  - Programming Pegasus::Class Definitions::CIMClass::getQualifier *CIMMethod getQualifier - ATTN:*
- Programming Pegasus::Class Definitions::CIMClass::getQualifierCount *CIMMethod getQualifierCount - Returns the number of qualifiers in the class.*
- Programming Pegasus::Class Definitions::CIMClass::getSuperClassName *CIMMethod getSuperClassName - Gets the name of the Parent*
- Programming Pegasus::Class Definitions::CIMClass::identical *CIMMethod identical - Compares with another class ATTN: Clarify exactly what identical means @parm Class object for the class to be compared*
- Programming Pegasus::Class Definitions::CIMClass::isAbstract *CIMMethod isAbstract*
- Programming Pegasus::Class Definitions::CIMClass::isAssociation *CIMMethod isAssociation - Identifies whether or not this CIM class is an association.*
- Programming Pegasus::Class Definitions::CIMClass::operator int *operator - ATTN:*
- Programming Pegasus::Class Definitions::CIMClass::operator= *Operator = Assigns the CIM Class constructor*
- Programming Pegasus::Class Definitions::CIMClass::print *CIMMethod print*
- Programming Pegasus::Class Definitions::CIMClass::removeProperty *CIMMethod removeProperty - Removes the property represented by the position input parameter from the class*
- Programming Pegasus::Class Definitions::CIMClass::resolve *CIMMethod Resolve - Resolve the class: inherit any properties and qualifiers.*
- Programming Pegasus::Class Definitions::CIMClass::setSuperClassName *CIMMethod setSuperClassName - Sets the name of the parent class from the input parameter.*
- Programming Pegasus::Class Definitions::CIMClass::toXml *CIMMethod toXML*
- Programming Pegasus::Class Definitions::CIMClass::~CIMClass *Destructor*
- **CIMDateTime:**
  - Programming Pegasus::Class Definitions::CIMDateTime::CIMDateTime *CIMDateTime CIMMethod - Creates the CIMDateTime instance from another CIMDateTime instance*
  - Programming Pegasus::Class Definitions::CIMDateTime::CIMDateTime *CIMDateTime CIMMethod*
  - Programming Pegasus::Class Definitions::CIMDateTime::CIMDateTime *CIMDateTime CIMMethod creates the CIM CIMDateTime from a string constant*

- Programming Pegasus::Class Definitions::CIMDateTime::clear *CIMDateTime method clear - Clears the datetime class instance*
- Programming Pegasus::Class Definitions::CIMDateTime::getString *method getString*
- Programming Pegasus::Class Definitions::CIMDateTime::isNull *CIMDateTime isNull method - Tests for an all zero date time*

```
CIMDateTime dt;         dtclear();      assert(dtisNull());
```

- Programming Pegasus::Class Definitions::CIMDateTime::operator= *CIMDateTime method again*
- Programming Pegasus::Class Definitions::CIMDateTime::set *method set - Sets the date time.*
- **CIMInstance:**
  - Programming Pegasus::Class Definitions::CIMInstance::CIMInstance *Constructor - Creates an Instance object with the classname from the input parameters*
  - Programming Pegasus::Class Definitions::CIMInstance::CIMInstance *Constructor - Create a CIM Instance object.*
  - Programming Pegasus::Class Definitions::CIMInstance::CIMInstance *Constructor - Create a CIMInstance object from another Instance.*
- Programming Pegasus::Class Definitions::CIMInstance::addProperty *addProperty - Adds a property object defined by the input parameter to the CIMInstance*
- Programming Pegasus::Class Definitions::CIMInstance::addQualifier *addQualifier - Adds the CIMQualifier object to the instance.*
- Programming Pegasus::Class Definitions::CIMInstance::clone *CIMMethod*
- Programming Pegasus::Class Definitions::CIMInstance::findProperty *findProperty - Searches the CIMProperty objects installed in the CIMInstance for property objects with the name defined by the input.*
- Programming Pegasus::Class Definitions::CIMInstance::findQualifier *findQualifier - Searches the instance for the qualifier object defined by the inputparameter.*
- Programming Pegasus::Class Definitions::CIMInstance::getClassName *getClassName - Returns the class name of the instance*
- Programming Pegasus::Class Definitions::CIMInstance::getInstanceName *getInstnaceName - Get the instance name of this instance.*
- **getProperty:**
  - Programming Pegasus::Class Definitions::CIMInstance::getProperty *getProperty - Gets the CIMproperty object in the CIMInstance defined by the input index parameter.*
  - Programming Pegasus::Class Definitions::CIMInstance::getProperty *getProperty - Gets the CIMproperty object in the CIMInstance defined by the input index parameter.*
- Programming Pegasus::Class Definitions::CIMInstance::getPropertyCount *getPropertyCount - Gets the numbercount of CIMProperty objects defined for this CIMInstance.*
- **getQualifier:**
  - Programming Pegasus::Class Definitions::CIMInstance::getQualifier *getQualifier - Retrieves the qualifier object defined by the index input parameter.*
  - Programming Pegasus::Class Definitions::CIMInstance::getQualifier *getQualifier - Retrieves the qualifier object defined by the index input parameter.*
- Programming Pegasus::Class Definitions::CIMInstance::getQualifierCount *getQualifierCount - Gets the numbercount of CIMQualifierobjects defined for this CIMInstance.*
- Programming Pegasus::Class Definitions::CIMInstance::identical *identical - Compares the CIMInstance with another CIMInstance defined by the input parameter for equality of all components.*

- Programming Pegasus::Class Definitions::CIMInstance::operator int *operator int() - ATTN:*
- Programming Pegasus::Class Definitions::CIMInstance::operator= *Constructor - ATTN*
- Programming Pegasus::Class Definitions::CIMInstance::print *CIMMethod*
- Programming Pegasus::Class Definitions::CIMInstance::resolve *resolve - ATTN:*
- Programming Pegasus::Class Definitions::CIMInstance::toXml *toXml - Creates an XML transformation of the CIMInstance compatiblewith the DMTF CIM Operations over HTTP defintions.*
- Programming Pegasus::Class Definitions::CIMInstance::~CIMInstance *Destructor*
- **CIMMethod:**
  - Programming Pegasus::Class Definitions::CIMMethod::CIMMethod *Creates and instantiates a CIM method from another method instance*
  - Programming Pegasus::Class Definitions::CIMMethod::CIMMethod *Creates a CIM method with the specified name, type, and classOrigin*
  - Programming Pegasus::Class Definitions::CIMMethod::CIMMethod *Creates and instantiates a CIM method.*
- Programming Pegasus::Class Definitions::CIMMethod::addParameter *CIMMethod addParameter - Adds the parameter defined by the input to the CIMMethod*
- Programming Pegasus::Class Definitions::CIMMethod::addQualifier *CIMMethod addQualifier - @parm CIMQualifier to add*
- Programming Pegasus::Class Definitions::CIMMethod::clone *CIMMethod clone - makes a distinct replica of this method*
- Programming Pegasus::Class Definitions::CIMMethod::findParameter *CIMMethod findParameter - Finds the parameter whose name is given by the name parameter.*
- Programming Pegasus::Class Definitions::CIMMethod::findQualifier *CIMMethod findQualifier - returns the position of the qualifier with the given name.*
- Programming Pegasus::Class Definitions::CIMMethod::getClassOrigin *CIMMethod getClassOrigin - Returns the class in which this method was defined.*
- Programming Pegasus::Class Definitions::CIMMethod::getName *CIMMethod getName - Gets the name of the method*
- **getParameter:**
  - Programming Pegasus::Class Definitions::CIMMethod::getParameter *CIMMethod getParameter - Gets the parameter defined by the index input as a parameter.*
  - Programming Pegasus::Class Definitions::CIMMethod::getParameter *CIMMethod getParameter - ATTN:*
- Programming Pegasus::Class Definitions::CIMMethod::getParameterCount *CIMMethod getParameterCount - Gets the count of the numbeer of Parameters attached to the CIMMethod.*
- Programming Pegasus::Class Definitions::CIMMethod::getPropagated *method getPropagated - ATTN:*
- Programming Pegasus::Class Definitions::CIMMethod::getQualifier *CIMMethod getQualifier - Gets the CIMQualifier defined by the index input as a parameter.*
- Programming Pegasus::Class Definitions::CIMMethod::getQualifierCount *CIMMethod getQualifierCount - Returns the number of Qualifiers attached to this method.*
- Programming Pegasus::Class Definitions::CIMMethod::getType *CIMMethod getType - gets the method type*
- Programming Pegasus::Class Definitions::CIMMethod::identical *CIMMethod identical - Returns true if this method is identical to the one given by the argument x*
- Programming Pegasus::Class Definitions::CIMMethod::operator int *Returns zero if CIMMethod refers to a null pointer*
- Programming Pegasus::Class Definitions::CIMMethod::operator= *Assignment operator*

- Programming Pegasus::Class Definitions::CIMMethod::print *method print - prints this method (in CIM encoded form).*
- **resolve:**
  - Programming Pegasus::Class Definitions::CIMMethod::resolve *CIMMethod resolve*
  - Programming Pegasus::Class Definitions::CIMMethod::resolve *method resolve - ATTN:*
- Programming Pegasus::Class Definitions::CIMMethod::setClassOrigin *CIMMethod setClassOrigin - ATTN:*
- Programming Pegasus::Class Definitions::CIMMethod::setName *CIMMethod setName - Set the method name*
- Programming Pegasus::Class Definitions::CIMMethod::setPropagated *method setPropagated - ATTN:*
- Programming Pegasus::Class Definitions::CIMMethod::setType *CIMMethod setType - Sets the method type to the specified CIM method type as defined in CIMType /Ref{TYPE}*
- Programming Pegasus::Class Definitions::CIMMethod::toXml *method toXML - placing XML encoding of this object into out arguemnt.*
- Programming Pegasus::Class Definitions::CIMMethod::~CIMMethod *Desctructor.*
- Programming Pegasus::Class Definitions::CIMName::equal *CIMMethod equal - Compares two names.*
- **legal:**
  - Programming Pegasus::Class Definitions::CIMName::legal *CIMMethod legal - Determine if the name string input is legal as defnined in the CIMName class definition ATTN: Define what is legal*
  - Programming Pegasus::Class Definitions::CIMName::legal *CIMMethod legal - Determine if the name string input is legal as defnined in the CIMName class definition*
- **CIMProperty:**
  - Programming Pegasus::Class Definitions::CIMProperty::CIMProperty *CIMMethod CIMProperty*
  - Programming Pegasus::Class Definitions::CIMProperty::CIMProperty *CIMMethod CIMProperty*
  - Programming Pegasus::Class Definitions::CIMProperty::CIMProperty *CIMMethod CIMProperty*
- Programming Pegasus::Class Definitions::CIMProperty::addQualifier *CIMMethod addQualifier - ATTN Throws AlreadyExists*
- Programming Pegasus::Class Definitions::CIMProperty::clone *CIMMethod clone - ATTN*
- Programming Pegasus::Class Definitions::CIMProperty::findQualifier *CIMMethod findQualifier - ATTN*
- Programming Pegasus::Class Definitions::CIMProperty::getArraySize *CIMMethod getArraySize - ATTN:*
- Programming Pegasus::Class Definitions::CIMProperty::getClassOrigin *CIMMethod getClassOrigin - ATTN*
- Programming Pegasus::Class Definitions::CIMProperty::getName *CIMMethod*
- Programming Pegasus::Class Definitions::CIMProperty::getPropagated *CIMMethod getPropagated - ATTN*
- **getQualifier:**
  - Programming Pegasus::Class Definitions::CIMProperty::getQualifier *CIMMethod getQualifier - ATTN*
  - Programming Pegasus::Class Definitions::CIMProperty::getQualifier *CIMMethod getQualifier - ATTN*
- Programming Pegasus::Class Definitions::CIMProperty::getQualifierCount *CIMMethod getQualifier - ATTN*
- Programming Pegasus::Class Definitions::CIMProperty::getReferenceClassName *CIMMethod getReferenceClassName - ATTN:*
- Programming Pegasus::Class Definitions::CIMProperty::getValue *CIMMethod getValue - ATTN:*
- Programming Pegasus::Class Definitions::CIMProperty::identical *CIMMethod identical - ATTN*
- Programming Pegasus::Class Definitions::CIMProperty::operator int *ATTN*
- Programming Pegasus::Class Definitions::CIMProperty::operator= *CIMMethod*

- Programming Pegasus::Class Definitions::CIMProperty::print *mthod print -ATTN*
- **resolve:**
  - Programming Pegasus::Class Definitions::CIMProperty::resolve *CIMMethod resolve*
  - Programming Pegasus::Class Definitions::CIMProperty::resolve *CIMMethod resolve - ATTN*
- Programming Pegasus::Class Definitions::CIMProperty::setClassOrigin *CIMMethod setClassOrigin*
- Programming Pegasus::Class Definitions::CIMProperty::setName *CIMMethod setName - Set the property name.*
- Programming Pegasus::Class Definitions::CIMProperty::setPropagated *CIMMethod setProgagated - ATTN*
- Programming Pegasus::Class Definitions::CIMProperty::setValue *CIMMethod setValue - ATTN*
- Programming Pegasus::Class Definitions::CIMProperty::toXml *mthod toXML*
- **CIMQualifier:**
  - Programming Pegasus::Class Definitions::CIMQualifier::CIMQualifier *Constructor instantiates a CIM qualifier with empty name value fieldsConstructor*
  - Programming Pegasus::Class Definitions::CIMQualifier::CIMQualifier *Constructor - instantiates a CIM qualifier object from another qualifier object.*
  - Programming Pegasus::Class Definitions::CIMQualifier::CIMQualifier *Constructor - Instantiates a CIM qualifier object with the parameters defined on input.*
- Programming Pegasus::Class Definitions::CIMQualifier::clone *CIMMethod*
- Programming Pegasus::Class Definitions::CIMQualifier::getFlavor *CIMMethod*
- Programming Pegasus::Class Definitions::CIMQualifier::getName *CIMMethod*
- Programming Pegasus::Class Definitions::CIMQualifier::getPropagated *CIMMethod*
- Programming Pegasus::Class Definitions::CIMQualifier::getType *CIMMethod*
- Programming Pegasus::Class Definitions::CIMQualifier::getValue *CIMMethod*
- Programming Pegasus::Class Definitions::CIMQualifier::identical *CIMMethod*
- Programming Pegasus::Class Definitions::CIMQualifier::isArray *CIMMethod*
- Programming Pegasus::Class Definitions::CIMQualifier::operator int *CIMMethod*
- Programming Pegasus::Class Definitions::CIMQualifier::operator= *operator*
- Programming Pegasus::Class Definitions::CIMQualifier::print *CIMMethod*
- Programming Pegasus::Class Definitions::CIMQualifier::setName *CIMMethod Throws IllegalName if name argument not legal CIM identifier*
- Programming Pegasus::Class Definitions::CIMQualifier::setPropagated *CIMMethod*
- Programming Pegasus::Class Definitions::CIMQualifier::setValue *CIMMethod*
- Programming Pegasus::Class Definitions::CIMQualifier::toXml *CIMMethod*
- Programming Pegasus::Class Definitions::CIMQualifier::~CIMQualifier *destructor*
- **CIMQualifierDecl:**
  - Programming Pegasus::Class Definitions::CIMQualifierDecl::CIMQualifierDecl *Constructor - ATTN:*
  - Programming Pegasus::Class Definitions::CIMQualifierDecl::CIMQualifierDecl *Constructor - ATTN:*
  - Programming Pegasus::Class Definitions::CIMQualifierDecl::CIMQualifierDecl *Constructor Throws IllegalName if name argument not legal CIM identifier.*
- Programming Pegasus::Class Definitions::CIMQualifierDecl::clone *CIMMethod*
- Programming Pegasus::Class Definitions::CIMQualifierDecl::getArraySize *CIMMethod*
- Programming Pegasus::Class Definitions::CIMQualifierDecl::getFlavor *CIMMethod*

- Programming Pegasus::Class Definitions::CIMQualifierDecl::getName *CIMMethod ATTN:*
- Programming Pegasus::Class Definitions::CIMQualifierDecl::getScope *CIMMethod*
- Programming Pegasus::Class Definitions::CIMQualifierDecl::getType *CIMMethod ATTN:*
- Programming Pegasus::Class Definitions::CIMQualifierDecl::getValue *CIMMethod*
- Programming Pegasus::Class Definitions::CIMQualifierDecl::identical *CIMMethod*
- Programming Pegasus::Class Definitions::CIMQualifierDecl::isArray *CIMMethod ATTN:*
- Programming Pegasus::Class Definitions::CIMQualifierDecl::operator int *CIMMethod*
- Programming Pegasus::Class Definitions::CIMQualifierDecl::operator= *Operator*
- Programming Pegasus::Class Definitions::CIMQualifierDecl::print *CIMMethod*
- Programming Pegasus::Class Definitions::CIMQualifierDecl::setName *CIMMethod ATTN:*
- Programming Pegasus::Class Definitions::CIMQualifierDecl::setValue *CIMMethod*
- Programming Pegasus::Class Definitions::CIMQualifierDecl::toXml *CIMMethod*
- Programming Pegasus::Class Definitions::CIMQualifierDecl::~CIMQualifierDecl *Destructor*
- **CIMReference:**
  - Programming Pegasus::Class Definitions::CIMReference::CIMReference *Default constructor.*
  - Programming Pegasus::Class Definitions::CIMReference::CIMReference *Copy constructor.*
  - Programming Pegasus::Class Definitions::CIMReference::CIMReference *Initializes this object from a CIM object name.*
  - Programming Pegasus::Class Definitions::CIMReference::CIMReference *Initializes this object from a CIM object name (char* version).*
  - Programming Pegasus::Class Definitions::CIMReference::CIMReference *Constructs a CIMReference from constituent elements.*
- Programming Pegasus::Class Definitions::CIMReference::clear *Clears out the internal fields of this object making it an empty (or unitialized reference).*
- Programming Pegasus::Class Definitions::CIMReference::getClassName *Accessor.*
- Programming Pegasus::Class Definitions::CIMReference::getHost *Accessor.*
- Programming Pegasus::Class Definitions::CIMReference::getKeyBindings *Accessor.*
- Programming Pegasus::Class Definitions::CIMReference::getNameSpace *Accessor*
- Programming Pegasus::Class Definitions::CIMReference::identical *Returns true if this reference is identical to the one given by the x argument*
- Programming Pegasus::Class Definitions::CIMReference::makeHashCode *Generates hash code for the given reference.*
- **operator=:**
  - Programming Pegasus::Class Definitions::CIMReference::operator= *Assignment operator*
  - Programming Pegasus::Class Definitions::CIMReference::operator= *Same as set() above except that it is an assignment operator*
  - Programming Pegasus::Class Definitions::CIMReference::operator= *Same as set() above except that it is an assignment operator*
- Programming Pegasus::Class Definitions::CIMReference::print *Prints the XML encoding of this objet*
- **set:**
  - Programming Pegasus::Class Definitions::CIMReference::set *Sets this reference from constituent elements.*
  - Programming Pegasus::Class Definitions::CIMReference::set *Set the reference from an object name .*

- Programming Pegasus::Class Definitions::CIMReference::setClassName *Sets the class name component to the following string.*
- Programming Pegasus::Class Definitions::CIMReference::setHost *Modifier.*
- Programming Pegasus::Class Definitions::CIMReference::setKeyBindings *Modifier.*
- Programming Pegasus::Class Definitions::CIMReference::setNameSpace *Sets the namespace component.*
- Programming Pegasus::Class Definitions::CIMReference::toString *Returns the object name represented by this reference.*
- Programming Pegasus::Class Definitions::CIMReference::toXml *Encodes this object as XML.*
- Programming Pegasus::Class Definitions::CIMReference::~CIMReference *Destructor*
- **CIMValue:**
  - Programming Pegasus::Class Definitions::CIMValue::CIMValue *Constructor*
  - Programming Pegasus::Class Definitions::CIMValue::CIMValue *Constructor*
  - Programming Pegasus::Class Definitions::CIMValue::CIMValue *Constructor*
  - Programming Pegasus::Class Definitions::CIMValue::CIMValue *Constructor*
  - Programming Pegasus::Class Definitions::CIMValue::CIMValue *Constructor*
  - Programming Pegasus::Class Definitions::CIMValue::CIMValue *Constructor*
  - Programming Pegasus::Class Definitions::CIMValue::CIMValue *Constructor*
  - Programming Pegasus::Class Definitions::CIMValue::CIMValue *Constructor*
  - Programming Pegasus::Class Definitions::CIMValue::CIMValue *Constructor*
  - Programming Pegasus::Class Definitions::CIMValue::CIMValue *Constructor*
  - Programming Pegasus::Class Definitions::CIMValue::CIMValue *Constructor*
  - Programming Pegasus::Class Definitions::CIMValue::CIMValue *Constructor*
  - Programming Pegasus::Class Definitions::CIMValue::CIMValue *Constructor*
  - Programming Pegasus::Class Definitions::CIMValue::CIMValue *Constructor*
  - Programming Pegasus::Class Definitions::CIMValue::CIMValue *Constructor*
  - Programming Pegasus::Class Definitions::CIMValue::CIMValue *Constructor*
  - Programming Pegasus::Class Definitions::CIMValue::CIMValue *Constructor*
  - Programming Pegasus::Class Definitions::CIMValue::CIMValue *Constructor*
  - Programming Pegasus::Class Definitions::CIMValue::CIMValue *Constructor*
  - Programming Pegasus::Class Definitions::CIMValue::CIMValue *Constructor*
  - Programming Pegasus::Class Definitions::CIMValue::CIMValue *Constructor*
  - Programming Pegasus::Class Definitions::CIMValue::CIMValue *Constructor*
  - Programming Pegasus::Class Definitions::CIMValue::CIMValue *Constructor*
  - Programming Pegasus::Class Definitions::CIMValue::CIMValue *Constructor*
  - Programming Pegasus::Class Definitions::CIMValue::CIMValue *Constructor*
  - Programming Pegasus::Class Definitions::CIMValue::CIMValue *Constructor*
  - Programming Pegasus::Class Definitions::CIMValue::CIMValue *Constructor*
  - Programming Pegasus::Class Definitions::CIMValue::CIMValue *Constructor*
  - Programming Pegasus::Class Definitions::CIMValue::CIMValue *Constructor*
  - Programming Pegasus::Class Definitions::CIMValue::CIMValue *Constructor*
  - Programming Pegasus::Class Definitions::CIMValue::CIMValue *Constructor*
  - Programming Pegasus::Class Definitions::CIMValue::CIMValue *Constructor*
  - Programming Pegasus::Class Definitions::CIMValue::CIMValue *Constructor*

- Programming Pegasus::Class Definitions::KeyBinding::setType *Modifier*
- Programming Pegasus::Class Definitions::KeyBinding::setValue *Modifier*
- Programming Pegasus::Class Definitions::KeyBinding::typeToString *Converts the given type to one of the following: "boolean", "string", or "numeric"*
- Programming Pegasus::Class Definitions::KeyBinding::~KeyBinding *Destructor*
- Programming Pegasus::Class Definitions::Stopwatch::Stopwatch *stopwatch constructor.*
- Programming Pegasus::Class Definitions::Stopwatch::getElapsed *getElapsed - Get the elapsed time for the defined stopwatch.*
- Programming Pegasus::Class Definitions::Stopwatch::printElapsed *printElapsed method sends the current value of the timer and sends it to standardout as a string with the word seconds attached*
- Programming Pegasus::Class Definitions::Stopwatch::reset *Reset Stopwatch resets an existing Stopwatch object to the current time value*
- **String:**
  - Programming Pegasus::Class Definitions::String::String *Initialize with first n characters of x*
  - Programming Pegasus::Class Definitions::String::String *Copy constructor*
  - Programming Pegasus::Class Definitions::String::String *Initialize with first n characters from x*
  - Programming Pegasus::Class Definitions::String::String *Initialize with x*
  - Programming Pegasus::Class Definitions::String::String *Default constructor without parameters.*
  - Programming Pegasus::Class Definitions::String::String *Initialize from a plain old C-String:*
  - Programming Pegasus::Class Definitions::String::String *Initialize from the first n characters of a plain old C-String:*
- Programming Pegasus::Class Definitions::String::allocateCString *Allocates an 8 bit representation of this string.*
- **append:**
  - Programming Pegasus::Class Definitions::String::append *Append the characters of str to this String object*
  - Programming Pegasus::Class Definitions::String::append *Append the given character to the string.*
  - Programming Pegasus::Class Definitions::String::append *Append n characters from str to this String object*
- Programming Pegasus::Class Definitions::String::appendToCString *Append the given string to a C-string.*
- **assign:**
  - Programming Pegasus::Class Definitions::String::assign *Assign this string with x*
  - Programming Pegasus::Class Definitions::String::assign *Assign this string with x*
  - Programming Pegasus::Class Definitions::String::assign *Assign this string with first n characters of x*
  - Programming Pegasus::Class Definitions::String::assign *Assign this string with the plain old C-String x*
  - Programming Pegasus::Class Definitions::String::assign *Assign this string with first n characters of the plain old C-String x*
- Programming Pegasus::Class Definitions::String::clear *Clear this string.*
- **compare:**
  - Programming Pegasus::Class Definitions::String::compare *Compare two null-terminated strings.*
  - Programming Pegasus::Class Definitions::String::compare *Compare the first n characters of the two strings.*
- **equal:**
  - Programming Pegasus::Class Definitions::String::equal *Compare two String objects for equality.*
  - Programming Pegasus::Class Definitions::String::equal *Return true if the two strings are equal*

- Programming Pegasus::Class Definitions::String::equal *Return true if the two strings are equal*
- Programming Pegasus::Class Definitions::String::equal *Return true if the two strings are equal*
- Programming Pegasus::Class Definitions::String::equal *Return true if the two strings are equal*

- **find:**
  - Programming Pegasus::Class Definitions::String::find *find substring*
  - Programming Pegasus::Class Definitions::String::find *Find the position of the first occurence of the string object.*
  - Programming Pegasus::Class Definitions::String::find *Find substring @ param - 16 bit character pointer*
  - Programming Pegasus::Class Definitions::String::find *Find the position of the first occurence of the character c.*
- Programming Pegasus::Class Definitions::String::getData *Returns a pointer to the first character in the null-terminated string string.*
- **operator+=:**
  - Programming Pegasus::Class Definitions::String::operator+= *Overload operator += appends the parameter String to this String.*
  - Programming Pegasus::Class Definitions::String::operator+= *Append the character given by c to this String object.*
  - Programming Pegasus::Class Definitions::String::operator+= *Append the character given by c to this string.*
- **operator=:**
  - Programming Pegasus::Class Definitions::String::operator= *Assign this string with x.*
  - Programming Pegasus::Class Definitions::String::operator= *Assign this string with x*
- **operator[]:**
  - Programming Pegasus::Class Definitions::String::operator[] *Returns the Ith character of the String (const version).*
  - Programming Pegasus::Class Definitions::String::operator[] *Returns the Ith character of the String object.*
- Programming Pegasus::Class Definitions::String::remove *Remove size characters from the string starting at the given position.*
- Programming Pegasus::Class Definitions::String::reserve *Reserves memory for capacity characters.*
- Programming Pegasus::Class Definitions::String::reverseFind *Same as find() but start looking in reverse (last character first).*
- Programming Pegasus::Class Definitions::String::size *Returns the length of the String object.*
- Programming Pegasus::Class Definitions::String::subString *Return a new String which is initialzed with* `length` *characters from this string starting at* `pos`.
- **toLower:**
  - Programming Pegasus::Class Definitions::String::toLower *Convert the plain old C-string to lower case:*
  - Programming Pegasus::Class Definitions::String::toLower *Converts all characters in this string to lower case*
- Programming Pegasus::Class Definitions::String::translate *Translate any occurences of fromChar to toChar*
- Programming Pegasus::Class Definitions::String::~String *String destructor.*
- ToLower *Return a version of this string whose characters have been shifted to lower case*
- operator!= *String operator ==.*
- operator+ *overload operator + - Concatenates String objects.*
- operator< *overload operator < - Compares String obects.*

- operator<= *overload operator <= compares String objects.*
- operator== *String operator ==.*
- operator== *String operator ==.*
- operator== *String operator ==.*
- operator> *Overload operator > compares String objects*
- operator>= *overload operator >= - Compares String objects*

Variables

- PEGASUS_NAMESPACE_END
- PEGASUS_NAMESPACE_END
- PEGASUS_NAMESPACE_END
- PEGASUS_NAMESPACE_END
- Programming Pegasus::Class Definitions::CIMDateTime::x
- PEGASUS_NAMESPACE_END
- Programming Pegasus::Class Definitions::String::EMPTY *EMPTY - Represent an empty string.*

Macros

- PEGASUS_ASSERT *define PEGASUS_ASSERT assertion statement.*
- Pegasus_Exception_h *Programming with Exceptions*

Enums, Unions, Structs

- CIMFlavor *CIMQualifier flavor constants*

Hierarchy of classes

Credits

Credit for work on the Pegasus implementation so far goes largely to the following companies and personnel:

- Michael Brasher - BMC, Chief Architect and principle Programmer *m.brasher@bmc.com*
- Bob Blair - BMC, Developer of the Pegasus Compiler *bblair@bmc.com*
- Mike Reynolds - BMC
- Heather Kreger - Originator of the extension services concept and one of our original believers in the concepts of Manageability *kreger@us.ibm.com*
- Martin Kirk - The Open Group, Open Group Enterprise Management Forum Program Manager, The Open Group *m.kirk@opengroup.org*
- Karl Schopmeyer - Formerly Tivoli Systems and now The Open Group, Open group Enterprise Management Forum Chair and Workgroup leader for the Pegasus Project *k.schopmeyer@opengroup.org*
- Raymond Williams - Tivoli Systems and VP Engineering DMTF for encouragement, direction, and support. *Raymond_Williams@tivoli.com*
- Mike Lambert, Open Group - For supporting and encouraging this effort *m.lambert@opengroup.org*

Get your name on this list. Support, contribute to and use Pegasus.

Alphabetic index *Hierarchy of classes*

Hierarchy of Classes

- Array
- CIMClass
- CIMConstClass
- CIMDateTime
- CIMInstance
- CIMMethod
- CIMName
- CIMOperations
    - CIMClient
    - CIMRepository
- CIMProperty
- CIMQualifier
- CIMQualifierDecl
- CIMReference
- CIMType
- CIMValue
- Char16
- Exception
    - AddedReferenceToClass
    - AlreadyExists
    - AssertionFailureException
    - BadQualifierOverride
    - BadQualifierScope
    - BadReference
    - CIMException
    - CannotCreateDirectory
    - CannotOpenFile
    - ClassAlreadyResolved
    - ClassNotResolved
    - ExpectedReferenceValue
    - FailedToRemoveDirectory

Definition of Terms

This section provides definitions of terms and concepts used in the specification of the DMTF CIM and WBEM specifications and of the Pegasus Managability Services Broker and its implementation.

Namespace - An object that defines a scope within which object keys must be unique. Multiple namespaces typically indicate the presence of multiple management spaces or multiple devices.

The namespace pragma

```
#pragma namespace( )
```

This pragma is used to specify a Namespace path. The syntax needs to conform to the following:

```
://
```

The contents of a MOF file are loaded into a namespace that provides a domain (in other words, a container), in which the instances of the classes are guaranteed to be unique per the KEY qualifier definitions. The term namespace is used to refer to an implementation that provides such a domain. Namespaces can be used to:

- Define chunks of management information (objects and associations) to limit implementation resource requirements, such as database size.
- Define views on the model for applications managing only specific objects, such as hubs.
- Pre-structure groups of objects for optimized query speed.

CIMOM = TBD Definition of a CIMOM

Repository - The CIM Repository stores the CIM metaschema (class repository) and instance information (instance repository). TheRepository is provided as part of the reference implementation, but is considered an independent service for the CIMOM. The interfaces to the respository are fixed but the supplied respository can be replaced with a different implementation.

PROPERTY - ATTN:

FLAVOR \Label{FLAOVOR - ATTN:

CIM INSTANCE - ATTN:

MOF - MOF is the standard language used to define elements of the Common Information Model (CIM). The MOF language specifies syntax for defining CIM classes and instances. Compilation of MOF files provides developers and administrators with a simple and fast

technique for modifying the CIM Repository. For more information about MOF, see the DMTF web page at http://www.dmtf.org.

Association -An association is a relationship between two classes or between instances of two classes. The properties of an association class include pointers, or references, to the two classes or instances.

Class association is one of the most powerful CIM features. It provides a way to organize a collection of management objects into meaningful relationships. For example, a CIM_ComputerSystem object might contain a CIM_Disk, Processor A, and Processor B. The CIM_ComputerSystem has an association with each of the objects it contains. Because this particular association is a containment association, it is represented by a class called CIM_contains. The CIM_contains class contains references to each of the objects that belong to the association. In this case, CIM_contains has a reference to Solaris_Disk and a reference to CIM_ComputerSystem.

WBEM - ATTN:

DMTF - The Distributed Management Task Force (DMTF), a group representing corporations in the computer and telecommunications industries, is leading the effort to develop and disseminate standards for management of desktop environments, enterprise-wide systems, and the Internet. The goal of the DMTF is to develop an integrated approach to managing networks across platforms and protocols, resulting in cost-effective products that interoperate as flawlessly as possible. For information about DMTF initiatives and outcomes, see the DMTF web site at http://www.dmtf.org.

MOF - MOF is the standard language used to define elements of the Common Information Model (CIM). The MOF language specifies syntax for defining CIM classes and instances. Compilation of MOF files provides developers and administrators with a simple and fast technique for modifying the CIM Repository. For more information about MOF, see the DMTF web page at http://www.dmtf.org.

XmlCIM - ATTN: The XML Schema describes the CIM metaschema. The CIM classes and instances are valid XML documents.

A Metaschema Mapping is one where the XML schema is used to describe the CIM metaschema, and both CIM classes and instances are valid XML documents for that schema. In this case, the DTD is used to describe, in a generic fashion, the notion of a CIM class or instance. CIM element names are mapped to XML attribute or element values. An alternate mechanism maps XML documents directly to CIM schema classes and instances. In this case, CIM element names are mapped to XML element names The advantages of the XML descriptions are: Only one DTD needs to be defined Avoids the limitations of DTD's (ordering, scoping, and no inheritance) Straightforward

For more detailed information on the mapping of XML to CIM schema, see http://www.dmtf.org/download/spec/xmls/CIM_XML_Mapping20.htm and http://www.dmtf.org/download/spec/xmls/CIM_HTTP_Mapping10.htm. 1.5 Managed Object Format

QUALIFIERR - Qualifiers are values that provide additional information about classes, associations, indications, methods, method parameters, triggers, instances, properties or references. All qualifiers have a name, type, value, scope, flavor and default value. Qualifiers cannot be duplicated; there cannot be more than one qualifier of the same name for any given class, instance, or property.

Qualifiers are defined in detail in the DMTF CIM Specificaiton.

There are several types of Qualifiers as listed below:

- **Meta-Qualifiers** - (ASSOCIATION,INDICATION are the MetaQualifiers) are used to refine the definition of the meta constructs in the model. These qualifiers are used to refine the actual usage of an object class or property declaration within the MOF syntax.
- **StandardQualifiers** -(See CIM Specificaiton for a list) All CIM-compliant implementations are required to handle. Any given object will not have all of the qualifiers listed.
- **Optional Qualifiers** - The optional qualifiers listed in the CIM Specificaiton address situations that are not common to all CIM-compliant implementations. Thus, CIM-compliant implementations can ignore optional qualifiers since they are not required to interpret or understand these qualifiers.

  PEGASUS Implements the following optional Qualifiers - ATTN:

- User Defined Qualifiers - The user can define any additional arbitrary named qualifiers. However, the CIM specificaiton recommends that only defined qualifiers be used, and that the list of qualifiers be extended only if there is no other way to accomplish a particular objective.

ATTN: Should we include the table of qualifiers?????

Aggregation -A strong form of an association. For example, the containment relationship between a system and the components that make up the system can be called an aggregation. An aggregation is expressed as a Qualifier on the association class. Aggregation often implies, but does not require, that the aggregated objects have mutual dependencies.

Association - A class that expresses the relationship between two other classes. The relationship is established by the presence of two or more references in the association class pointing to the related classes.

Cardinality - A relationship between two classes that allows more than one object to be related to a single object. For example, Microsoft Office* is made up of the software elements Word, Excel, Access and PowerPoint.

CIM - Common Information Model is the schema of the overall managed environment. It is divided into a Core model, Common model and extended schemas. CIM Schema The schema representing the Core and Common models. Versions of this schema will be released by the DMTF over time as the schema evolves.

Class - A collection of instances, all of which support a common type; that is, a set of

properties and methods. The common properties and methods are defined as features of the class. For example, the class called Modem represents all the modems present in a system. Common model A collection of models specific to a particular area, derived from the Core model. Included are the system model, the application model, the network model and the device model.

Core model - A subset of CIM, not specific to any platform. The Core model is set of classes and associations that establish a conceptual framework for the schema of the rest of the managed environment. Systems, applications, networks and related information are modeled as extensions to the Core model.

Domain - A virtual room for object names that establishes the range in which the names of objects are unique.

Explicit Qualifier - A qualifier defined separately from the definition of a class, property or other schema element (see implicit qualifier). Explicit qualifier names must be unique across the entire schema. Implicit qualifier names must be unique within the defining schema element; that is, a given schema element may not have two qualifiers with the same name.

Extended schema - A platform specific schema derived from the Common model. An example is the Win32 schema.

Feature - A property or method belonging to a class.

Flavor - Part of a qualifier spcification indicating overriding and inheritance rules. For example, the qualifier KEY has Flavor(DisableOverride ToSubclass), meaning that every subclass must inherit it and cannot override it.

Implicit Qualifier - A qualifier defined as a part of the definition of a class, property or other schema element (see explicit qualifier).

Indication - A type of class usually created as a result of the occurrence of a trigger.

Inheritance - A relationship between two classes in which all the members of the subclass are required to be members of the superclass. Any member of the subclass must also support any method or property supported by the superclass. For example, Modem is a subclass of Device.

Instance - A unit of data. An instance is a set of property values that can be uniquely identified by a key.

Key - One or more qualified class properties that can be used to construct a name. One or more qualified object properties which uniquely identify instances of this object in a namespace.

Managed Object - The actual item in the system environment that is accessed by the provider. For example, a Network Interface Card.

Meta model - A set of classes, associations and properties that expresses the types of things that can be defined in a Schema. For example, the meta model includes a class called

property which defines the properties known to the system, a class called method which defines the methods known to the system, and a class called class which defines the classes known to the system.

Meta schema**Meta schema** - The schema of the meta model. Method A declaration of a signature; that is, the method name, return type and parameters, and, in the case of a concrete class, may imply an implementation.

METHOD - Methods represent the behavior relevant for a class. A method is defined as an operation together with its signature. The signature consists of a possibly empty list of parameters and a return type.

A declaration of a signature; that is, the method name, return type and parameters, and, in the case of a concrete class, may imply an implementation.

Model - A set of classes, properties and associations that allows the expression of information about a specific domain. For example, a Network may consist of Network Devices and Logical Networks. The Network Devices may have attachment associations to each other, and may have member associations to Logical Networks.

Model Path - A reference to an object within a namespace. Namespace An object that defines a scope within which object keys must be unique.

Namespath Path - A reference to a namespace within an implementation that is capable of hosting CIM objects.

Name - Combination of a Namespace path and a Model path that identifies a unique object.

Trigger - The occurrence of some action such as the creation, modification or deletion of an object, access to an object, or modification or access to a property. Triggers may also be fired as a result of the passage of a specified period of time. A trigger typically results in an Indication.

- A subclass may redefine the implementation of a method or property inherited from its superclass. The property or method is thereby redefined, even if the superclass is used to access the object. For example, Device may define availability as a string, and may return the values “powersave”, "on" or "off." The Modem subclass of Device may redefine (override) availability by returning "on," "off," but not "powersave". If all Devices are enumerated, any Device that happens to be a modem will not return the value "powersave" for the availability property.

Property - A value used to characterize an instance of a class. For example, a Device may have aproperty called status.

Provider - An executable that can return or set information about a given managed object. Qualifier A value used to characterize a method, property, or class in the meta schema. For example, if a property has the qualifier KEY with the value TRUE, the property is a key for the class.

Reference - Special property types that are references or "pointers" to other instances.

Schema - A namespace and unit of ownership for a set of classes. Schemas may come in forms such as a text file, information in a repository, or diagrams in a CASE tool.

Scope Scope - Part of a Qualifier specification indicating with which meta constructs the Qualifier can be used. For example, the Qualifier ABSTRACT has Scope(Class Association Indication), meaning that it can only be used with Classes, Associations and Indications.

Scoping Object - Objects which represent a real-world managed element, which in turn propagate keys to other objects.

Signature - The return type and parameters supported by a method.

Subclass - See Inheritance.

Superclass - See Inheritance.

Top Level Object - A class or object that has no scoping object.

- **NAMESPACE.**

Alphabetic index *Hierarchy of classes*

NAMESPACE.

Defintion of the namespace.

Alphabetic index *Hierarchy of classes*

Document References

This section will contain references to external documents.

NOTE: documents we must reference include:

- DMTF Specifications
-

The following is some test tags to find problems and techniquest in DOC++ There is the label FOO

Here is the reference to Pegasus, the top fo the document Pegasus

Here is a reference to the label FOO "FOO" "FOO" End of all this

Alphabetic index *Hierarchy of classes*

Objectives

Objectives of the Pegasus Project

The Pegasus project was initiated by the Open Group to provide a means to define and clarify new standards for the manageability of IT environments.

The Project has the following major objectives:

- Creation of a prototype environment for a manageability architecture that uses existing and emerging standards.
- Provide a manageability implementation that is sufficiently rich and efficient that it can be used as the basis for production environments.

  Provide a manageability environment that can be used to test and demonstrate components of the manageability architecture. This would allow us not only to create specifications but to create demonstratable solutions and to test those solutions agains the specifications. We have found in the last few years that creating standards and specificaitons alone is not enough. These solutions will only be used if there is real supporting code. Further, we have come to beleive that we cannot really understand the correctness or effectiveness of standards without implementations.
- Create standards and specifications for manageability APIs
- Contribute back to the DMTF i

Manageability

We see the problem of managing IT environments as having several major components:

- The Management environment - This is represented by the systems that perform the tasks of managing our IT environment. There are a number of major and minor suppliers of systems of this type, either for enterprise wide management or for managemetn of specific tasks, functions, applicaitons, systems, networks, etc.
- The Manageability environment - The management the interfaces the managed resources with management. This includes instrumentation for capture of information, functions to execute tasks within the environment and additional facilities to make this all available to the Management envirionment
- The management/manageability interface - This is the key interface that allows the sepeartion of manageability and management. This interface must define the information and protocols for the passage of management information between the manageability environment and the management environment
- The management/Instrumentation Interface - ATTN: Explain this one.

WHY SEPARATE THE TWO

ATTN: add this

ADDITIONAL COMMENTS THAT NEED TO BE MADE:

Today, these two environments are heavily integrated. Most management systems include

thier own manageability components (agents, information capture tools, control tools). Thus, the commitment to a management system is also the commitment to instrumentation.

The objective of standards like SNMP and WBEM is to provide the standards to separate manageability from Management.

Alphabetic index *Hierarchy of classes*

Overview

Pegasus is an implementation of the components of a CIM environment as defined by the DMTF http://www.dmtf.org

The OpenGroup is in the process of creating standard APIs for an implementation of DMTF CIM Server that will be known as Pegasus.

This will include an OpenGroup Specification of the architecture and interfaces and one or more implementations that are available as opensource from http://www.opengroup.org.

Pegasus is an implementation of the components of a CIM environment as defined by the DMTF http://www.dmtf.org

ATTN: Not finished

Alphabetic index *Hierarchy of classes*

Pegasus Architecture

This section defines the overall architecture of the Pegasus implementation.

Alphabetic index *Hierarchy of classes*

Design Goals

The Pegasus design team set some basic design goals early in the development of Pegasus as follows:

- C++ as the core development language. We selected C++ because it represented a compromise between the ability to work with objects and a language that would be acceptable for high avaailablity platforms.
- Modular Architecture - We wanted to be able to create the architecture based on well understand standardized modules that have clean well defined interfaces between the modules.
- Open to a wide range of specialization and customization.
- Minimize the functionality of the basic core CIMOM. We wanted to create an environment where the majority of customization could be created by working with attached modules that would both extend and modify the functionality of the broker.
- Use only open source components in the pegasus environment
- Design for maximum portability. The initial targets would be Linux, Unix, and NT but the product should be usable in a wide variety of platforms and platform sizes.
- The APIs and interfaces should be clear, given that they are in C++
-

Alphabetic index *Hierarchy of classes*

Extension Services

ATTN: Document this as an architectural component

Alphabetic index *Hierarchy of classes*

Functional Flow

The Common Information Model Object Broker (often known as the CIM Information manager or CIMOM) brokers CIM objects between a number of sources and destinations. A CIM object should be a representation, or model, of a managed resource, such as a printer, disk drive, or central processing unit (CPU). In the Pegasus implementation, CIM objects are represented internally as C++ classes. The CIMOM transfers information between WBEM clients, the CIM Object Manager Repository, and managed resources.

NOTE: We are very careful in the use of CIM and WBEM. These are terms defined and controled by the DMTF and they have specific meanings both technically and legally. Thus, the objects are CIM objects. However, the client is a WBEM client because it uses the DMTF XML/HTTP specificaitons to transfer information and that specification and CIM form WBEM.

When a WBEM client application accesses information about a managed resource, the CIM Object Manager contacts either the appropriate provider for the CIM object that represents that managed resource or the CIM Object Manager Repository. Providers are classes that communicate with managed objects to retrieve data. If the requested data is not available from the CIM Object manager Repository, the CIM Object Manager forwards the request to the provider for that managed resource.

Using the Repository.

The provider dynamically retrieves the requested information, which is sent back to the requester. The CIM Object Manager Repository only contains static data. Classes that are handled by a provider must have a Provider qualifier that identifies the provider to contact for the class. When the CIM Object Manager receives a request for a class that has a Provider qualifier, it should route the request to the specified provider. If no provider is specified, it should route the request to the CIM Object Manager Repository.

When a WBEM client connects to a CIM Object Manager, it will get a handle to the CIM Object Manager. The client can then perform WBEM operations.using this reference. At startup, the CIM Object Manager should perform the following functions: Listen for RMI connections on RMI port 5987 and for XML/HTTP connections on HTTP.

NOTE: The current version of the CIMOM does not incorporate events. Therefore, this description is written around a CIMOM without events functionality.

Note - The listener for connections may not be the Object Manager; it could be another entity that is performing the operation for the Object Manager. This could be a servlet in a Web server. Conformant object managers are required to support XML over HTTP - Pegasus is conformant.

The CIMOM accepts requests called WBEM Operations from the WBEM client. These operations are explicitly defined in the WBEM specification. They represent the operations possible on CIM objects (ex. create/modify/delete class/instance, etc.) During normal operations, the CIMOM performs the following for each operation request received:

- Security checks to authenticate user login and authorization to access the CIMOM information.
- Syntactic and semantic checks of the CIM data operations to ensure that they comply with the current version of the CIM specification.
- Route requests to the appropriate provider orthe Repository. The CIMOM iteself does not serve as a Repository for CIM class definitions and instance data. Persistence is provided by the Repository; however, the contact point is the CIMOM. Thus, the Repository could be considered as an option except that the CIMIM is required to keep class information for all semantic and syntatic confirmation and therefore the class repository is a requirement of a working CIMOM, not an option.
- Deliver data from providers and from the CIM Object Manager Repository to the originating WBEM client application.

The CIMOM should be a process that accepts requests for CIM operations, as defined by the DMTF, and carries out these operations. The Pegasus CIMOM runs as a daemon process that waits for requests.

Authentication

Before any requests can be made to the CIM Object Manager, an authenticated session must be established.

NOTE:The current version of Pegasus does not have any authentication. However, it is planned for version 1.1.

An identifier for the user and optionally a role will be associated with the authenticated session. A role is a principal identity associated with the current session, in addition to the user identity. Systems that do not support roles can ignore them as described in the Security Interface. These can be maintained in an internal Hash map.

Request Reception

The CIMOM receives requests through CIM operations over HTTP. Each request will be associated with a session that is set up as part of the initial authentication exchange. Since the session has an associated user, each request automatically has a user associated with it. This should be useful for authorization checking for a given request. Once the request has been received, the appropriate components for handling the specific request will be invoked. The Pegasus implementation has methods for each of the major CIM operations over HTTP. Once the request is received, the appropriate method will be called..

Authorization

The default implementation is Access Control List (ACL) based. Access control lists can be maintained per namespace or on a per namespace/user basis. These lists will be maintained in the root/security namespace. The CIM Object Manager will grant read or write permissions within a namespace based on the access control list. Since CIM operations are done within the context of a namespace, these ACLs will enforce rules on whether an operation should be allowed. For operations that will ultimately be handled by a provider, the appropriate provider can replace the authorization scheme. This will allow providers to

enforce finer grained control if desired. A provider can replace the default authorization checking scheme by implementing the Authorizable interface. If implemented, no calls are made to the CIM Object Manager.

Provider

Provider RegistrationB/B>

The Pegasus CIMOM enables developers to write providers, which serve dynamic information to the CIMOM (see Providers). Providers register themselves by specifying their location in a Provider qualifier. Providers can be set up on a class, property, or method basis. Providers can have one or more of the different provider types. The DMTF CIM specification allows the Provider qualifier to have an implementation specific interpretation. For Pegasus, the Provider qualifier constitutes the executable name of a provider executable implementing the provider functions for the class.

There are a number of conceptual interfaces that can be implemented by providers:

- **InstanceProvider**
- **MethodProvider**
- **PropertyProvider**
- **AssociatorProvider**

. Each conceptual interface provides a subset of the WBEM Operations as follows:

NOTE: ATTN: Table defining the types vs. operations

However

Providers should be loaded "on demand" by the CIMOM. Classes and properties marked by the provider qualifier will be an indication to the object manager that the associated information is dynamic and must be obtained from the providers rather than the repository. When the object manager determines that a specific request needs dynamic data, provider should be loaded and instantiated. Additionally, the "initialize" method of the Provider will be invoked. There should be only a single instance of the provider.

ATTN: Review the following: In the reference implementation, the ProviderChecker maintains a hash map of all the providers. This will enable the CIM Object Manager to load a provider only if it has not been loaded previously. There should be no specified time when a provider can be "unloaded", however providers have a "cleanup" method that can be invoked if, and when, this behavior is specified for the object manager.

The CIM Object Manager will not act as as a provider for classes. However, there are instances where classes must interact with the CIMOM itself. These might include authentication classes, authorization classes, namespace classes, and classes that provide information on the CIMOM iteslf.

These classes will be handled by providers but these will be specialized providers that have access back to the CIMOM itself. All of this is being defined as part of a services extension interface to PEGASUS. This interface will be discussed in a future version of this document:

ATTN: add the services interfaces.

ATTN: Dealing with multiple providers per class.

Request Routing

One of the main functions of the CIMOM is operation request routing. Depending on the request, the request may need to be authorized and passed to semantic checkers, providers, and the repository.

Requests may be for static information such as schema definitions or static instances. In this case, the CIMOM should route the request to the proper repository.

The more complex routing will involve operations that can traverse multiple classes and their instances. An example of such an operation is association traversal. In order to determine the associated instances of a given input instance, the CIMOM should first determine the associations that the given instance class participates in. It will obtain this from the associations that have been compiled and stored in the repository. Once these associations are determined, the CIM Object Manager should find those instances of the associations in which the given input instance plays a role. These associations may, or may not be, dynamic. Depending on whether the associations are dynamic or not, the CIM Object Manager may route the requests to providers or the repository. Once the results are returned, they should be concatenated together and returned because of the request. The CIM Object Manager will use schema information to determine which providers to contact. As can be seen, a given request can result in multiple sub-requests to the providers or the repository. A similar situation will occur when a deep enumeration is performed on instances of a class.

Semantic Checking

The CIMOM performs semantic checks before classes or instances can be set or createdusing internal class, property, instance, method, and qualifier checkers and the rules ov validation defined by the CIM specification. These verifiers ensure that the CIM rules are enforced. This includes type verification,type conversions, verification of proper key usage, and other checks.,

Alphabetic index *Hierarchy of classes*

Pegasus Clients.

ATTN: define in more detail

Alphabetic index *Hierarchy of classes*

Pegasus Providers.

ATTN: Define in more detail

Alphabetic index *Hierarchy of classes*

Pegasus Documentation

The Broker

The Common Information Model Object Broker (often known as the CIM Information manager or CIMOM) brokers CIM objects between a number of sources and destinations. A CIM object should be a representation, or model, of a managed resource, such as a printer, disk drive, or central processing unit (CPU). In the Pegasus implementation, CIM objects are represented internally as C++ classes. The CIMOM transfers information between WBEM clients, the CIM Object Manager Repository, and managed resources.

Alphabetic index *Hierarchy of classes*

Pegasus Code Examples

The following sections provide some examples of the use of the Pegasus APIs for clients, providers and services.

There are also a number of good examples of Pegasus API and Class usage in the tests defined for the major Pegasus components. See the tests directories for a list of the tests currently implemented.

- **Client Coding Examples**
- **Provider Coding Examples**

Alphabetic index *Hierarchy of classes*

Client Coding Examples

Example 1

Declaration of a Class and the addition of qualifiers, properties, and methods to that class. Finally, the code converts the class defined to XML with print, outputs it, restores it to a new class and compares the results.

This example is one of the tests defined for Pegasus in the /commmon/tests directory

```
Example of Class Declaration


#include
#include


using namespace Pegasus;


void test01()
{
    ClassDecl class1("MyClass", "YourClass");


    class1
        .addQualifier(Qualifier("association", true))
        .addQualifier(Qualifier("q1", Uint32(55)))
        .addQualifier(Qualifier("q2", "Hello"))
        .addProperty(Property("message", "Hello"))
        .addProperty(Property("count", Uint32(77)))
        .addProperty(
            Property("ref1", Reference("MyClass.key1=\"fred\""), "MyClass"))
        .addMethod(Method("isActive", Type::BOOLEAN)
            .addParameter(Parameter("hostname", Type::STRING))
            .addParameter(Parameter("port", Type::UINT32)));


    // class1.print();


    OutBuffer out;
    out << class1;


    InBuffer in(out.getData());
    ClassDecl tmp;
    in >> tmp;


    assert(class1.identical(tmp));
}


int main()
{
```

```
    try
    {
        test01();
    }
    catch (Exception& e)
    {
        cout << "Exception: " << e.getMessage() << endl;
    }

    cout << "+++++ passed all tests" << endl;

    return 0;
}
```

Alphabetic index *Hierarchy of classes*

Provider Coding Examples

ATTN: This section empty for the moment

Alphabetic index *Hierarchy of classes*

Pegasus Components

This section defines the major components that make up the Pegasus environment, the architectureal components, the built components, and the supporting directory structure.

- **Component Descriptions**
- **Pegasus Directory Structure**

Alphabetic index *Hierarchy of classes*

Component Descriptions

The major components of Pegasus are:

Pegasus Server - WBEM/CIM CIM_Server with interfaces for providers and Clients

Pegasus Repository - Today Pegasus provides a defined class repository interface and a simple file based class repository. Future will include the object repository

Pegasus Client SDK - Tools for building Pegasus clients based on the Pegasus C++ interfaces and using the WBEM HTTP/XML protocols or directly interfacing with Pegasus

Pegasus Test Clients - Simple test clients being developed as part of the Pegasus development process

Pegasus HTML Test Client - To aid some testing we created a test client for Pegasus that uses a WEB server (ex. Apache) with a set of CGI modules and HTML to allow the entry of Pegasus operations from a WEB browser as forms and the recept of the response as WEB pages. This has proven useful as a test tool and can be used for a wide variety of demonstrations.

Pegasus Providers - Pegasus providers are build as separate components that can be dynamically loaded by the Pegasus server.

Pegasus Service Extensions - Future (version 1.0)

Pegasus MOF Compiler - The Pegasus MOF compiler compiles MOF files and installs them into the Pegasus repository.

ATTN: Define the files that make up these components and their structure.

Alphabetic index *Hierarchy of classes*

Pegasus Directory Structure

Pegasus is distributed as a complete directory structure that should be installed either from one of the snapshots or from CVS.

This structure is generally as follows:

```
Pegasus                          - Root directory
    bin
    build                    Destination for all intermediate files from build
                             if no alternative is specified see PEGASUS_HOME


    cgi-bin                  Source and make for the Pegasus WEB Based Test client
software
    doc                      Miscelaneous Pegasus Documents. Includes the DMTF XML for
CIM 2.4
    html                     HTML files for the Browser test client.
    mak                      General make files (used by the root make and other
makes)
    Repository               This Directory containes the created repository
    src                      All Pegasus Source Files
        ACEExamples          Test directrory with examples of the use of ACE
(developers)
        Clients              Top level directory for Pegasus Client Programs
           CGI_Client        Source for the Pegasus client for the WEB demo


        Pegasus
            CGI              CGI files for the WEB test client
                CGIClient
            Client           Pegasus Client SDK and Test client using the SDK
                tests        Test programs for the client software
            Common           Pegasus Common Functions (C++ source and headers
                tests        Test programs for the common functions
            Protocol         Pegasus Client HTTP/XML Protocol Modules
                tests
            Repository       Pegasus Repository Interfaces and Simple Repository
                tests        Tests for Repository Functions
            Server           Pegasus Server Modules
                tests        Unit tests defined for the server functions


        Providers            Top Level Directory for Pegasus written Providers
            Generic          Non-system oriented providers
            Windows          Providers defined for the Windows environment
            Unix             Providers defined for the Unix environment
        Services             To-be-defined.


        Utils
    manual                   Pegasus User/developer manual source modules
        HTML                 Output from the Pegasus Manual compilartion.
```

Alphabetic index *Hierarchy of classes*

Pegasus Interfaces

The Pegasus MSB interfaces with several different entities:

- Providers
- Services (Including respositories and other services)
- Repository

Further, it includes both the APIs and the definitions of CIM data that are required by the broker.

Pegasus has effectivly implemented the same operations and data defintions across all of these interfaces with minor differences because of the special characteristics of each interface.

CIM Operations over HTTP The core operations implemented in Pegasus are based entirely on the CIM operations over HTTP definitions by the DMTF in CIM Operations over HTTP specificaiton.

The creation of all Pegasus interfaces is based on this core was a key objective in the design of Pegasus. These operations provide the creation, deletion, manipulation of CIM classes and objects and their components.

Pegasus implements these operations as methods in the Pegasus class Operations The interfaces for this class and methods is defined in the header file operations.h.

Client Interface In Pegasus the client is a separate component that can communicate with the Pegasus server either through the WBEM CIM/XML protocol or directly through the Pegasus Client C++ Interfaces.

The Client interfaces are an extension of the Pegasus Operations interfaces with some specific methods added for client/server communication control.

Provider Interfaces

In Pegasus, the provider is a separate executable that accesses the managed resources and is used by the CIMOM to provide access to data. Providers forward this information to the CIMOM for integration and interpretation. When the CIMOM receives a request from a management application for data that is not available from the CIMOM Repository, it forwards the request to a provider. The CIMOM Repository only contains static data. Providers implement a provider interface that supports the type of service specific to their role. In order to implement the interface, a provider class must first declare the interface in an implements clause, and then it must provide an implementation (a body) for all of the abstract methods of the.interface.

ATTN: Need to put something here.

Repository Interfaces

The repository interface isused by the CIMOM to interface with implementations of the

repository thatstore and retrieve provide persistence for class and instance information.

A prototype implementation of both a class and instance provider is provided with Pegasus (ATTN: SEE ALSO). However, it is expected that this will be replaced in many installations. .

- **CIM Operations over HTTP**
- **Pegasus Client Interfaces**
- **Pegasus Provider Interfaces**
- **Pegasus Service Extension Interfaces**
- **Repository Interfaces**

Alphabetic index *Hierarchy of classes*

CIM Operations over HTTP

The core of the Pegasus external interface is the WBEM Operations (Formally known as CIM Operations over HTTP in the DMTF specificaion). Pegasus uses the basic WBEM operations as the basis for all of the interfaces.

Pegasus defines these interfaces in exactly the same manner as they are defined in the DMTF specification (CIM Operations over HTTP)

Thus, Pegasus implements a Class named CIMOperations. Within that class, each of the WBEM HTTP operations is implemented as a method.

For example, the WBEM operation GetClass has its direct equivalent in Pegasus CIMOperations, the getClass method. Further, Pegasus maintains the same parameters as are defined in the DMTF document.

The CIMOperations class is the heart of all of the CIM interfaces including the Client interface, the Provider interface, the repository interface, and the services interface. Each of these interfaces derives from CIMOperations adding methods as required to complete that particular interface.

This section defines these interfaces in the reference CIMOperations below:

- **CIMOperations** — *The CIMOperations Class.*
- **PEGASUS_NAMESPACE_END**

Alphabetic index *Hierarchy of classes*

Pegasus Client Interfaces

The DMTF WBEM specifications define an interface between a CIM_Client and a CIM_Server based on XML and HTTP in the WBEM HTTP operations standard, version 1.0 with erratta.

This interface defines the operations required of a CIM Server and the encoding of these operations into XML with the HTTP transmission protocol.

The Pegasus MSB has implemented both that WBEM interface and also an interface based on C++ APIs.

The C++ APIs implement the operations as defined within the WBEM HTTP documentation using effectively the same parameters as the HTTP Operations but with C++ calls and the data support classes implementation the manipulation of CIM information.

The C++ Interface is implemented both at:

- The direct interface to the MSB - a module (esample an executable) that direcetly implementes the C++ data definitions and calls below and links either statically or dynamically to the Pegasus MSB can execute the WBEM Operations. These operations are defined in the Pegasus header file operations.h
- As a client SDK that generates WBEM based XML/HTTP operations

The goal was to create a single API that could be used both locally (dynamically linked to the Pegasus MSB) or remotely through WBEM XML/HTTP interface.

A client program can be built to use either interface simply by changing the build and linking it either directly to the Pegasus MSB library statically or to the Pegasus dll dynamically or to the Pegasus Client SDK library.

ATTN: Somewhere we need to define the files, linkages, etc. in detail.

The interfaces and data definitions are defined in the following sections.

Pegasus Provider Interfaces

ATTN: Today this section is under construction

Today, the CIM/WBEM standards do not define any standards for provider interfaces.

Pegasus implements a set of C++ interfaces for the provider.

In the initial version of Pegasus, these interfaces are implemented as in-process calls from the MSB to the Client.

In future versions of Pegasus, extensions to the provider SDK will be implemented to allow process-process communication and possibly system to system communication between the provider and the MSB.

The Provider Interfaces implement the same operations as the client interface defined in section Pegasus Client Interfaces and the same datatypes.

This interface implements one extra concept to allow the provider to have access to selected information from the MSB. It implements a thread-local set of required information as follows:

ATTN: This needs to be defined

The Provider SDK ATTN: Need to document this.

- **PEGASUS_NAMESPACE_END**

Alphabetic index *Hierarchy of classes*

Pegasus Service Extension Interfaces

Service extensions provide the means to extend the capabilities of the Pegasus MSB. As an example, of a similar concept, Apache uses modules to extend the capabilities of the Apache WEB server.

ATTN: This section is incomplete

Alphabetic index *Hierarchy of classes*

Repository Interfaces

The repository interfaces is based on the general Pegasus operations interface with a selected set of additions for functions required by the repository.

The repository interface is documented in repository.h

The repository interface provides a means to communicate with any number of different repository implementations.

The Pegasus implementation to date as repositories as documented in the repository section of this document. REPOSITORIES

ATTN: Incomplete. ATTN: TODO Today the repository interface header file is built seperatly from the operations.h file, depending the programmer to keep these interfaces in sync.

- **CIMRepository**

  This class derives from the CIMOperations class and provides a simple implementation of a CIM repository.

- **PEGASUS_NAMESPACE_END**

Alphabetic index *Hierarchy of classes*

Pegasus Utilization

This section of the Pegasus working manual defines usage of the Pegasus CIM Server and Provider implementations and thier associated tools.

It provides information on the availability, installation, startup and usage of the components of the Pegasus platform.

Alphabetic index *Hierarchy of classes*

Pegasus Availability

Pegasus is available today in source code form from the Open Group.

Go to http://www.opengroup.org/management to acquire a copy of the current code. The code will be made available in regular snapshots of the source and in addition stable binary releases for platforms of interest.

Pegasus has currently been tested with window and Lnux platforms.

Thie Pegasus code is freely available under the MIT open source license as defined below:

The intention of this licensing is to make Pegasus as widely available as possible without restrictions or limitations.

Alphabetic index *Hierarchy of classes*

Pegasus CIM Clients

Connecting a client

ATTN: Under Construction

Alphabetic index *Hierarchy of classes*

Pegasus Installation

The Installation for Pegasus is currently defined in the readme file in the Pegasus root directory

Alphabetic index *Hierarchy of classes*

Pegasus MOF Compiler

The Pegasus MOF compiler is a command line utility that compiles MOF files (using the MOF format defined by the DMTF CIM Specification) into a Pegasus repository. It allows compiling from strucutures of MOF files using the include pragma and can either compile into a Pegasus repository or simply perform a syntax check on the MOF files.

In the syntax check mode, it checks each class independently and does not do semantic checks between classes (ex. check for super-classes, etc.). When compiling into a Pegasus repository, the compiler uses Pegasus to install the classes and instances into the repository and uses the semantic checking built into Pegasus.

The compiler operates standalone in the syntax checking mode but requires the Pegasus libraries when compiling into a Pegasus respository.

The compiler requires that the input MOF files be in the current directory or that a fully qualified path be given. MOF files included using #pragma include must be in the current directory or in a directory specified by a -I command line switch.

The compiler assumes that the file extension is .mof it is not specified. (This feature is not yet implemented.)

The actual configuration and type of repository created depends on the characteristics of the repository implemented in Pegasus. See the description of the Pegasus repositories for more information.

EXAMPLE

```
cimmof -w -Rtestrepository -I./MOF MOF/CIMSchema25.mof
```

Compile the mof file defined in the directory MOF with the name CIMSchema25.mof and with include pragmas for other MOF files also in that directory and create the repository testrepository

NAME

cimmof - Compile DMTF CIM MOF

SYNOPSIS

```
cimmof [OPTION]... [FILE]...
```

DESCRIPTION

The MOF compiler TBD

OPTIONS

- -h, --help Print out usage message with command line definitions.

- -E - Perform only a syntax check on the input and creates no repository. Inthis mode, the compiler does not do the sematic checks that are done when a CIM object is\ added to a repository

  -w -- Suppresses warning messages.
- -R - Specifies the path to the repository to be written. This is an alternative to the PEGASUS_HOME environment variable. If PEGASUS_HOME is set the repository gets written to $PEGASUS_HOME/repository. The -R flag one the command line overrides this with specified in the directive. Specify an absolute path.
- --CIMRepository=
- -I -- Specifies the path to included MOF files. If the inputmof file has include pragmas and the included files do not reside in the current directory, the -I directive must be used to specify a path to them on the compiler command line. Do this with the -I flag.

  cimmof -I~/testfiles ~/testfiles/main.mof

  The path may be relative or absolute.

Limitations of the current version

See the README in the COMPILER section of the Pegasus code tree for up-to-date information on both the current Limitations and the current TODOs.

The compiler does not handle missing include files very sanely right now. It just skips them.

TODOs

- - Parse and store references correctly
- - Test parsing and storage of instances. They probably don't work now.
- - Rationalize the error logging scheme. Things are spit randomly to sterr now.
- - Extend error detection and handling.

Alphabetic index *Hierarchy of classes*

Pegasus Operation

Starting the broker

ATTN: Under Construction

Alphabetic index *Hierarchy of classes*

Pegasus Providers

ATTN: Under Construction

Alphabetic index *Hierarchy of classes*

Pegasus Documentation

Pegasus FAQ

The following is a working list of questions and answers concerning Pegausus that do not fit into other catagories.

Alphabetic index *Hierarchy of classes*

Programming Pegasus

Pegasus was developed in C++.

- **CIM Objects in Pegasus**
- **CIM Object Table**
- **Class Definitions**

Alphabetic index *Hierarchy of classes*

CIM Object Table

tr>

CIM Objects anc Concepts and their Pegasus Implementation			
CIM Concept / Object	**Pegasus Class**	**Reference**	**Description**
CIM Classes	CIMClass	CIMClass	A CIM Class is a collection of instances, all of which support a common type; that is, a set of properties and methods. The common properties and methods are defined as features of the class
CIM Instance	CIMInstance	CIMInstance	ATT: Description
CIM DateTime	CIMDateTime	CIMDateTime	ATT: Description
CIM Qualifier Declaration	?????	ATTN	ATT: Description
CIM Qualifier	CIMQualifier	CIMQualifier	ATT: Description
CIM Property	CIMProperty	PROPERTY	A value used to characterize an instance of a class.
CIM Object Path	CIMReference	CIMReference	ATT: Description
CIM Method	CIMMethod	CIMMethod	ATT: Description
CIM Scope	CIMScope	CIMScope	ATT: Description
CIM flavor	CIMFlavor	CIMFLAVOR	Part of a qualifier spcification indicating overriding and inheritance rules.
CIM ????	Qualifier Type	???	ATT: Description
CIM Array	Array	Array	ATT: Description

CIM Value	CIMValue	CIMValue	ATT: Description
String	String	String	ATT: Description
CIM Type	CIMType	CIMType	ATT: Description

Alphabetic index *Hierarchy of classes*

CIM Objects in Pegasus

The Common Information model (CIM) consists of a number of basic objects defined in the CIM specification. This includes:

- Classes CLASS
- Instances of Classes \Rf{REFERENCE}
- Qualifiers QUALIFIER
- Properties
- Methods
- Arrays

Pegasus implements representations of these objects in C++ classes and provides methods for creation, manipulation, and deletion of the objects.

In addition, Pegasus implements a number of the key concepts built into CIM as objects including:

- Scope
- Flavor
- Object Path
- etc.

Much of the programming of the Pegasus object broker, clients, and in particular providers and service extensions in the Pegasus envrinment depends on the use of these C++ representations of CIM Objects and concepts.

Each Pegasus CIM C++ Object representation includes:

- TConstructors and destructors
- Methods for manipulation of sub-objects. Thus, the class and instance classes provide for manipulation of the property, qualifier, and method objects that are contained in these classes.
- Methods for finding sub-objects. Thus, the class and instance classes provide for finding property, qualifier, and method objects within an instnace of a CIMClass or CIMInstance. These methods are generally built around defining the name of the object to be found and having an index to that object returned.
- Methods for comparison
- Methods for cloning
- Methods to convert the object to XML
- Methods to convert the object to MOF (Planned)

There is a class for every major CIM object and concept as shown in the table in CIM Object Table

Alphabetic index *Hierarchy of classes*

Class Definitions

- **Array**

  Array Class.

- **CIMClass**

  The CIMClass class is used to represent CIM classes in Pegasus.

- **CIMConstClass**

  CIMConstClass - ATTN: define this.

- **CIMDateTime**

  The CIMDateTime class represents the CIM datetime data type as a C++ class CIMDateTime.

- **CIMQualifierDecl**

  Class CIMQualifierDecl

- **String**

  The Pegasus String C++ Class implements the CIM string type.

- **operator==**

  String operator ==.

- **operator==**

  String operator ==.

- **operator==**

  String operator ==.

- **operator!=**

  String operator ==.

- **operator+**

  overload operator + - Concatenates String objects.

- **operator<**

  overload operator < - Compares String obects.

- **operator<=**

  overload operator <= compares String objects.

- **operator>**

  Overload operator > compares String objects

- **operator>=**

  overload operator >= - Compares String objects

- **ToLower**

  Return a version of this string whose characters have been shifted to lower case

- **CompareIgnoreCase**

  Compare two strings but ignore any case differences

- **GetLine**

  Get the next line from the input file

- **CIMType**

  The CIMType Class defines the CIMType enumeration which introduces symbolic constants for the CIM data types.

- **CIMInstance**

  Class CIMInstance - The CIMInstance class represents the instance of a CIM class in Pegasus.

- **CIMValue**

  The CIMValue class represents a value of any of the CIM data types (see CIMTypeh for a list of valid CIM data types).

- **CIMProperty**

  CIMProperty Class - ATTN:

- **CIMScope**

  The CIMQualifier Scopes are as follows: NONE, Class, ASSOCIATION,INDICATION,PROPERTY, REFERENCE, METHOD, PARAMETER, ANY

- **KeyBinding**

  The KeyBinding class associates a key name, value, and type.

- **CIMReference**

  The CIMReference class represents the value of a reference.

- **CIMQualifier**

  Class CIMQualifier - This class defines the Pegasus implementation of the CIM CIMQualifier QUALIFIER

- **CIMName**

  The name class defines static methods for handling CIM names.

- **CIMMethod**

  Class CIMMethod - This class defines the operations associated with manipulation of the Pegasus implementation of the CIM CIMMethod.

- **CIMFlavor**

  CIMQualifier flavor constants

- **Char16**

  The Char16 class represents a CIM sixteen bit character (char16).

- **Stopwatch**

  Stopwatch - A class for measuring elapsed time Stopwatch is a class for measuring time intervals within the environment.

- **TimeValue**

  The TimeValue class represents time expressed in seconds plus microseconds

- **PEGASUS_NAMESPACE_END**

Alphabetic index *Hierarchy of classes*

PEGASUS_NAMESPACE_END

- Pegasus_Exception_h

Programming with Exceptions

- Exception

Class Exception

- AssertionFailureException

Class AssertionFailureException This is an Exception class tied to the definiton of an assert named PEGASUS_ASSERT.

- PEGASUS_ASSERT

define PEGASUS_ASSERT assertion statement.

- BadReference

ATTN:

- OutOfBounds

ATTN:

- AlreadyExists

ATTN:

- NullPointer

ATTN:

- IllegalName

ATTN:

- UnitializedHandle

ATTN:

- InvalidPropertyOverride

ATTN:

- InvalidMethodOverride

ATTN:

- UndeclaredQualifier

ATTN:

- BadQualifierScope

ATTN:

- BadQualifierOverride

ATTN:

- NullType

ATTN:

- AddedReferenceToClass

ATTN:

- ClassAlreadyResolved

ATTN:

- ClassNotResolved

ATTN:

- InstanceAlreadyResolved

ATTN:

- **InstantiatedAbstractClass** *ATTN:*
- **NoSuchProperty** *ATTN:*
- **TruncatedCharacter** *ATTN:*
- **ExpectedReferenceValue** *ATTN:*
- **MissingReferenceClassName** *ATTN:*
- **IllegalTypeTag** *ATTN:*
- **TypeMismatch** *ATTN:*
- **NoSuchFile** *ATTN:*
- **FailedToRemoveDirectory** *ATTN:*
- **FailedToRemoveFile** *ATTN:*
- **NoSuchDirectory** *ATTN:*
- **CannotCreateDirectory** *ATTN:*
- **NoSuchNameSpace** *ATTN:*
- **CannotOpenFile** *ATTN:*
- **NotImplemented** *ATTN:*
- **CIMException** *The CIMException defines the CIM exceptions that are formally defined in the CIM Operations over HTTP specification.*
- **PEGASUS_NAMESPACE_END**

Alphabetic index *Hierarchy of classes*

Revision History

Revision History

Alphabetic index *Hierarchy of classes*

Pegasus Documentation

Revision History

Versions	Date	Author	Comments
0.5	01/02/01	Karl	
0.6	01/04/01	Karl	Clean up the API defintions. and Add repository section.
0.7	02/20/01	Karl	Change Class names, Add definitions, Add Architecture documentation.
0.75	02/25/01	Karl	Create Frames Version and redo structure..
0.8	03/12/2001	Karl	Generate for 0.8 Release.
0.85	03/12/2001	Karl	Generate for 0.8.5 Release.
0.91	05/03/2001	Karl	Generate for 0.9 Pegasus Code

Writing Providers.

ATTN:

Alphabetic index *Hierarchy of classes*

In file ..\..\src\Pegasus\Common\CIMScope.h:

namespace CIMScope

The CIMQualifier Scopes are as follows: NONE, Class, ASSOCIATION,INDICATION,PROPERTY, REFERENCE, METHOD, PARAMETER, ANY

Documentation

The CIMQualifier Scopes are as follows: NONE, Class, ASSOCIATION,INDICATION,PROPERTY, REFERENCE, METHOD, PARAMETER, ANY

Alphabetic index *HTML hierarchy of classes* *or* *Java*

In file ..\..\src\Pegasus\Common\CIMOperations.h:

class PEGASUS_COMMON_LINKAGE CIMOperations

The CIMOperations Class.

Documentation

The CIMOperations Class. This class defines the external client operations for Pegasus These operations are based on the WBEM operations defined in the DMTF CIM CIMOperations over HTTP operations Specification. They duplicate the names, functions, and parameters of those CIMOperations. Each WBEM operation is a method in this class.

This library (operations.h) defines the external APIs for the Pegasus MSB. These are synchronous calls and match the operations defined by the DMTF in the document "Specification for CIM CIMOperations over HTTP" Version 1.0 with errata

The CIMOperations class is used by other classes such as the client, provider, and repository classes to instantiate the operations interfaces.

Inheritance:

Public Methods

virtual CIMClass	**getClass** (const String& nameSpace, const String& className, Boolean localOnly = true, Boolean includeQualifiers = true, Boolean includeClassOrigin = false, const Array<String>& propertyList = StringArray()) *The `GetClass` method returns a single CIM Class from the target Namespace where the ClassName input parameter defines the name of the class to be retrieved.*
virtual CIMInstance	**getInstance** (const String& nameSpace, const CIMReference& instanceName, Boolean localOnly = true, Boolean includeQualifiers = false, Boolean includeClassOrigin = false, const Array<String>& propertyList = StringArray()) *The <`GetInstance` method returns a single CIM Instance from the target Namespace based on the InstanceName parameter provided.*

virtual void	**deleteClass** (const String& nameSpace, const String& className) *The `DeleteClass` method deletes a single CIM Class from the target Namespace.*
virtual void	**deleteInstance** (const String& nameSpace, const CIMReference& instanceName) *The `DeleteInstance` operation deletes a single CIM Instance from the target Namespace.*
virtual void	**createClass** (const String& nameSpace, CIMClass& newClass) *The `createClass` method creates a single CIM Class in the target Namespace.*
virtual void	**createInstance** (const String& nameSpace, CIMInstance& newInstance) *The `createInstance` method creates a single CIM Instance in the target Namespace.*
virtual void	**modifyClass** (const String& nameSpace, CIMClass& modifiedClass) *The `modifyClass` method modifies an existing CIM Class in the target Namespace.*
virtual void	**modifyInstance** (const String& nameSpace, const CIMInstance& modifiedInstance) *The `modifyInstance` method is used to modify an existing CIM Instance in the target Namespace.*
virtual Array<CIMClass>	**enumerateClasses** (const String& nameSpace, const String& className = String::EMPTY, Boolean deepInheritance = false, Boolean localOnly = true, Boolean includeQualifiers = true, Boolean includeClassOrigin = false) *The `enumerateClasses` method is used to enumerate subclasses of a CIM Class in the target Namespace.*
virtual Array<String>	**enumerateClassNames** (const String& nameSpace, const String& className = String::EMPTY, Boolean deepInheritance = false) *The `enumerateClassNames` operation is used to enumerate the names of subclasses of a CIM Class in the target Namespace.*
virtual Array<CIMInstance>	**enumerateInstances** (const String& nameSpace, const String& className, Boolean deepInheritance = true, Boolean localOnly = true, Boolean includeQualifiers = false, Boolean includeClassOrigin = false, const Array<String>& propertyList = StringArray()) *The `enumerateInstances` method enumerates instances of a CIM Class in the target Namespace.*

virtual Array<CIMReference>	**enumerateInstanceNames** (const String& nameSpace, const String& className) *The `enumerateInstanceNames` operation enumerates the names (model paths) of the instances of a CIM Class in the target Namespace.*
virtual Array<CIMInstance>	**execQuery** (const String& queryLanguage, const String& query) *The `execQuery` is used to execute a query against the target Namespace.*
virtual Array<CIMInstance>	**associators** (const String& nameSpace, const CIMReference& objectName, const String& assocClass = String::EMPTY, const String& resultClass = String::EMPTY, const String& role = String::EMPTY, const String& resultRole = String::EMPTY, Boolean includeQualifiers = false, Boolean includeClassOrigin = false, const Array<String>& propertyList = StringArray()) *The `Associators` method enumerates CIM Objects (Classes or Instances) that are associated to a particular source CIM Object.*
virtual Array<CIMReference>	**associatorNames** (const String& nameSpace, const CIMReference& objectName, const String& assocClass = String::EMPTY, const String& resultClass = String::EMPTY, const String& role = String::EMPTY, const String& resultRole = String::EMPTY) *The `associatorNames` operation is used to enumerate the names of CIM Objects (Classes or Instances) that are associated to a particular source CIM Object.*
virtual Array<CIMInstance>	**references** (const String& nameSpace, const CIMReference& objectName, const String& resultClass = String::EMPTY, const String& role = String::EMPTY, Boolean includeQualifiers = false, Boolean includeClassOrigin = false, const Array<String>& propertyList= StringArray()) *The `references` operation enumerates the association objects that refer to a particular target CIM Object (Class or Instance).*
virtual Array<CIMReference>	**referenceNames** (const String& nameSpace, const CIMReference& objectName, const String& resultClass = String::EMPTY, const String& role = String::EMPTY) *The `referenceNames` operation is used to enumerate the association objects that refer to a particular target CIM Object (Class or Instance).*
virtual CIMValue	**getProperty** (const String& nameSpace, const CIMReference& instanceName, const String& propertyName) *This operation is used to retrieve a single property value from a CIM Instance in the target Namespace.*

virtual void	**setProperty** (const String& nameSpace, const CIMReference& instanceName, const String& propertyName, const CIMValue& newValue = CIMValue()) *The `setProperty` operation sets a single property value in a CIM Instance in the target Namespace.*
virtual CIMQualifierDecl	**getQualifier** (const String& nameSpace, const String& qualifierName) *The `getQualifier` operation retrieves a single CIMQualifier declaration from the target Namespace.*
virtual void	**setQualifier** (const String& nameSpace, const CIMQualifierDecl& qualifierDecl) *The `setQualifier` creates or update a single CIMQualifier declaration in the target Namespace.*
virtual void	**deleteQualifier** (const String& nameSpace, const String& qualifierName) *The `deleteQualifier` operation deletes a single CIMQualifier declaration from the target Namespace.*
virtual Array<CIMQualifierDecl>	**enumerateQualifiers** (const String& nameSpace) *The `enumerateQualifiers` operation is used to enumerate CIMQualifier declarations from the target Namespace.*
virtual CIMValue	**invokeMethod** (const String& nameSpace, const CIMReference& instanceName, const String& methodName, const Array<CIMValue>& inParameters, Array<CIMValue>& outParameters) *Any CIM Server is assumed to support extrinsic methods.*

`virtual CIMClass getClass( const String& nameSpace, const String& className, Boolean localOnly = true, Boolean includeQualifiers = true, Boolean includeClassOrigin = false, const Array<String>& propertyList = StringArray())`

The `GetClass` method returns a single CIM Class from the target Namespace where the ClassName input parameter defines the name of the class to be retrieved.

Parameters:

NameSpace - The NameSpace parameter is a string that defines the target namespace NAMESPACE
ClassName - The ClassName input parameter defines the name of the Class to be retrieved.
LocalOnly - If the `LocalOnly` input parameter is true, this specifies that only CIM Elements (properties, methods and qualifiers) overriden within the definition of the Class are returned. If false, all elements are returned. This parameter therefore effects a CIM Server-side mechanism to filter certain elements of the returned object based on whether or not they have been propagated from the parent Class (as defined by the PROPAGATED attribute).

IncludeQualifiers - If the `IncludeQualifiers` input parameter is true, this specifies that all Qualifiers for that Class (including Qualifiers on the Class and on any returned Properties, Methods or CIMMethod Parameters) MUST be included as elements in the response. If false no elements are present in the returned Class.
IncludeClassOrigin - If the `IncludeClassOrigin` input parameter is true, this specifies that the CLASSORIGIN attribute MUST be present on all appropriate elements in the returned Class. If false, no CLASSORIGIN attributes are present in the returned Class.
PropertyList - If the `PropertyList` input parameter is not NULL, the members of the array define one or more CIMProperty names. The returned Class MUST NOT include elements for any Properties missing from this list. Note that if LocalOnly is specified as true this acts as an additional filter on the set of Properties returned (for example, if CIMProperty A is included in the PropertyList but LocalOnly is set to true and A is not local to the requested Class, then it will not be included in the response). If the PropertyList input parameter is an empty array this signifies that no Properties are included in the response. If the PropertyList input parameter is NULL this specifies that all Properties (subject to the conditions expressed by the other parameters) are included in the response. If the `PropertyList` contains duplicate elements, the Server MUST ignore the duplicates but otherwise process the request normally. If the PropertyList contains elements which are invalid CIMProperty names for the target Class, the Server MUST ignore such entries but otherwise process the request normally.

Returns:

If successful, the return value is a single CIM Class. If unsuccessful, one of the following status codes MUST be returned by this method, where the first applicable error in the list (starting with the first element of the list, and working down) is the error returned. Any additional method-specific interpretation of the error in is given in parentheses.

- CIM_ERR_ACCESS_DENIED
- CIM_ERR_INVALID_NAMESPACE
- CIM_ERR_INVALID_PARAMETER (including missing, duplicate,unrecognized or otherwise incorrect parameters)
- CIM_ERR_NOT_FOUND (the request CIM Class does not exist in the specified namespace)
- CIM_ERR_FAILED (some other unspecified error occurred)

virtual CIMInstance getInstance(const String& nameSpace, const CIMReference& instanceName, Boolean localOnly = true, Boolean includeQualifiers = false, Boolean includeClassOrigin = false, const Array<String>& propertyList = StringArray())

The <`GetInstance` method returns a single CIM Instance from the target Namespace based on the InstanceName parameter provided.

Parameters:

NameSpace - The NameSpace parameter is a string that defines the target namespace

NAMESPACE

InstanceName - The InstanceName input parameter defines the name of the Instance to be retrieved.

LocalOnly - If the `LocalOnly` input parameter is true, this specifies that only elements (properties and qualifiers) overriden within the definition of the Instance are returned. If false, all elements are returned. This parameter therefore effects a CIM Server-side mechanism to filter certain elements of the returned object based on whether or not they have been propagated from the parent Class (as defined by the PROPAGATED attribute).

IncludeQualifiersIf - the `IncludeQualifiers` input parameter is true, this specifies that all Qualifiers for that Instance (including Qualifiers on the Instance and on any returned Properties) MUST be included as elements in the response. If false no elements are present in the returned Instance.

IncludeClassOrigin - If the `IncludeClassOrigin` input parameter is true, this specifies that the CLASSORIGIN attribute MUST be present on all appropriate elements in the returned Instance. If false, no CLASSORIGIN attributes are present in the returned instance.

PropertyList - If the `PropertyList` input parameter is not NULL, the members of the array define one or more CIMProperty names. The returned Instance MUST NOT include elements for any Properties missing from this list. Note that if LocalOnly is specified as true this acts as an additional filter on the set of Properties returned (for example, if CIMProperty A is included in the PropertyList but LocalOnly is set to true and A is not local to the requested Instance, then it will not be included in the response). If the PropertyList input parameter is an empty array this signifies that no Properties are included in the response. If the PropertyList input parameter is NULL this specifies that all Properties (subject to the conditions expressed by the other parameters) are included in the response. If the `PropertyList` contains duplicate elements, the Server MUST ignore the duplicates but otherwise process the request normally. If the PropertyList contains elements which are invalid CIMProperty names for the target Instance, the Server MUST ignore such entries but otherwise process the request normally.

Returns:

If successful, the return value is a single CIM Instance. If unsuccessful, one of the following status codes MUST be returned by this method, where the first applicable error in the list (starting with the first element of the list, and working down) is the error returned. Any additional method-specific interpretation of the error in is given in parentheses.

- CIM_ERR_ACCESS_DENIED
- CIM_ERR_INVALID_NAMESPACE
- CIM_ERR_INVALID_PARAMETER (including missing, duplicate, unrecognized or otherwise incorrect parameters)
- CIM_ERR_INVALID_CLASS (the CIM Class does not exist in the specified namespace)
- CIM_ERR_NOT_FOUND (the CIM Class does exist, but the requested CIM Instance does not exist in the specified namespace)
- CIM_ERR_FAILED (some other unspecified error occurred)

virtual void deleteClass(const String& nameSpace, const String& className)

The `DeleteClass` method deletes a single CIM Class from the target Namespace.

Parameters:

NameSpace - The NameSpace parameter is a string that defines the target namespace NAMESPACE
ClassName - The ClassName input parameter defines the name of the Class to be deleted.

Returns:

If successful, the specified Class (including any subclasses and any instances) MUST have been removed by the CIM Server. The operation MUST fail if any one of these objects cannot be deleted. If unsuccessful, one of the following status codes MUST be returned by this method, where the first applicable error in the list (starting with the first element of the list, and working down) is the error returned. Any additional method-specific interpretation of the error in is given in parentheses.

- CIM_ERR_ACCESS_DENIED
- CIM_ERR_NOT_SUPPORTED
- CIM_ERR_INVALID_NAMESPACE
- CIM_ERR_INVALID_PARAMETER (including missing, duplicate, unrecognized or otherwise incorrect parameters)
- CIM_ERR_NOT_FOUND (the CIM Class to be deleted does not exist)
- CIM_ERR_CLASS_HAS_CHILDREN (the CIM Class has one or more subclasses which cannot be deleted)
- CIM_ERR_CLASS_HAS_INSTANCES (the CIM Class has one or more instances which cannot be deleted)
- CIM_ERR_FAILED (some other unspecified error occurred)

virtual void deleteInstance(const String& nameSpace, const CIMReference& instanceName)

The `DeleteInstance` operation deletes a single CIM Instance from the target Namespace.

Parameters:

NameSpace - The NameSpace parameter is a string that defines the target namespace NAMESPACE
InstanceName - The InstanceName input parameter defines the name (model path) of the Instance to be deleted.

Returns:

If successful, the specified Instance MUST have been removed by the CIM Server. If unsuccessful, one of the following status codes MUST be returned by this method, where the first applicable error in the list (starting with the first element of the list, and working

down) is the error returned. Any additional method-specific interpretation of the error in is given in parentheses.

- CIM_ERR_ACCESS_DENIED
- CIM_ERR_NOT_SUPPORTED
- CIM_ERR_INVALID_NAMESPACE
- CIM_ERR_INVALID_PARAMETER (including missing, duplicate, unrecognized or otherwise incorrect parameters)
- CIM_ERR_INVALID_CLASS (the CIM Class does not exist in the specified namespace)
- CIM_ERR_NOT_FOUND (the CIM Class does exist, but the requested CIM Instance does not exist in the specified namespace)
- CIM_ERR_FAILED (some other unspecified error occurred)

`virtual void createClass( const String& nameSpace, CIMClass& newClass)`

The `createClass` method creates a single CIM Class in the target Namespace. The Class MUST NOT already exist. The NewClass input parameter defines the new Class. The proposed definition MUST be a correct Class definition according to the CIM specification.

In processing the creation of the new Class, the following rules MUST be conformed to by the CIM Server:

Any CLASSORIGIN and PROPAGATED attributes in the NewClass MUST be ignored by the Server. If the new Class has no Superclass, the NewClass parameter defines a new base Class. The Server MUST ensure that all Properties and Methods of the new Class have a CLASSORIGIN attribute whose value is the name of the new Class. If the new Class has a Superclass, the NewClass parameter defines a new Subclass of that Superclass. The Superclass MUST exist. The Server MUST ensure that:

- Any Properties, Methods or Qualifiers in the Subclass not defined in the Superclass are created as new elements of the Subclass. In particular the Server MUST set the CLASSORIGIN attribute on the new Properties and Methods to the name of the Subclass, and ensure that all other Properties and Methods preserve their CLASSORIGIN attribute value from that defined in the Superclass

  If a CIMProperty is defined in the Superclass and in the Subclass, the value assigned to that property in the Subclass (including NULL) becomes the default value of the property for the Subclass. If a CIMProperty or CIMMethod of the Superclass is not specified in the Subclass, then that CIMProperty or CIMMethod is inherited without modification by the Subclass
- Any Qualifiers defined in the Superclass with a TOSUBCLASS attribute value of true MUST appear in the resulting Subclass. Qualifiers in the Superclass with a TOSUBCLASS attribute value of false MUST NOT be propagated to the Subclass . Any CIMQualifier propagated from the Superclass cannot be modified in the Subclass if the OVERRIDABLE

attribute of that CIMQualifier was set to false in the Superclass. It is a Client error to specify such a CIMQualifier in the NewClass with a different definition to that in the Superclass (where definition encompasses the name, type and flavor attribute settings of the element, and the value of the CIMQualifier).

Parameters:

NameSpace - The NameSpace parameter is a string that defines the target namespace NAMESPACE @parm NewClass The NewClass input parameter defines the new Class.

Returns:

If successful, the specified Class MUST have been created by the CIM Server. If unsuccessful, one of the following status codes MUST be returned by this method, where the first applicable error in the list (starting with the first element of the list, and working down) is the error returned. Any additional method-specific interpretation of the error in is given in parentheses.

- CIM_ERR_ACCESS_DENIED
- CIM_ERR_NOT_SUPPORTED
- CIM_ERR_INVALID_NAMESPACE
- CIM_ERR_INVALID_PARAMETER (including missing, duplicate, unrecognized or otherwise incorrect parameters)
- CIM_ERR_ALREADY_EXISTS (the CIM Class already exists)
- CIM_ERR_INVALID_SUPERCLASS (the putative CIM Class declares a non-existent superclass)
- CIM_ERR_FAILED (some other unspecified error occurred)

`virtual void createInstance( const String& nameSpace, CIMInstance& newInstance)`

The `createInstance` method creates a single CIM Instance in the target Namespace. The Instance MUST NOT already exist.

In processing the creation of the new Instance, the following rules MUST be conformed to by the CIM Server:

Any CLASSORIGIN and PROPAGATED attributes in the NewInstance MUST be ignored by the Server.

The Server MUST ensure that:

- Any Qualifiers in the Instance not defined in the Class are created as new elements of the Instance.
- All Properties of the Instance preserve their CLASSORIGIN attribute value from that defined in the Class.
- If a CIMProperty is specified in the ModifiedInstance parameter, the value assigned to that property in the Instance (including NULL) becomes the value of the property for the Instance. Note that it is a Client error to specify a CIMProperty that does not belong to the

Class.

- If a CIMProperty of the Class is not specified in the Instance, then that CIMProperty is inherited without modification by the Instance.
- Any Qualifiers defined in the Class with a TOINSTANCE attribute value of true appear in the Instance. Qualifiers in the Class with a TOINSTANCE attribute value of false MUST NOT be propagated to the Instance.
- Any CIMQualifier propagated from the Class cannot be modified in the Instance if the OVERRIDABLE attribute of that CIMQualifier was set to false in the Class. It is a Client error to specify such a CIMQualifier in the NewInstance with a different definition to that in the Class (where definition encompasses the name, type and flavor attribute settings of the element, and the value of the CIMQualifier).

Parameters:

NameSpace - The NameSpace parameter is a string that defines the target namespace NAMESPACE
newInstance - The `newInstance` input parameter defines the new Instance. The proposed definition MUST be a correct Instance definition for the underlying CIM Class according to the CIM specification.

Returns:

If successful, the return value defines the object path of the new CIM Instance relative to the target Namespace (i.e. the Model Path as defined by the CIM specification), created by the CIM Server. It is returned in case one or more of the new keys of the Instance are allocated dynamically during the creation process rather than specified in the request. If unsuccessful, one of the following status codes MUST be returned by this method, where the first applicable error in the list (starting with the first element of the list, and working down) is the error returned. Any additional method-specific interpretation of the error in is given in parentheses.

- CIM_ERR_ACCESS_DENIED
- CIM_ERR_NOT_SUPPORTED
- CIM_ERR_INVALID_NAMESPACE
- CIM_ERR_INVALID_PARAMETER (including missing, duplicate, unrecognized or otherwise incorrect parameters)
- CIM_ERR_INVALID_CLASS (the CIM Class of which this is to be a new Instance does not exist)
- CIM_ERR_ALREADY_EXISTS (the CIM Instance already exists)
- CIM_ERR_FAILED (some other unspecified error occurred)

`virtual void modifyClass( const String& nameSpace, CIMClass& modifiedClass)`

The `modifyClass` method modifies an existing CIM Class in the target Namespace. The Class MUST already exist. The `ModifiedClass` input parameter defines the set of changes (which MUST be correct amendments to the CIM Class as defined by the CIM Specification) to be made

to the current class definition.

In processing the modifcation of the Class, the following rules MUST be conformed to by the CIM Server.

- Any `CLASSORIGIN` and `PROPAGATED` attributes in the `ModifiedClass` MUST be ignored by the Server.
- If the modified Class has no Superclass, the`ModifiedClass` parameter defines modifications to a base Class. The Server MUST ensure that:
  - All Properties and Methods of the modified Class have a `CLASSORIGIN` attribute whose value is the name of this Class.
  - Any Properties, Methods or Qualifiers in the existing Class definition which do not appear in the ModifiedClass parameter are removed from the resulting modified Class.
- If the modified Class has a Superclass,the ModifiedClass parameter defines modifications to a Subclass of that Superclass. The Superclass MUST exist, and the Client MUST NOT change the name of the Superclass in the modified Subclass. The Server MUST ensure that:
  - Any Properties, Methods or Qualifiers in the Subclass not defined in the Superclass are created as elements of the Subclass. In particular the Server MUST set the CLASSORIGIN attribute on the new Properties and Methods to the name of the Subclass, and MUST ensure that all other Properties and Methods preserve their CLASSORIGIN attribute value from that defined in the Superclass.
  - Any CIMProperty, CIMMethod or CIMQualifier previously defined in the Subclass but not defined in the Superclass, and which is not present in the ModifiedClass parameter, is removed from the Subclass.
  - If a CIMProperty is specified in the ModifiedClass parameter, the value assigned to that property therein (including NULL) becomes the default value of the property for the Subclass.
  - If a CIMProperty or CIMMethod of the Superclass is not specified in the Subclass, then that CIMProperty or CIMMethod is inherited without modification by the Subclass (so that any previous changes to such an Element in the Subclass are lost).
  - If a CIMQualifier in the Superclass is not specified in the Subclass, and the CIMQualifier is defined in the Superclass with a TOSUBCLASS attribute value of true, then the CIMQualifier MUST still be present in the resulting modified Subclass (it is not possible to remove a propagated CIMQualifier from a Subclass).
  - Any CIMQualifier propagated from the Superclass cannot be modified in the Subclass if the OVERRIDABLE attribute of that CIMQualifier was set to false in the Superclass. It is a Client error to specify such a CIMQualifier in the ModifiedClass with a different definition to that in the Superclass (where definition encompasses the name, type and flavor attribute settings of the <QUALIFIER> element, and the value of the CIMQualifier).
  - Any Qualifiers defined in the Superclass with a TOSUBCLASS attribute value of false MUST NOT be propagated to the Subclass.

Parameters:

NameSpace - The NameSpace parameter is a string that defines the target namespace NAMESPACE
ModifiedClass - The ModifiedClass input parameter defines the set of changes (which MUST be correct amendments to the CIM Class as defined by the CIM Specification) to be made to the current class definition.

Returns:

If successful, the specified Class MUST have been updated by the CIM Server. The request to modify the Class MUST fail if the Server cannot update any existing Subclasses or Instances of that Class in a consistent manner.
If unsuccessful, one of the following status codes MUST be returned by this method, where the first applicable error in the list (starting with the first element of the list, and working down) is the error returned. Any additional method-specific interpretation of the error in is given in parentheses.

- CIM_ERR_ACCESS_DENIED
- CIM_ERR_NOT_SUPPORTED
- CIM_ERR_INVALID_NAMESPACE
- CIM_ERR_INVALID_PARAMETER (including missing, duplicate, unrecognized or otherwise incorrect parameters)
- CIM_ERR_NOT_FOUND (the CIM Class does not exist)
- CIM_ERR_INVALID_SUPERCLASS (the putative CIM Class declares a non-existent or incorrect superclass)
- CIM_ERR_CLASS_HAS_CHILDREN (the modification could not be performed because it was not possible to update the subclasses of the Class in a consistent fashion)
- CIM_ERR_CLASS_HAS_INSTANCES (the modification could not be performed because it was not possible to update the instances of the Class in a consistent fashion)
- CIM_ERR_FAILED (some other unspecified error occurred)

virtual void modifyInstance(const String& nameSpace, const CIMInstance& modifiedInstance)

The modifyInstance method is used to modify an existing CIM Instance in the target Namespace.

Parameters:

NameSpace - The NameSpace parameter is a string that defines the target namespace NAMESPACE
ModifiedInstance - The ModifiedInstance input parameter identifies the name of the Instance to be modified, and defines the set of changes (which MUST be correct amendments to the Instance as defined by the CIM Specification) to be made to the current Instance definition. In processing the modifcation of the Instance, the following rules MUST be conformed to by the CIM Server:

- Any CLASSORIGIN and PROPAGATED attributes in the ModifiedInstance MUST be

ignored by the Server.

- The Class MUST exist, and the Client MUST NOT change the name of the Class in the modified Instance. The Server MUST ensure that:
  - Any Qualifiers in the Instance not defined in the Class are created as new elements of the Instance.
  - All Properties of the Instance preserve their CLASSORIGIN attribute value from that defined in the Class.
  - Any CIMQualifier previously defined in the Instance but not defined in the Class, and which is not present in the ModifiedInstance parameter, is removed from the Instance.
  - If a CIMProperty is specified in the ModifiedInstance parameter, the value assigned to that property therein (including NULL) becomes the value of the property for the Instance. Note that it is a Client error to specify a CIMProperty that does not belong to the Class.
  - If a CIMProperty of the Class is not specified in the Instance, then that CIMProperty is inherited without modification by the Instance (so that any previous changes to that CIMProperty in the Instance are lost).
  - Any Qualifiers defined in the Class with a TOINSTANCE attribute value of true appear in the Instance (it is not possible remove a propagated CIMQualifier from an Instance. Qualifiers in the Class with a TOINSTANCE attribute value of false MUST NOT be propagated to the Instance.
  - Any CIMQualifier propagated from the Class cannot be modified by the Server if the OVERRIDABLE attribute of that CIMQualifier was set to false in the Class. It is a Client error to specify such a CIMQualifier in the ModifiedInstance with a different definition to that in the Class (where definition encompasses the name, type and flavor attribute settings of the <QUALIFIER> element, and the value of the CIMQualifier).
  - Any CIMQualifier propagated from the Class cannot be modified in the Instance if the OVERRIDABLE attribute of that CIMQualifier was set to false in the Class. It is a Client error to specify such a CIMQualifier in the ModifiedInstance with a different definition to that in the Class (where definition encompasses the name, type and flavor attribute settings of the <QUALIFIER> element, and the value of the CIMQualifier).

Returns:

If successful, the specified Instance MUST have been updated by the CIM Server. If unsuccessful, one of the following status codes MUST be returned by this method, where the first applicable error in the list (starting with the first element of the list, and working down) is the error returned. Any additional method-specific interpretation of the error in is given in parentheses

- CIM_ERR_ACCESS_DENIED
- CIM_ERR_NOT_SUPPORTED
- CIM_ERR_INVALID_NAMESPACE
- CIM_ERR_INVALID_PARAMETER (including missing, duplicate, unrecognized or otherwise incorrect parameters)
- CIM_ERR_INVALID_CLASS (the CIM Class of which this is to be a new Instance does not exist)
- CIM_ERR_NOT_FOUND (the CIM Instance does not exist)
- CIM_ERR_FAILED (some other unspecified error occurred)

virtual Array<CIMClass> enumerateClasses(const String& nameSpace, const String& className = String::EMPTY, Boolean deepInheritance = false, Boolean localOnly = true, Boolean includeQualifiers = true, Boolean includeClassOrigin = false)

The enumerateClasses method is used to enumerate subclasses of a CIM Class in the target Namespace.

Parameters:

NameSpace - The NameSpace parameter is a string that defines the target namespace NAMESPACE
className - The ClassName input parameter defines the Class that is the basis for the enumeration.
DeepInheritance - If the DeepInheritance input parameter is true, this specifies that all subclasses of the specified Class should be returned (if the ClassName input parameter is absent, this implies that all Classes in the target Namespace should be returned). If false, only immediate child subclasses are returned (if the ClassName input parameter is NULL, this implies that all base Classes in the target Namespace should be returned).
LocalOnly - If the LocalOnly input parameter is true, it specifies that, for each returned Class, only elements (properties, methods and qualifiers) overriden within the definition of that Class are included. If false, all elements are returned. This parameter therefore effects a CIM Server-side mechanism to filter certain elements of the returned object based on whether or not they have been propagated from the parent Class (as defined by the PROPAGATED attribute).
IncludeQualifiers - If the IncludeQualifiers input parameter is true, this specifies that all Qualifiers for each Class (including Qualifiers on the Class and on any returned Properties, Methods or CIMMethod Parameters) MUST be included as <QUALIFIER> elements in the response. If false no <QUALIFIER> elements are present in each returned Class.
IncludeClassOrigin - If the IncludeClassOrigin input parameter is true, this specifies that the CLASSORIGIN attribute MUST be present on all appropriate elements in each returned Class. If false, no CLASSORIGIN attributes are present in each returned Class.

Returns:

If successful, the method returns zero or more Classes that meet the required criteria. If unsuccessful, one of the following status codes MUST be returned by this method, where the first applicable error in the list (starting with the first element of the list, and working

down) is the error returned. Any additional method-specific interpretation of the error in is given in parentheses.

- CIM_ERR_ACCESS_DENIED
- CIM_ERR_NOT_SUPPORTED
- CIM_ERR_INVALID_NAMESPACE
- CIM_ERR_INVALID_PARAMETER (including missing, duplicate, unrecognized or otherwise incorrect parameters)
- CIM_ERR_INVALID_CLASS (the CIM Class that is the basis for this enumeration does not exist)
- CIM_ERR_FAILED (some other unspecified error occurred)

virtual Array<String> enumerateClassNames(const String& nameSpace, const String& className = String::EMPTY, Boolean deepInheritance = false)

The enumerateClassNames operation is used to enumerate the names of subclasses of a CIM Class in the target Namespace.

Parameters:

NameSpace - The NameSpace parameter is a string that defines the target namespace NAMESPACE
className - The ClassName input parameter defines the Class that is the basis for the enumeration.
DeepInheritance - If the DeepInheritance input parameter is true, this specifies that the names of all subclasses of the specified Class should be returned (if the ClassName input parameter is absent, this implies that the names of all Classes in the target Namespace should be returned). If false, only the names of immediate child subclasses are returned (if the ClassName input parameter is NULL, this implies that the names of all base Classes in the target Namespace should be returned).

Returns:

If successful, the method returns zero or more names of Classes that meet the requested criteria. If unsuccessful, one of the following status codes MUST be returned by this method, where the first applicable error in the list (starting with the first element of the list, and working down) is the error returned. Any additional method-specific interpretation of the error in is given in parentheses.

- CIM_ERR_ACCESS_DENIED
- CIM_ERR_NOT_SUPPORTED
- CIM_ERR_INVALID_NAMESPACE
- CIM_ERR_INVALID_PARAMETER (including missing, duplicate, unrecognized or otherwise incorrect parameters)
- CIM_ERR_INVALID_CLASS (the CIM Class that is the basis for this enumeration does not exist)
- CIM_ERR_FAILED (some other unspecified error occurred)

virtual Array<CIMInstance> enumerateInstances(const String& nameSpace, const String& className, Boolean deepInheritance = true, Boolean localOnly = true, Boolean includeQualifiers = false, Boolean includeClassOrigin = false, const Array<String>& propertyList = StringArray())

The enumerateInstances method enumerates instances of a CIM Class in the target Namespace.

Parameters:

NameSpace - The NameSpace parameter is a string that defines the target namespace NAMESPACE

ClassName - The ClassName input parameter defines the Class that is the basis for the enumeration.

LocalOnly - If the LocalOnly input parameter is true, this specifies that, for each returned Instance, only elements (properties and qualifiers) overriden within the definition of that Instance are included. If false, all elements are returned. This parameter therefore effects a CIM Server-side mechanism to filter certain elements of the returned object based on whether or not they have been propagated from the parent Class (as defined by the PROPAGATED attribute). Only elements (properties, methods and qualifiers) defined or overridden within the class are included in the response. Propagated properties are not included because their values are based on another class' information. If not specified, all elements of the class' definition are returned. Note: When instances are returned, the InstanceName must include all keys, including propagated keys. Therefore, these attributes are included in the name part of the method response, but not in the value information.

DeepInheritance - If the DeepInheritance input parameter is true, this specifies that, for each returned Instance of the Class, all properties of the Instance MUST be present (subject to constraints imposed by the other parameters), including any which were added by subclassing the specified Class. If false, each returned Instance includes only properties defined for the specified Class. The Enumerate Instances operation returns the same number of instances regardless of whether or not the DeepInheritance flag is set. The DeepInheritance flag is only used to determine whether or not the subclass property values should be returned.

IncludeQualifiersIf - the IncludeQualifiers input parameter is true, this specifies that all Qualifiers for each Instance (including Qualifiers on the Instance and on any returned Properties) MUST be included as <QUALIFIER> elements in the response. If false no <QUALIFIER> elements are present in each returned Instance.

IncludeClassOrigin - If the IncludeClassOrigin input parameter is true, this specifies that the CLASSORIGIN attribute MUST be present on all appropriate elements in each returned Instance. If false, no CLASSORIGIN attributes are present in each returned Instance.

PropertyList - If the PropertyList input parameter is not NULL, the members of the array define one or more CIMProperty names. Each returned Instance MUST NOT include elements for any Properties missing from this list. Note that if LocalOnly is specified as true (or DeepInheritance is specified as false) this acts as an additional filter on the set of Properties returned (for example, if CIMProperty A is included in the PropertyList but LocalOnly is set to true and A is not local to a returned Instance, then it will not be included in that Instance). If the PropertyList input parameter is an empty array this signifies that no Properties are included in each returned Instance. If the PropertyList input parameter is NULL this specifies that all Properties (subject to the conditions

expressed by the other parameters) are included in each returned Instance. If the PropertyList contains duplicate elements, the Server MUST ignore the duplicates but otherwise process the request normally. If the PropertyList contains elements which are invalid CIMProperty names for any target Instance, the Server MUST ignore such entries but otherwise process the request normally.

Returns:

If successful, the method returns zero or more named Instances that meet the required criteria. If unsuccessful, one of the following status codes MUST be returned by this method, where the first applicable error in the list (starting with the first element of the list, and working down) is the error returned. Any additional method-specific interpretation of the error in is given in parentheses.

- CIM_ERR_ACCESS_DENIED
- CIM_ERR_NOT_SUPPORTED
- CIM_ERR_INVALID_NAMESPACE
- CIM_ERR_INVALID_PARAMETER (including missing, duplicate, unrecognized or otherwise incorrect parameters)
- CIM_ERR_INVALID_CLASS (the CIM Class that is the basis for this enumeration does not exist)
- CIM_ERR_FAILED (some other unspecified erroroccurred)

virtual Array<CIMReference> enumerateInstanceNames(const String& nameSpace, const String& className)

The enumerateInstanceNames operation enumerates the names (model paths) of the instances of a CIM Class in the target Namespace.

Parameters:

NameSpace - The NameSpace parameter is a string that defines the target namespace NAMESPACE
ClassName - The ClassName input parameter defines the Class that is the basis for the enumeration.

Returns:

If successful, the method returns zero or more names of Instances (model paths) that meet the requsted criteria. If unsuccessful, one of the following status codes MUST be returned by this method, where the first applicable error in the list (starting with the first element of the list, and working down) is the error returned. Any additional method-specific interpretation of the error in is given in parentheses.

- CIM_ERR_ACCESS_DENIED
- CIM_ERR_NOT_SUPPORTED
- CIM_ERR_INVALID_NAMESPACE
- CIM_ERR_INVALID_PARAMETER (including missing, duplicate, unrecognized or otherwise incorrect parameters)
- CIM_ERR_INVALID_CLASS (the CIM Class that is the basis for this enumeration does not exist)

- CIM_ERR_FAILED (some other unspecified error occurred)

virtual Array<CIMInstance> execQuery(const String& queryLanguage, const String& query)

The execQuery is used to execute a query against the target Namespace.

Parameters:

NameSpace - The NameSpace parameter is a string that defines the target namespace NAMESPACE
QueryLanguage - The QueryLanguage input parameter defines the query language in which the Query parameter is expressed.
The - Query input parameter defines the query to be executed.

Returns:

If successful, the method returns zero or more CIM Classes or Instances that correspond to the results set of the query. If unsuccessful, one of the following status codes MUST be returned by this method, where the first applicable error in the list (starting with the first element of the list, and working down) is the error returned. Any additional method-specific interpretation of the error in is given in parentheses.

- CIM_ERR_ACCESS_DENIED
- CIM_ERR_NOT_SUPPORTED
- CIM_ERR_INVALID_NAMESPACE
- CIM_ERR_INVALID_PARAMETER (including missing, duplicate, unrecognized or otherwise incorrect parameters)
- CIM_ERR_QUERY_LANGUAGE_NOT_SUPPORTED (the requested query language is not recognized)
- CIM_ERR_INVALID_QUERY (the query is not a valid query in the specified query language)
- CIM_ERR_FAILED (some other unspecified error ccurred)

virtual Array<CIMInstance> associators(const String& nameSpace, const CIMReference& objectName, const String& assocClass = String::EMPTY, const String& resultClass = String::EMPTY, const String& role = String::EMPTY, const String& resultRole = String::EMPTY, Boolean includeQualifiers = false, Boolean includeClassOrigin = false, const Array<String>& propertyList = StringArray())

The Associators method enumerates CIM Objects (Classes or Instances) that are associated to a particular source CIM Object.

Parameters:

NameSpace - The NameSpace parameter is a string that defines the target namespace NAMESPACE @ObjectName The ObjectName input parameter defines the source CIM Object whose associated Objects are to be returned. This may be either a Class name or Instance name (model path).
AssocClass - The AssocClass input parameter, if not NULL, MUST be a valid CIM 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.
ResutlClass - The ResultClass input parameter, if not NULL, MUST be a valid CIM 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) or be this Class (or one of its subclasses).
Role - The Role input parameter, if not NULL, MUST be a valid CIMProperty 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 CIMProperty in the Association Class that refers to the source Object MUST match the value of this parameter).
ResultRole - The ResultRole input parameter, if not NULL, MUST be a valid CIMProperty 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 CIMProperty in the Association Class that refers to the returned Object MUST match the value of this parameter).
IncludeQualifiers - If the IncludeQualifiers input parameter is true, this specifies that all Qualifiers for each Object (including Qualifiers on the Object and on any returned Properties) MUST be included as elements in the response. If false no elements are present in each returned Object.
IncludeClassOrigin - If the IncludeClassOrigin input parameter is true, this specifies that the CLASSORIGIN attribute MUST be present on all appropriate elements in each returned Object. If false, no CLASSORIGIN attributes are present in each returned Object. @parm PropertyList If the PropertyList input parameter is not NULL, the members of the array define one or more CIMProperty names. Each returned Object MUST NOT include elements for any Properties missing from this list. Note that if LocalOnly is specified as true (or DeepInheritance is specified as false) this acts as an additional filter on the set of Properties returned (for example, if CIMProperty A is included in the PropertyList but LocalOnly is set to true and A is not local to a returned Instance, then it will not be included in that Instance). If the PropertyList input parameter is an empty array this signifies that no Properties are included in each returned Object. If the PropertyList input parameter is NULL this specifies that all Properties (subject to the conditions expressed by the other parameters) are included in each returned Object. If the PropertyList contains duplicate elements, the Server MUST ignore the duplicates but otherwise process the request normally. If the PropertyList contains elements which are invalid CIMProperty names for any target Object, the Server MUST ignore such entries but otherwise process the request normally. Clients SHOULD NOT explicitly specify properties in the PropertyList parameter unless they have specified a non-NULL value for the ResultClass parameter.

Returns:

If successful, the method returns zero or more CIM Classes or Instances meeting the requested criteria. Since it is possible for CIM Objects from different hosts or namespaces to be associated, each returned Object includes location information. If unsuccessful, one of the following status codes MUST be returned by this method, where the first applicable error in the list (starting with the first element of the list, and working down) is the error returned. Any additional method-specific interpretation of the error in is given in parentheses.

- CIM_ERR_ACCESS_DENIED

- CIM_ERR_NOT_SUPPORTED
- CIM_ERR_INVALID_NAMESPACE
- CIM_ERR_INVALID_PARAMETER (including missing,duplicate, unrecognized or otherwise incorrect parameters)
- CIM_ERR_FAILED (some other unspecified error occurred)

virtual Array<CIMReference> associatorNames(const String& nameSpace, const CIMReference& objectName, const String& assocClass = String::EMPTY, const String& resultClass = String::EMPTY, const String& role = String::EMPTY, const String& resultRole = String::EMPTY)

The associatorNames operation is used to enumerate the names of CIM Objects (Classes or Instances) that are associated to a particular source CIM Object.

Parameters:

NameSpace - The NameSpace parameter is a string that defines the target namespace NAMESPACE
objectName - - The ObjectName input parameter defines the source CIM Object whose associated names are to be returned. This is either a Class name or Instance name (model path).
AssocClass - - The AssocClass input parameter, if not NULL, MUST be a valid CIM Association Class name. It acts as a filter on the returned set of names by mandating that each returned name identifies an Object that MUST be associated to the source Object via an Instance of this Class or one of its subclasses.
ResultClass - - The ResultClass input parameter, if not NULL, MUST be a valid CIM Class name. It acts as a filter on the returned set of names by mandating that each returned name identifies an Object that MUST be either an Instance of this Class (or one of its subclasses) or be this Class (or one of its subclasses).
Role - - The Role input parameter, if not NULL, MUST be a valid CIMProperty name. It acts as a filter on the returned set of names by mandating that each returned name identifies an Object that 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 CIMProperty in the Association Class that refers to the source Object MUST match the value of this parameter).
ResultRole - - The ResultRole input parameter, if not NULL, MUST be a valid CIMProperty name. It acts as a filter on the returned set of names by mandating that each returned name identifies an Object that MUST be associated to the source Object via an Association in which the named returned Object plays the specified role (i.e. the name of the CIMProperty in the Association Class that refers to the returned Object MUST match the value of this parameter).

Returns:

If successful, the method returns zero or more full CIM Class paths or Instance paths of Objects meeting the requested criteria. Since it is possible for CIM Objects from different hosts or namespaces to be associated, each returned path is an absolute path that includes host and namespace information. If unsuccessful, one of the following status codes MUST be returned by this method, where the first applicable error in the list (starting with the first element of the list, and working down) is the error returned. Any additional method-specific interpretation of the error in is given in parentheses.

- CIM_ERR_ACCESS_DENIED
- CIM_ERR_NOT_SUPPORTED
- CIM_ERR_INVALID_NAMESPACE;
- CIM_ERR_INVALID_PARAMETER (including missing, duplicate, unrecognized or otherwise incorrect parameters)
- CIM_ERR_FAILED (some other unspecified error occurred)

virtual Array<CIMInstance> references(const String& nameSpace, const CIMReference& objectName, const String& resultClass = String::EMPTY, const String& role = String::EMPTY, Boolean includeQualifiers = false, Boolean includeClassOrigin = false, const Array<String>& propertyList= StringArray())

The references operation enumerates the association objects that refer to a particular target CIM Object (Class or Instance).

Parameters:

The - NameSpace parameter is a string that defines the target namespace NAMESPACE
The - ObjectName input parameter defines the target CIM Object whose referring Objects are to be returned. This is either a Class name or Instance name (model path).
The - ResultClass input parameter, if not NULL, MUST be a valid CIM Class name. It acts as a filter on the returned set of Objects by mandating that each returned Object MUST be an Instance of this Class (or one of its subclasses), or this Class (or one of its subclasses).
The - Role input parameter, if not NULL, MUST be a valid CIMProperty name. It acts as a filter on the returned set of Objects by mandating that each returned Objects MUST refer to the target Object via a CIMProperty whose name matches the value of this parameter.
If - the IncludeQualifiers input parameter is true, this specifies that all Qualifiers for each Object (including Qualifiers on the Object and on any returned Properties) MUST be included as elements in the response. If false no elements are present in each returned Object.
IncludeClassOrigin - If the IncludeClassOrigin input parameter is true, this specifies that the CLASSORIGIN attribute MUST be present on all appropriate elements in each returned Object. If false, no CLASSORIGIN attributes are present in each returned Object.
PropertyList - If the PropertyList input parameter is not NULL, the members of the array define one or more CIMProperty names. Each returned Object MUST NOT include elements for any Properties missing from this list. Note that if LocalOnly is specified as true (or DeepInheritance is specified as false) this acts as an additional filter on the set of Properties returned (for example, if CIMProperty A is included in the PropertyList but LocalOnly is set to true and A is not local to a returned Instance, then it will not be included in that Instance). If the PropertyList input parameter is an empty array this signifies that no Properties are included in each returned Object. If the PropertyList input parameter is NULL this specifies that all Properties (subject to the conditions expressed by the other parameters) are included in each returned Object. If the PropertyList contains duplicate elements, the Server MUST ignore the duplicates but otherwise process the request normally. If the PropertyList contains elements which are invalid CIMProperty names for any target Object, the Server MUST ignore such entries but otherwise process the request normally. Clients SHOULD NOT explicitly specify properties in the PropertyList parameter unless they have specified a non-NULL value for the ResultClass parameter.

Returns:

If successful, the method returns zero or more CIM Classes or Instances meeting the requested criteria. Since it is possible for CIM Objects from different hosts or namespaces to be associated, each returned Object includes location information. If unsuccessful, one of the following status codes MUST be returned by this method, where the first applicable error in the list (starting with the first element of the list, and working down) is the error returned. Any additional method-specific interpretation of the error in is given in parentheses.

- CIM_ERR_ACCESS_DENIED
- CIM_ERR_NOT_SUPPORTED
- CIM_ERR_INVALID_NAMESPACE
- CIM_ERR_INVALID_PARAMETER (including missing, duplicate, unrecognized or otherwise incorrect parameters)
- CIM_ERR_FAILED (some other unspecified error occurred)

virtual Array<CIMReference> referenceNames(const String& nameSpace, const CIMReference& objectName, const String& resultClass = String::EMPTY, const String& role = String::EMPTY)

The referenceNames operation is used to enumerate the association objects that refer to a particular target CIM Object (Class or Instance).

Parameters:

The - NameSpace parameter is a string that defines the target namespace NAMESPACE
The - ObjectName input parameter defines the target CIM Object whose referring object names are to be returned. It may be either a Class name or an Instance name (model path).
The - ResultClass input parameter, if not NULL, MUST be a valid CIM Class name. It acts as a filter on the returned set of Object Names by mandating that each returned Object CIMName MUST identify an Instance of this Class (or one of its subclasses), or this Class (or one of its subclasses).
The - role input parameter, if not NULL, MUST be a valid CIMProperty name. It acts as a filter on the returned set of Object Names by mandating that each returned Object CIMName MUST identify an Object that refers to the target Instance via a CIMProperty whose name matches the value of this parameter.

Returns:

If successful,the method returns the names of zero or more full CIM Class paths or Instance paths of Objects meeting the requested criteria. Since it is possible for CIM Objects from different hosts or namespaces to be associated, each returned path is an absolute path that includes host and namespace information. If unsuccessful, one of the following status codes MUST be returned by this method, where the first applicable error in the list (starting with the first element of the list, and working down) is the error returned. Any additional method-specific interpretation of the error in is given in parentheses.

- CIM_ERR_ACCESS_DENIED
- CIM_ERR_NOT_SUPPORTED
- CIM_ERR_INVALID_NAMESPACE

- CIM_ERR_INVALID_PARAMETER (including missing, duplicate, unrecognized or otherwise incorrect parameters)
- CIM_ERR_FAILED (some other unspecified error occurred)

virtual CIMValue getProperty(const String& nameSpace, const CIMReference& instanceName, const String& propertyName)

This operation is used to retrieve a single property value from a CIM Instance in the target Namespace.

Parameters:

NameSpace - The NameSpace parameter is a string that defines the target namespace NAMESPACE
The - InstanceName input parameter specifies the name of the Instance (model path) from which the CIMProperty value is requested. \INSTANCENAME
The - PropertyName input parameter specifies the name of the CIMProperty whose value is to be returned.

Returns:

If successful, the return value specifies the value of the requested CIMProperty. If the value is NULL then no element is returned. If unsuccessful, one of the following status codes MUST be returned by this method, where the first applicable error in the list (starting with the first element of the list, and working down) is the error returned. Any additional method-specific interpretation of the error in is given in parentheses.

- CIM_ERR_ACCESS_DENIED
- CIM_ERR_INVALID_NAMESPACE
- CIM_ERR_INVALID_PARAMETER (including missing, duplicate, unrecognized or otherwise incorrect parameters)
- CIM_ERR_INVALID_CLASS (the CIM Class does not exist in the specified namespace)
- CIM_ERR_NOT_FOUND (the CIM Class does exist, but the requested CIM Instance does not exist in the specified namespace)
-
- CIM_ERR_NO_SUCH_PROPERTY (the CIM Instance does exist, but the requested CIMProperty does not)
- CIM_ERR_FAILED (some other unspecified error occurred)

virtual void setProperty(const String& nameSpace, const CIMReference& instanceName, const String& propertyName, const CIMValue& newValue = CIMValue())

The setProperty operation sets a single property value in a CIM Instance in the target Namespace.

Parameters:

NameSpace - The NameSpace parameter is a string that defines the target namespace NAMESPACE
InstanceName - The InstanceName input parameter specifies the name of the Instance (model path) for which the CIMProperty value is to be updated.

PropertyName - The PropertyName input parameter specifies the name of the CIMProperty whose value is to be updated.
Newvalue - The NewValue input parameter specifies the new value for the CIMProperty (which may be NULL).

Returns:

If unsuccessful, one of the following status codes MUST be returned by this method, where the first applicable error in the list (starting with the first element of the list, and working down) is the error returned. Any additional method-specific interpretation of the error in is given in parentheses.

- CIM_ERR_ACCESS_DENIED
- CIM_ERR_INVALID_NAMESPACE
- CIM_ERR_INVALID_PARAMETER (including missing,duplicate, unrecognized or otherwise incorrect parameters)
- CIM_ERR_INVALID_CLASS (the CIM Class does not exist in the specified namespace)
- CIM_ERR_NOT_FOUND (the CIM Class does exist, but the requested CIM Instance does not exist in the specified namespace)
- CIM_ERR_NO_SUCH_PROPERTY (the CIM Instance does exist, but the requested CIMProperty does not)
- CIM_ERR_FAILED (some other unspecified error occurred)

virtual CIMQualifierDecl getQualifier(const String& nameSpace, const String& qualifierName)

The getQualifier operation retrieves a single CIMQualifier declaration from the target Namespace.

Parameters:

NameSpace - The NameSpace parameter is a string that defines the target namespace NAMESPACE
QualifierName - The QualifierName input parameter identifies the CIMQualifier whose declaration to be retrieved.

Returns:

If successful, the method returns the CIMQualifier declaration for the named CIMQualifier. If unsuccessful, one of the following status codes MUST be returned by this method, where the first applicable error in the list (starting with the first element of the list, and working down) is the error returned. Any additional method-specific interpretation of the error in is given in parentheses.

- CIM_ERR_ACCESS_DENIED
- CIM_ERR_NOT_SUPPORTED
- CIM_ERR_INVALID_NAMESPACE
- CIM_ERR_INVALID_PARAMETER (including missing, duplicate, unrecognized or otherwise incorrect parameters)
- CIM_ERR_INVALID_CLASS (the CIM Class does not exist in the specified namespace)
- CIM_ERR_NOT_FOUND (the CIM Class does exist, but the requested CIM Instance does not exist in the specified namespace)

- CIM_ERR_NO_SUCH_PROPERTY (the CIM Instance does exist, but the requested CIMProperty does not)
- CIM_ERR_TYPE_MISMATCH (the supplied value is incompatible with the type of the CIMProperty)
- CIM_ERR_FAILED (some other unspecified error occurred)

virtual void setQualifier(const String& nameSpace, const CIMQualifierDecl& qualifierDecl)

The setQualifier creates or update a single CIMQualifier declaration in the target Namespace. If the CIMQualifier declaration already exists it is overwritten.

Parameters:

NameSpace - The NameSpace parameter is a string that defines the target namespace NAMESPACE
- CIMQualifier Declaration The QualifierDeclaration input parameter defines the CIMQualifier Declaration to be added to the Namespace. @return If successful, the CIMQualifier declaration MUST have been added to the target Namespace. If a CIMQualifier declaration with the same CIMQualifier name already existed, then it MUST have been replaced by the new declaration.

Returns:

If unsuccessful, one of the following status codes MUST be returned by this method, where the first applicable error in the list (starting with the first element of the list, and working down) is the error returned. Any additional method-specific interpretation of the error in is given in parentheses.

- CIM_ERR_ACCESS_DENIED
- CIM_ERR_NOT_SUPPORTED
- CIM_ERR_INVALID_NAMESPACE
- CIM_ERR_INVALID_PARAMETER (including missing, duplicate, unrecognized or otherwise incorrect parameters)
- CIM_ERR_NOT_FOUND (the requested CIMQualifier declaration did not exist)
- CIM_ERR_FAILED (some other unspecified error occurred)

virtual void deleteQualifier(const String& nameSpace, const String& qualifierName)

The deleteQualifier operation deletes a single CIMQualifier declaration from the target Namespace.

Parameters:

NameSpace - The NameSpace parameter is a string that defines the target namespace NAMESPACE
QualifierName - The QualifierName input parameter identifies the CIMQualifier whose declaration to be deleted. @return If successful, the specified CIMQualifier declaration MUST have been deleted from the Namespace.

Returns:

If unsuccessful, one of the following status codes MUST be returned by this method, where the first applicable error in the list (starting with the first element of the list, and working down) is the

error returned. Any additional method-specific interpretation of the error in is given in parentheses.

- CIM_ERR_ACCESS_DENIED
- CIM_ERR_NOT_SUPPORTED
- CIM_ERR_INVALID_NAMESPACE
- CIM_ERR_INVALID_PARAMETER (including missing, duplicate, unrecognized or otherwise incorrect parameters)
- CIM_ERR_FAILED (some other unspecified error occurred)

virtual Array<CIMQualifierDecl> enumerateQualifiers(const String& nameSpace)

The enumerateQualifiers operation is used to enumerate CIMQualifier declarations from the target Namespace.

Parameters:

NameSpace - The NameSpace parameter is a string that defines the target namespace NAMESPACE

Returns:

If successful, the method returns zero or more CIMQualifier declarations. If unsuccessful, one of the following status codes MUST be returned by this method, where the first applicable error in the list (starting with the first element of the list, and working down) is the error returned. Any additional method-specific interpretation of the error in is given in parentheses.

- CIM_ERR_ACCESS_DENIED
- CIM_ERR_NOT_SUPPORTED
- CIM_ERR_INVALID_NAMESPACE
- CIM_ERR_INVALID_PARAMETER (including missing, duplicate, unrecognized or otherwise incorrect parameters)
- CIM_ERR_NOT_FOUND (the requested CIMQualifier declaration did not exist)
- CIM_ERR_FAILED (some other unspecified error occurred)

virtual CIMValue invokeMethod(const String& nameSpace, const CIMReference& instanceName, const String& methodName, const Array<CIMValue>& inParameters, Array<CIMValue>& outParameters)

Any CIM Server is assumed to support extrinsic methods. Extrinsic methods are defined by the Schema supported by the Cim Server. If a CIM Server does not support extrinsic method invocations, it MUST (subject to the considerations described in the rest of this section) return the error code CIM_ERR_NOT_SUPPORTED to any request to execute an extrinsic method. This allows a CIM client to determine that all attempts to execute extrinsic methods will fail.

@retrun - If the Cim Server is unable to perform the extrinsic method invocation, one of the following status codes MUST be returned by the CimServer, where the first applicable error in the list (starting with the first element of the list, and working down) is the error returned. Any additional specific interpretation of the error is given in parentheses.

- CIM_ERR_ACCESS_DENIED

- CIM_ERR_NOT_SUPPORTED (the CimServer does not support extrinsic method invocations)
- CIM_ERR_INVALID_NAMESPACE
- CIM_ERR_INVALID_PARAMETER (including missing, duplicate, unrecognized or otherwise incorrect parameters)
- CIM_ERR_NOT_FOUND (the target CIM Class or instance does not exist in the specified namespace)
- CIM_ERR_METHOD_NOT_FOUND
- CIM_ERR_METHOD_NOT_AVAILABLE (the CimServer is unable to honor the invocation request)
- CIM_ERR_FAILED (some other unspecified error occurred)

Direct child classes:

CIMRepository
CIMClient

Alphabetic index HTML hierarchy of classes or Java

In file ..\..\src\Pegasus\Client\CIMClient.h:

class PEGASUS_CLIENT_LINKAGE CIMClient

Class CIMClient - This class defines the client interfaces for Pegasus.

Documentation

Class CIMClient - This class defines the client interfaces for Pegasus. These are the interfaces that could be used by a CIM CIMClient written in C++. These interfaces are based completely on the operations interfaces defined in the Pegasus CIMOperations.h file with some extensions for the client interface. @see "The CIMOperations Class"

Inheritance:

Public Methods

	CIMClient (Selector* selector, Uint32 timeOutMilliseconds = DEFAULT_TIMEOUT_MILLISECONDS)
	~CIMClient ()
Uint32	**getTimeOut** () const
void	**setTimeOut** (Uint32 timeOutMilliseconds)
void	**connect** (const char* address)
void	**get** (const char* document) const
void	**runOnce** ()
void	**runForever** ()
virtual CIMClass	**getClass** (const String& nameSpace, const String& className, Boolean localOnly = true, Boolean includeQualifiers = true, Boolean includeClassOrigin = false, const Array<String>& propertyList = StringArray())
virtual CIMInstance	**getInstance** (const String& nameSpace, const CIMReference& instanceName, Boolean localOnly = true, Boolean includeQualifiers = false, Boolean includeClassOrigin = false, const Array<String>& propertyList = StringArray())
virtual void	**deleteClass** (const String& nameSpace, const String& className)

virtual void	**deleteInstance** (const String& nameSpace, const CIMReference& instanceName)
virtual void	**createClass** (const String& nameSpace, CIMClass& newClass)
virtual void	**createInstance** (const String& nameSpace, CIMInstance& newInstance)
virtual void	**modifyClass** (const String& nameSpace, CIMClass& modifiedClass)
virtual void	**modifyInstance** (const String& nameSpace, const CIMInstance& modifiedInstance)
virtual Array<CIMClass>	**enumerateClasses** (const String& nameSpace, const String& className = String::EMPTY, Boolean deepInheritance = false, Boolean localOnly = true, Boolean includeQualifiers = true, Boolean includeClassOrigin = false)
virtual Array<String>	**enumerateClassNames** (const String& nameSpace, const String& className = String::EMPTY, Boolean deepInheritance = false)
virtual Array<CIMInstance>	**enumerateInstances** (const String& nameSpace, const String& className, Boolean deepInheritance = true, Boolean localOnly = true, Boolean includeQualifiers = false, Boolean includeClassOrigin = false, const Array<String>& propertyList = StringArray())
virtual Array<CIMReference>	**enumerateInstanceNames** (const String& nameSpace, const String& className)
virtual Array<CIMInstance>	**execQuery** (const String& queryLanguage, const String& query)
virtual Array<CIMInstance>	**associators** (const String& nameSpace, const CIMReference& objectName, const String& assocClass = String::EMPTY, const String& resultClass = String::EMPTY, const String& role = String::EMPTY, const String& resultRole = String::EMPTY, Boolean includeQualifiers = false, Boolean includeClassOrigin = false, const Array<String>& propertyList = StringArray())
virtual Array<CIMReference>	**associatorNames** (const String& nameSpace, const CIMReference& objectName, const String& assocClass = String::EMPTY, const String& resultClass = String::EMPTY, const String& role = String::EMPTY, const String& resultRole = String::EMPTY)

virtual Array<CIMInstance>	**references** (const String& nameSpace, const CIMReference& objectName, const String& resultClass = String::EMPTY, const String& role = String::EMPTY, Boolean includeQualifiers = false, Boolean includeClassOrigin = false, const Array<String>& propertyList = StringArray())
virtual Array<CIMReference>	**referenceNames** (const String& nameSpace, const CIMReference& objectName, const String& resultClass = String::EMPTY, const String& role = String::EMPTY)
virtual CIMValue	**getProperty** (const String& nameSpace, const CIMReference& instanceName, const String& propertyName)
virtual CIMQualifierDecl	**getQualifier** (const String& nameSpace, const String& qualifierName)
virtual void	**setQualifier** (const String& nameSpace, const CIMQualifierDecl& qualifierDeclaration)
virtual void	**deleteQualifier** (const String& nameSpace, const String& qualifierName)
virtual Array<CIMQualifierDecl>	**enumerateQualifiers** (const String& nameSpace)
virtual CIMValue	**invokeMethod** (const String& nameSpace, const CIMReference& instanceName, const String& methodName, const Array<CIMValue>& inParameters, Array<CIMValue>& outParameters)

Inherited from CIMOperations:

Public Methods

- virtual void **setProperty**(const String& nameSpace, const CIMReference& instanceName, const String& propertyName, const CIMValue& newValue = CIMValue())
- **`CIMClient( Selector* selector, Uint32 timeOutMilliseconds = DEFAULT_TIMEOUT_MILLISECONDS)`**
- **`~CIMClient()`**
- **`Uint32 getTimeOut() const`**
- **`void setTimeOut(Uint32 timeOutMilliseconds)`**
- **`void connect(const char* address)`**
- **`void get(const char* document) const`**
- **`void runOnce()`**

- void runForever()
- virtual CIMClass getClass(const String& nameSpace, const String& className, Boolean localOnly = true, Boolean includeQualifiers = true, Boolean includeClassOrigin = false, const Array<String>& propertyList = StringArray())
- virtual CIMInstance getInstance(const String& nameSpace, const CIMReference& instanceName, Boolean localOnly = true, Boolean includeQualifiers = false, Boolean includeClassOrigin = false, const Array<String>& propertyList = StringArray())
- virtual void deleteClass(const String& nameSpace, const String& className)
- virtual void deleteInstance(const String& nameSpace, const CIMReference& instanceName)
- virtual void createClass(const String& nameSpace, CIMClass& newClass)
- virtual void createInstance(const String& nameSpace, CIMInstance& newInstance)
- virtual void modifyClass(const String& nameSpace, CIMClass& modifiedClass)
- virtual void modifyInstance(const String& nameSpace, const CIMInstance& modifiedInstance)
- virtual Array<CIMClass> enumerateClasses(const String& nameSpace, const String& className = String::EMPTY, Boolean deepInheritance = false, Boolean localOnly = true, Boolean includeQualifiers = true, Boolean includeClassOrigin = false)
- virtual Array<String> enumerateClassNames(const String& nameSpace, const String& className = String::EMPTY, Boolean deepInheritance = false)
- virtual Array<CIMInstance> enumerateInstances(const String& nameSpace, const String& className, Boolean deepInheritance = true, Boolean localOnly = true, Boolean includeQualifiers = false, Boolean includeClassOrigin = false, const Array<String>& propertyList = StringArray())
- virtual Array<CIMReference> enumerateInstanceNames(const String& nameSpace, const String& className)
- virtual Array<CIMInstance> execQuery(const String& queryLanguage,

const String& query)

- virtual Array<CIMInstance> associators(const String& nameSpace, const CIMReference& objectName, const String& assocClass = String::EMPTY, const String& resultClass = String::EMPTY, const String& role = String::EMPTY, const String& resultRole = String::EMPTY, Boolean includeQualifiers = false, Boolean includeClassOrigin = false, const Array<String>& propertyList = StringArray())
- virtual Array<CIMReference> associatorNames(const String& nameSpace, const CIMReference& objectName, const String& assocClass = String::EMPTY, const String& resultClass = String::EMPTY, const String& role = String::EMPTY, const String& resultRole = String::EMPTY)
- virtual Array<CIMInstance> references(const String& nameSpace, const CIMReference& objectName, const String& resultClass = String::EMPTY, const String& role = String::EMPTY, Boolean includeQualifiers = false, Boolean includeClassOrigin = false, const Array<String>& propertyList = StringArray())
- virtual Array<CIMReference> referenceNames(const String& nameSpace, const CIMReference& objectName, const String& resultClass = String::EMPTY, const String& role = String::EMPTY)
- virtual CIMValue getProperty(const String& nameSpace, const CIMReference& instanceName, const String& propertyName)
- virtual CIMQualifierDecl getQualifier(const String& nameSpace, const String& qualifierName)
- virtual void setQualifier(const String& nameSpace, const CIMQualifierDecl& qualifierDeclaration)
- virtual void deleteQualifier(const String& nameSpace, const String& qualifierName)
- virtual Array<CIMQualifierDecl> enumerateQualifiers(const String& nameSpace)
- virtual CIMValue invokeMethod(const String& nameSpace, const CIMReference& instanceName, const String& methodName, const Array<CIMValue>& inParameters, Array<CIMValue>& outParameters)

This class has no child classes.

Pegasus Documentation

In file ..\..\src\Pegasus\Repository\CIMRepository.h:

class PEGASUS_REPOSITORY_LINKAGE CIMRepository

This class derives from the CIMOperations class and provides a simple implementation of a CIM repository.

Documentation

This class derives from the CIMOperations class and provides a simple implementation of a CIM repository. It only implements the methods for manipulating classes and instances. The others throw this exception:

```
CIMException(CIMException::NOT_SUPPORTED)
```

Inheritance:

Public Methods

	CIMRepository (const String& repositoryRoot) *Constructor*
virtual	**~CIMRepository** () *Descructor*
virtual CIMClass	**getClass** (const String& nameSpace, const String& className, Boolean localOnly = true, Boolean includeQualifiers = true, Boolean includeClassOrigin = false, const Array<String>& propertyList = StringArray()) *virtual class CIMClass.*
virtual CIMInstance	**getInstance** (const String& nameSpace, const CIMReference& instanceName, Boolean localOnly = true, Boolean includeQualifiers = false, Boolean includeClassOrigin = false, const Array<String>& propertyList = StringArray()) *getInstance*
virtual void	**deleteClass** (const String& nameSpace, const String& className) *deleteClass*

virtual void	**deleteInstance** (const String& nameSpace, const CIMReference& instanceName) *deleteInstance*
virtual void	**createClass** (const String& nameSpace, CIMClass& newClass) *createClass*
virtual void	**createInstance** (const String& nameSpace, CIMInstance& newInstance) *createInstance*
virtual void	**modifyClass** (const String& nameSpace, CIMClass& modifiedClass) *modifyClass*
virtual void	**modifyInstance** (const String& nameSpace, const CIMInstance& modifiedInstance) *modifyInstance*
virtual Array<CIMClass>	**enumerateClasses** (const String& nameSpace, const String& className = String::EMPTY, Boolean deepInheritance = false, Boolean localOnly = true, Boolean includeQualifiers = true, Boolean includeClassOrigin = false) *enumerateClasses*
virtual Array<String>	**enumerateClassNames** (const String& nameSpace, const String& className = String::EMPTY, Boolean deepInheritance = false) *enumerateClassNames*
virtual Array<CIMInstance>	**enumerateInstances** (const String& nameSpace, const String& className, Boolean deepInheritance = true, Boolean localOnly = true, Boolean includeQualifiers = false, Boolean includeClassOrigin = false, const Array<String>& propertyList = StringArray()) *enumerateInstances*
virtual Array<CIMReference>	**enumerateInstanceNames** (const String& nameSpace, const String& className) *enumerateInstanceNames*
virtual Array<CIMInstance>	**execQuery** (const String& queryLanguage, const String& query) *execQuery*
virtual Array<CIMInstance>	**associators** (const String& nameSpace, const CIMReference& objectName, const String& assocClass = String::EMPTY, const String& resultClass = String::EMPTY, const String& role = String::EMPTY, const String& resultRole = String::EMPTY, Boolean includeQualifiers = false, Boolean includeClassOrigin = false, const Array<String>& propertyList = StringArray()) *associators*

virtual Array<CIMReference>	**associatorNames** (const String& nameSpace, const CIMReference& objectName, const String& assocClass = String::EMPTY, const String& resultClass = String::EMPTY, const String& role = String::EMPTY, const String& resultRole = String::EMPTY) *associateNames*
virtual Array<CIMInstance>	**references** (const String& nameSpace, const CIMReference& objectName, const String& resultClass = String::EMPTY, const String& role = String::EMPTY, Boolean includeQualifiers = false, Boolean includeClassOrigin = false, const Array<String>& propertyList = StringArray()) *references*
virtual Array<CIMReference>	**referenceNames** (const String& nameSpace, const CIMReference& objectName, const String& resultClass = String::EMPTY, const String& role = String::EMPTY) *referenceNames*
virtual CIMValue	**getProperty** (const String& nameSpace, const CIMReference& instanceName, const String& propertyName) *getProperty*
virtual void	**setProperty** (const String& nameSpace, const CIMReference& instanceName, const String& propertyName, const CIMValue& newValue = CIMValue()) *setProperty*
virtual CIMQualifierDecl	**getQualifier** (const String& nameSpace, const String& qualifierName) *getQualifier*
virtual void	**setQualifier** (const String& nameSpace, const CIMQualifierDecl& qualifierDecl) *setQualifier*
virtual void	**deleteQualifier** (const String& nameSpace, const String& qualifierName) *virtual deleteQualifier*
virtual Array<CIMQualifierDecl>	**enumerateQualifiers** (const String& nameSpace) *enumerateQualifiers*
virtual CIMValue	**invokeMethod** (const String& nameSpace, const CIMReference& instanceName, const String& methodName, const Array<CIMValue>& inParameters, Array<CIMValue>& outParameters) *invokeMethod*

void	**createNameSpace** (const String& nameSpace) *CIMMethod createNameSpace - Creates a new namespace in the repository*
virtual Array<String>	**enumerateNameSpaces** () const *CIMMethod enumerateNameSpaces - Get all of the namespaces in the repository.*
void	**deleteNameSpace** (const String& nameSpace) *CIMMethod deleteNameSpace - Deletes a namespace in the repository.*

Inherited from CIMOperations:

- **CIMRepository(const String& repositoryRoot)**

  Constructor

- **virtual ~CIMRepository()**

  Descructor

- **virtual CIMClass getClass(const String& nameSpace, const String& className, Boolean localOnly = true, Boolean includeQualifiers = true, Boolean includeClassOrigin = false, const Array<String>& propertyList = StringArray())**

  virtual class CIMClass. From the operations class

- **virtual CIMInstance getInstance(const String& nameSpace, const CIMReference& instanceName, Boolean localOnly = true, Boolean includeQualifiers = false, Boolean includeClassOrigin = false, const Array<String>& propertyList = StringArray())**

  getInstance

- **virtual void deleteClass(const String& nameSpace, const String& className)**

  deleteClass

- **virtual void deleteInstance(const String& nameSpace, const CIMReference& instanceName)**

  deleteInstance

- **virtual void createClass(const String& nameSpace, CIMClass& newClass)**

  createClass

- **virtual void createInstance(const String& nameSpace, CIMInstance& newInstance)**

createInstance

- **virtual void modifyClass(const String& nameSpace, CIMClass& modifiedClass)**

modifyClass

- **virtual void modifyInstance(const String& nameSpace, const CIMInstance& modifiedInstance)**

modifyInstance

- **virtual Array<CIMClass> enumerateClasses(const String& nameSpace, const String& className = String::EMPTY, Boolean deepInheritance = false, Boolean localOnly = true, Boolean includeQualifiers = true, Boolean includeClassOrigin = false)**

enumerateClasses

- **virtual Array<String> enumerateClassNames(const String& nameSpace, const String& className = String::EMPTY, Boolean deepInheritance = false)**

enumerateClassNames

- **virtual Array<CIMInstance> enumerateInstances(const String& nameSpace, const String& className, Boolean deepInheritance = true, Boolean localOnly = true, Boolean includeQualifiers = false, Boolean includeClassOrigin = false, const Array<String>& propertyList = StringArray())**

enumerateInstances

- **virtual Array<CIMReference> enumerateInstanceNames(const String& nameSpace, const String& className)**

enumerateInstanceNames

- **virtual Array<CIMInstance> execQuery(const String& queryLanguage, const String& query)**

execQuery

- **virtual Array<CIMInstance> associators(const String& nameSpace, const CIMReference& objectName, const String& assocClass = String::EMPTY, const String& resultClass = String::EMPTY, const String& role = String::EMPTY, const String& resultRole = String::EMPTY, Boolean includeQualifiers = false, Boolean includeClassOrigin = false, const Array<String>& propertyList = StringArray())**

associators

- **virtual Array<CIMReference> associatorNames(const String&**

nameSpace, const CIMReference& objectName, const String& assocClass = String::EMPTY, const String& resultClass = String::EMPTY, const String& role = String::EMPTY, const String& resultRole = String::EMPTY)

associateNames

- **virtual Array<CIMInstance> references(const String& nameSpace, const CIMReference& objectName, const String& resultClass = String::EMPTY, const String& role = String::EMPTY, Boolean includeQualifiers = false, Boolean includeClassOrigin = false, const Array<String>& propertyList = StringArray())**

references

- **virtual Array<CIMReference> referenceNames(const String& nameSpace, const CIMReference& objectName, const String& resultClass = String::EMPTY, const String& role = String::EMPTY)**

referenceNames

- **virtual CIMValue getProperty(const String& nameSpace, const CIMReference& instanceName, const String& propertyName)**

getProperty

- **virtual void setProperty(const String& nameSpace, const CIMReference& instanceName, const String& propertyName, const CIMValue& newValue = CIMValue())**

setProperty

- **virtual CIMQualifierDecl getQualifier(const String& nameSpace, const String& qualifierName)**

getQualifier

- **virtual void setQualifier(const String& nameSpace, const CIMQualifierDecl& qualifierDecl)**

setQualifier

- **virtual void deleteQualifier(const String& nameSpace, const String& qualifierName)**

virtual deleteQualifier

- **virtual Array<CIMQualifierDecl> enumerateQualifiers(const String& nameSpace)**

enumerateQualifiers

- **virtual CIMValue invokeMethod(const String& nameSpace, const CIMReference& instanceName, const String& methodName, const**

Array<CIMValue>& inParameters, Array<CIMValue>& outParameters)

invokeMethod

- **void createNameSpace(const String& nameSpace)**

  CIMMethod createNameSpace - Creates a new namespace in the repository

  Throws:

  - Throws "Already_Exists if the Namespace exits. Throws "CannotCreateDirectory" if there are problems in the creation.

  Parameters:

  - String with the name of the namespace

- **virtual Array<String> enumerateNameSpaces() const**

  CIMMethod enumerateNameSpaces - Get all of the namespaces in the repository. NAMESPACE

  Returns:

  Array of strings with the namespaces

- **void deleteNameSpace(const String& nameSpace)**

  CIMMethod deleteNameSpace - Deletes a namespace in the repository. The deleteNameSpace method will only delete a namespace if there are no classed defined in the namespace. Today this is a Pegasus characteristics and not defined as part of the DMTF standards.

  Throws:

  - Throws NoSuchDirectory if the Namespace does not exist.

  Parameters:

  - String with the name of the namespace

This class has no child classes.

Alphabetic index HTML hierarchy of classes or Java

In file ..\..\src\Pegasus\Common\Array.h:

template<class T> class Array

Array Class.

Documentation

Array Class. This clas is used to represent arrays of intrinsic data types in CIM. And it is also used by the implementation to represent arrays of other kinds of objects (e.g., it is used to implement the String class). However, the user will only use it directly to manipulate arrays of CIM data types.

Inheritance:

Public Methods

	Array () *Default constructor*
	Array (const Array<T>& x) *Copy Constructor*
	Array (Uint32 size) *Constructs an array with size elements.*
	Array (Uint32 size, const T& x) *Constructs an array with size elements.*
	Array (const T* items, Uint32 size) *Constructs an array with size elements.*
	~Array () *Destructs the objects, freeing any resources*
Array<T> &	**operator=** (const Array<T>& x) *Assignment operator*
void	**clear** () *Clears the contents of the array.*
void	**reserve** (Uint32 capacity) *Reserves memory for capacity elements.*
void	**grow** (Uint32 size, const T& x) *Make the size of the array grow by size elements.*

void	**swap** (Array<T>& x) *Swaps the contents of two arrays*
Uint32	**size** () const *Returns the number of elements in the array.*
Uint32	**getCapacity** () const *Returns the capacity of the array*
const T*	**getData** () const *Returns a pointer to the first element of the array*
T&	**operator[]** (Uint32 pos) *Returns the element at the index given by the pos argument.*
const T&	**operator[]** (Uint32 pos) const *Same as the above method except that this is the version called on const arrays.*
void	**append** (const T& x) *Appends an element to the end of the array.*
void	**append** (const T* x, Uint32 size) *Appends size elements at x to the end of this array*
void	**appendArray** (const Array<T>& x) *Appends one array to another.*
void	**prepend** (const T& x) *Appends one element to the beginning of the array.*
void	**prepend** (const T* x, Uint32 size) *Appends size elements to the array starting at the memory address given by x.*
void	**insert** (Uint32 pos, const T& x) *Inserts the element at the given index in the array.*
void	**insert** (Uint32 pos, const T* x, Uint32 size) *Inserts size elements at x into the array at the given position.*
void	**remove** (Uint32 pos) *Removes the element at the given position from the array.*
void	**remove** (Uint32 pos, Uint32 size) *Removes size elements starting at the given position.*

- **`Array()`**

  Default constructor

- **`Array(const Array<T>& x)`**

  Copy Constructor

- **`Array(Uint32 size)`**

  Constructs an array with size elements. The elements are initialized with their copy constructor.

  Parameters:

  - size defines the number of elements

- **`Array(Uint32 size, const T& x)`**

  Constructs an array with size elements. The elements are initialized with x.

- **`Array(const T* items, Uint32 size)`**

  Constructs an array with size elements. The values come from the items pointer.

- **`~Array()`**

  Destructs the objects, freeing any resources

- **`Array<T> & operator=(const Array<T>& x)`**

  Assignment operator

- **`void clear()`**

  Clears the contents of the array. After calling this, size() returns zero.

- **`void reserve(Uint32 capacity)`**

  Reserves memory for capacity elements. Notice that this does not change the size of the array (size() returns what it did before). If the capacity of the array is already greater or equal to the capacity argument, this method has no effect. After calling reserve(), getCapacity() returns a value which is greater or equal to the capacity argument.

  Parameters:

  capacity - defines the size that is to be reserved

- **`void grow(Uint32 size, const T& x)`**

  Make the size of the array grow by size elements. Thenew size will be size() + size. The new elements (there are size of them) are initialized with x. @parm size defines the number of elements by which the array is to grow.

- **`void swap(Array<T>& x)`**

  Swaps the contents of two arrays

- **`Uint32 size() const`**

  Returns the number of elements in the array.

  Returns:

  The number of elements in the array.

- **`Uint32 getCapacity() const`**

  Returns the capacity of the array

- **`const T* getData() const`**

  Returns a pointer to the first element of the array

- **`T& operator[](Uint32 pos)`**

  Returns the element at the index given by the pos argument.

  Returns:

  A reference to the elementdefined by index so that it may be modified.

- **`const T& operator[](Uint32 pos) const`**

  Same as the above method except that this is the version called on const arrays. The return value cannot be modified since it is constant.

- **`void append(const T& x)`**

  Appends an element to the end of the array. This increases the size of the array by one.

  Parameters:

  Element - to append.

- **`void append(const T* x, Uint32 size)`**

  Appends size elements at x to the end of this array

- **`void appendArray(const Array<T>& x)`**

  Appends one array to another. The size of this array grows by the size of the other.

  Parameters:

  - Array to append.

- **`void prepend(const T& x)`**

  Appends one element to the beginning of the array. This increases the size by one.

  Parameters:

  Element - to prepend.

- **`void prepend(const T* x, Uint32 size)`**

  Appends size elements to the array starting at the memory address given by x. The array grows by size elements.

- **`void insert(Uint32 pos, const T& x)`**

  Inserts the element at the given index in the array. Subsequent elements are moved down. The size of the array grows by one.

- **`void insert(Uint32 pos, const T* x, Uint32 size)`**

  Inserts size elements at x into the array at the given position. Subsequent elements are moved down. The size of the array grows by size elements.

- **`void remove(Uint32 pos)`**

  Removes the element at the given position from the array. The size of the array shrinks by one.

- **`void remove(Uint32 pos, Uint32 size)`**

  Removes size elements starting at the given position. The size of the array shrinks by size elements.

This class has no child classes.

Friends:

CIMValue

Pegasus Documentation

In file ..\..\src\Pegasus\Common\CIMClass.h:

class PEGASUS_COMMON_LINKAGE CIMClass

The CIMClass class is used to represent CIM classes in Pegasus.

Documentation

The CIMClass class is used to represent CIM classes in Pegasus. In CIM, a class object may be a class or an associator. A CIM class must contain a name and may contain methods, properties, and qualifiers. It is a template for creating a CIM instance. A CIM class represents a collection of CIM instances, all of which support a common type (for example, a set of properties, methods, and associations).

Inheritance:

Public Methods

	CIMClass () *Constructor - Creates an uninitiated a new CIM object reprenting a CIM class.*
	CIMClass (const CIMClass& x) *Constructor - Creates a class from a previous class*
CIMClass&	**operator=** (const CIMClass& x) *Operator = Assigns the CIM Class constructor*
	CIMClass (const String& className, const String& superClassName = String()) *Constructor - Creates a Class from inputs of a classname and SuperClassName*
	~CIMClass () *Destructor*
Boolean	**isAssociation** () const *CIMMethod isAssociation - Identifies whether or not this CIM class is an association.*
Boolean	**isAbstract** () const *CIMMethod isAbstract*
const String&	**getClassName** () const *CIMMethod Gets the name of the class ATTN: COMMENT.*
const String&	**getSuperClassName** () const *CIMMethod getSuperClassName - Gets the name of the Parent*

void	**setSuperClassName** (const String& superClassName) *CIMMethod setSuperClassName - Sets the name of the parent class from the input parameter.*
CIMClass&	**addQualifier** (const CIMQualifier& qualifier) *CIMMethod addQualifier - Adds the specified qualifier to the class and increments the qualifier count.*
Uint32	**findQualifier** (const String& name) *CIMMethod findQualifier - Finds a qualifier with the specified input name if it exists in the class @param name CIMName of the qualifier to be found @return Position of the qualifier in the Class ATTN: Clarify the return.*
CIMQualifier	**getQualifier** (Uint32 pos) *CIMMethod getQualifier - Gets the CIMQualifier object defined by the input parameter*
CIMConstQualifier	**getQualifier** (Uint32 pos) const *CIMMethod getQualifier - ATTN:*
Uint32	**getQualifierCount** () const *CIMMethod getQualifierCount - Returns the number of qualifiers in the class.*
CIMClass&	**addProperty** (const CIMProperty& x) *CIMMethod addProperty - Adds the specified property object to the properties in the CIM class*
void	**removeProperty** (Uint32 pos) *CIMMethod removeProperty - Removes the property represented by the position input parameter from the class*
Uint32	**findProperty** (const String& name) *CIMMethod findProperty - Finds the property object with the name defined by the input parameter in the class.*
CIMProperty	**getProperty** (Uint32 pos) *CIMMethod getProperty - Returns a property representing the property defined by the input parameter*
CIMConstProperty	**getProperty** (Uint32 pos) const *CIMMethod getProperty - ATTN*
Uint32	**getPropertyCount** () const *CIMMethod getProperty - Gets the count of the number of properties defined in the class.*
CIMClass&	**addMethod** (const CIMMethod& x) *CIMMethod addMethod - Adds the method object defined by the input parameter to the class and increments the count of the number of methods in the class*
Uint32	**findMethod** (const String& name) *CIMMethod findMethod - Located the method object defined by the name input*
CIMMethod	**getMethod** (Uint32 pos) *CIMMethod getMethod - Gets the method object defined by the input parameter.*

CIMConstMethod	**getMethod** (Uint32 pos) const *CIMMethod getMethod - ATTN:*
Uint32	**getMethodCount** () const *CIMMethod getMethodCount - Count of the number of methods in the class*
void	**resolve** (DeclContext* declContext, const String& nameSpace) *CIMMethod Resolve - Resolve the class: inherit any properties and qualifiers.*
	operator int () const *operator - ATTN:*
void	**toXml** (Array<Sint8>& out) const *CIMMethod toXML*
void	**print** (std::ostream &o=std::cout) const *CIMMethod print*
Boolean	**identical** (const CIMConstClass& x) const *CIMMethod identical - Compares with another class ATTN: Clarify exactly what identical means @parm Class object for the class to be compared*
CIMClass	**clone** () const *CIMMethod clone - ATTN:*

- **`CIMClass()`**

  Constructor - Creates an uninitiated a new CIM object reprenting a CIM class. The class object created by this constructor can only be used in an operation such as the copy constructor. It cannot be used to create a class by appending names, properties, etc. since it is unitiated.

  Use one of the other constructors to create an initiated new CIM class object.

  Throws:

  Throws an exception "unitialized handle" if this unitialized handle is used /REF(HPEGASUS_HANDLES)

- **`CIMClass(const CIMClass& x)`**

  Constructor - Creates a class from a previous class

- **`CIMClass& operator=(const CIMClass& x)`**

  Operator = Assigns the CIM Class constructor

- **`CIMClass( const String& className, const String& superClassName = String())`**

  Constructor - Creates a Class from inputs of a classname and SuperClassName

  Parameters:

  className - - String representing name of the class being created

  superClassName - - String representing name of the SuperClass ATTN: Define what makes up legal name.

  Returns:

  Throws IllegalName if className argument illegal CIM identifier.

```
CIMClass NewCass("MyClass", "YourClass");
```

- **~CIMClass()**

  Destructor

- **Boolean isAssociation() const**

  CIMMethod isAssociation - Identifies whether or not this CIM class is an association. An association is a relationship between two (or more) classes or instances of two classes. The properties of an association class include pointers, or references, to the two (or more) instances. All CIM classes can be included in one or more associations. ATTN: Move the association definition elsewhere

  Returns:

  Boolean True if this CIM class belongs to an association; otherwise, false.

- **Boolean isAbstract() const**

  CIMMethod isAbstract

- **const String& getClassName() const**

  CIMMethod Gets the name of the class ATTN: COMMENT. Why not just get name so we have common method for all.

- **const String& getSuperClassName() const**

  CIMMethod getSuperClassName - Gets the name of the Parent

  Returns:

  String with parent class name.

- **void setSuperClassName(const String& superClassName)**

  CIMMethod setSuperClassName - Sets the name of the parent class from the input parameter. \REF{CLASSNAME}. ATTN: Define legal classnames

  Parameters:

  - - String defining parent name.

  Returns:

  Throws IllegalName if superClassName argument not legal CIM identifier.

- **CIMClass& addQualifier(const CIMQualifier& qualifier)**

  CIMMethod addQualifier - Adds the specified qualifier to the class and increments the qualifier count. It is illegal to add the same qualifier more than one time.

  Parameters:

  - - CIMQualifier object representing the qualifier to be added ATTN: Pointer to qualifier object.

  Returns:

Throws AlreadyExists.

- **Uint32 findQualifier(const String& name)**

  CIMMethod findQualifier - Finds a qualifier with the specified input name if it exists in the class @param name CIMName of the qualifier to be found @return Position of the qualifier in the Class ATTN: Clarify the return. What if not found, etc.

- **CIMQualifier getQualifier(Uint32 pos)**

  CIMMethod getQualifier - Gets the CIMQualifier object defined by the input parameter

  Parameters:

  pos - defines the position of the qualifier in the class from the findQualifier method

  Returns:

  CIMQualifier object representing the qualifier found. ATTN: what is error return here?

- **CIMConstQualifier getQualifier(Uint32 pos) const**

  CIMMethod getQualifier - ATTN:

- **Uint32 getQualifierCount() const**

  CIMMethod getQualifierCount - Returns the number of qualifiers in the class.

  Returns:

  ATTN:

- **CIMClass& addProperty(const CIMProperty& x)**

  CIMMethod addProperty - Adds the specified property object to the properties in the CIM class

- **void removeProperty(Uint32 pos)**

  CIMMethod removeProperty - Removes the property represented by the position input parameter from the class

  Parameters:

  position - parameter for the property to be removed from the findPropety method

  Returns:

  ATTN:

- **Uint32 findProperty(const String& name)**

  CIMMethod findProperty - Finds the property object with the name defined by the input parameter in the class.

  Parameters:

  - String parameter with the property name.

  Returns:

  position representing the property object found. ATTN: Clarify case of not found

- **CIMProperty getProperty(Uint32 pos)**

  CIMMethod getProperty - Returns a property representing the property defined by the input

parameter

Parameters:

position - for this property ATTN: Should we not use something like handle for position???

Returns:

CIMProperty object ATTN: what is error return?

- **`CIMConstProperty getProperty(Uint32 pos) const`**

  CIMMethod getProperty - ATTN

- **`Uint32 getPropertyCount() const`**

  CIMMethod getProperty - Gets the count of the number of properties defined in the class.

  Returns:

  count of number of proerties in the class

- **`CIMClass& addMethod(const CIMMethod& x)`**

  CIMMethod addMethod - Adds the method object defined by the input parameter to the class and increments the count of the number of methods in the class

  Parameters:

  - - method object representing the method to be added

- **`Uint32 findMethod(const String& name)`**

  CIMMethod findMethod - Located the method object defined by the name input

  Parameters:

  - String representing the name of the method to be found

  Returns:

  Position of the method object in the class to be used in subsequent getmethod, etc. operations

- **`CIMMethod getMethod(Uint32 pos)`**

  CIMMethod getMethod - Gets the method object defined by the input parameter.

  Parameters:

  ATTN - : @ method object representing the method defined ATTN: Error???

- **`CIMConstMethod getMethod(Uint32 pos) const`**

  CIMMethod getMethod - ATTN:

- **`Uint32 getMethodCount() const`**

  CIMMethod getMethodCount - Count of the number of methods in the class

  Returns:

  integer representing the number of methods in the class

- **`void resolve( DeclContext* declContext, const String& nameSpace)`**

  CIMMethod Resolve - Resolve the class: inherit any properties and qualifiers. Make sure the

superClass really exists and is consistent with this class. Also set the propagated flag class-origin for each class feature. ATTN: explain why this here

- **`operator int() const`**

  operator - ATTN:

- **`void toXml(Array<Sint8>& out) const`**

  CIMMethod toXML

- **`void print(std::ostream &o=std::cout) const`**

  CIMMethod print

- **`Boolean identical(const CIMConstClass& x) const`**

  CIMMethod identical - Compares with another class ATTN: Clarify exactly what identical means @parm Class object for the class to be compared

  Returns:

  True if the classes are identical

- **`CIMClass clone() const`**

  CIMMethod clone - ATTN:

This class has no child classes.

Friends:

class CIMConstClass

Alphabetic index HTML hierarchy of classes or *Java*

Pegasus Documentation

In file ..\..\src\Pegasus\Common\CIMClass.h:

class PEGASUS_COMMON_LINKAGE CIMConstClass

CIMConstClass - ATTN: define this.

Documentation

CIMConstClass - ATTN: define this.

Inheritance:

This class has no child classes.

Friends:

class CIMClassRep
class CIMClass
class CIMInstanceRep

Alphabetic index *HTML hierarchy of classes* *or* *Java*

In file ..\..\src\Pegasus\Common\CIMDateTime.h:

class PEGASUS_COMMON_LINKAGE CIMDateTime

The CIMDateTime class represents the CIM datetime data type as a C++ class CIMDateTime.

Documentation

The CIMDateTime class represents the CIM datetime data type as a C++ class CIMDateTime. A CIM datetime may contain a date or an interval. CIMDateTime is an intrinsic CIM data type which represents the time as a formated fixedplength string.

```
A date has the following form:
    yyyymmddhhmmss.mmmmmmsutc

Where

    yyyy = year (0-1999)
    mm = month (1-12)
    dd = day (1-31)
    hh = hour (0-23)
    mm = minute (0-59)
    ss = second (0-59)
    mmmmmm = microseconds.
    s = '+' or '-' to represent the UTC sign.
    utc = UTC offset (same as GMT offset).

An interval has the following form:

    dddddddhhmmss.mmmmmm:000

Where

    dddddddd = days
    hh = hours
    mm = minutes
    ss = seconds
    mmmmmm = microseconds
```

Note that intervals always end in ":000" (this is how they are distinguished from dates).

Intervals are really durations since they do not specify a start and end time (as one expects when speaking about an interval). It is better to think of an interval as specifying time elapsed since some event.

CIMDateTime objects are constructed from C character strings or from other CIMDateTime objects. These character strings must be exactly twenty-five characters and conform to one of the forms idententified above.

CIMDateTime objects which are not explicitly initialized will be implicitly initialized with the null time interval:

00000000000000.000000:000

Instances can be tested for nullness with the isNull() method.

ATTN: Add inequalities.

Inheritance:

Public Fields

const CIMDateTime&	**x**

Public Methods

	CIMDateTime () *CIMDateTime CIMMethod*
	CIMDateTime (const char* str) *CIMDateTime CIMMethod creates the CIM CIMDateTime from a string constant*
	CIMDateTime (const CIMDateTime& x) *CIMDateTime CIMMethod - Creates the CIMDateTime instance from another CIMDateTime instance*
CIMDateTime&	**operator=** (const CIMDateTime& x) *CIMDateTime method again*
Boolean	**isNull** () const *CIMDateTime isNull method - Tests for an all zero date time* `CIMDateTime dt; dtclear(); assert(dtisNull());`
const char*	**getString** () const *method getString*
void	**set** (const char* str) *method set - Sets the date time.*
void	**clear** () *CIMDateTime method clear - Clears the datetime class instance*

- **`CIMDateTime()`**

  CIMDateTime CIMMethod

- **`CIMDateTime(const char* str)`**

  CIMDateTime CIMMethod creates the CIM CIMDateTime from a string constant

- **`CIMDateTime(const CIMDateTime& x)`**

  CIMDateTime CIMMethod - Creates the CIMDateTime instance from another CIMDateTime instance

- **`CIMDateTime& operator=(const CIMDateTime& x)`**

  CIMDateTime method again

- **`Boolean isNull() const`**

  CIMDateTime isNull method - Tests for an all zero date time

```
CIMDateTime dt;
dtclear();
assert(dtisNull());
```

  Returns:

  This method returns true of the contents are "00000000000000.000000:000". Else it returns false

- **`const char* getString() const`**

  method getString

- **`void set(const char* str)`**

  method set - Sets the date time. Creates the CIMDateTime instance from the input string constant which must be in the datetime format.

```
CIMDateTime dt;
```

```
dt.set("19991224120000.000000+360");
```

Throws:

Throws exception BadDateTimeFormat on format error.

Returns:

On format error, CIMDateTime set throws the exception BadDateTimeFormat

- **`void clear()`**

  CIMDateTime method clear - Clears the datetime class instance

- **`const CIMDateTime& x`**

This class has no child classes.

Friends:

Boolean operator==(

Alphabetic index *HTML hierarchy of classes* *or* *Java*

In file ..\..\src\Pegasus\Common\CIMInstance.h:

class PEGASUS_COMMON_LINKAGE CIMInstance

Class CIMInstance - The CIMInstance class represents the instance of a CIM class in Pegasus.

Documentation

Class CIMInstance - The CIMInstance class represents the instance of a CIM class in Pegasus. It is used manipulate instances and the characteristics of instances

---

Inheritance:

Public Methods

	CIMInstance () *Constructor - Create a CIM Instance object.*
	CIMInstance (const CIMInstance& x) *Constructor - Create a CIMInstance object from another Instance.*
CIMInstance&	**operator=** (const CIMInstance& x) *Constructor - ATTN*
	CIMInstance (const String& className) *Constructor - Creates an Instance object with the classname from the input parameters*
	~CIMInstance () *Destructor*
const String&	**getClassName** () const *getClassName - Returns the class name of the instance*
CIMInstance&	**addQualifier** (const CIMQualifier& qualifier) *addQualifier - Adds the CIMQualifier object to the instance.*
Uint32	**findQualifier** (const String& name) *findQualifier - Searches the instance for the qualifier object defined by the inputparameter.*
CIMQualifier	**getQualifier** (Uint32 pos) *getQualifier - Retrieves the qualifier object defined by the index input parameter.*

CIMConstQualifier	**getQualifier** (Uint32 pos) const *getQualifier - Retrieves the qualifier object defined by the index input parameter.*
Uint32	**getQualifierCount** () const *getQualifierCount - Gets the numbercount of CIMQualifierobjects defined for this CIMInstance.*
CIMInstance&	**addProperty** (const CIMProperty& x) *addProperty - Adds a property object defined by the input parameter to the CIMInstance*
Uint32	**findProperty** (const String& name) *findProperty - Searches the CIMProperty objects installed in the CIMInstance for property objects with the name defined by the input.*
CIMProperty	**getProperty** (Uint32 pos) *getProperty - Gets the CIMproperty object in the CIMInstance defined by the input index parameter.*
CIMConstProperty	**getProperty** (Uint32 pos) const *getProperty - Gets the CIMproperty object in the CIMInstance defined by the input index parameter.*
Uint32	**getPropertyCount** () const *getPropertyCount - Gets the numbercount of CIMProperty objects defined for this CIMInstance.*
	operator int () const *operator int() - ATTN:*
void	**resolve** (DeclContext* declContext, const String& nameSpace) *resolve - ATTN:*
void	**toXml** (Array<Sint8>& out) const *toXml - Creates an XML transformation of the CIMInstance compatiblewith the DMTF CIM Operations over HTTP defintions.*
void	**print** (std::ostream &o=std::cout) const *CIMMethod*
Boolean	**identical** (const CIMConstInstance& x) const *identical - Compares the CIMInstance with another CIMInstance defined by the input parameter for equality of all components.*
CIMInstance	**clone** () const *CIMMethod*
CIMReference	**getInstanceName** (const CIMConstClass& cimClass) const *getInstnaceName - Get the instance name of this instance.*

`CIMInstance()`

Constructor - Create a CIM Instance object.

Returns:

Instance created

- **CIMInstance(const CIMInstance& x)**

  Constructor - Create a CIMInstance object from another Instance.

  Parameters:

  Instance - object from which the new instance is created.

  Returns:

  New instance @example ATTN:

- **CIMInstance& operator=(const CIMInstance& x)**

  Constructor - ATTN

- **CIMInstance(const String& className)**

  Constructor - Creates an Instance object with the classname from the input parameters

  Throws:

  Throws IllegalName if className argument not legal CIM identifier. ATTN: Clarify the defintion of legal CIM identifier.

  Parameters:

  - - String className to be used with new instance object

  Returns:

  The new instance object

- **~CIMInstance()**

  Destructor

- **const String& getClassName() const**

  getClassName - Returns the class name of the instance

  Returns:

  String with the class name.

- **CIMInstance& addQualifier(const CIMQualifier& qualifier)**

  addQualifier - Adds the CIMQualifier object to the instance. Thows an exception of the CIMQualifier already exists in the instance

  Throws:

  Throws AlreadyExists.

  Parameters:

  - CIMQualifier object to add to instance

  Returns:

  ATTN:

- **Uint32 findQualifier(const String& name)**

  findQualifier - Searches the instance for the qualifier object defined by the inputparameter.

  Parameters:

- String defining the qualifier to be found

Returns:

- Index of the qualifier to be used in subsequent operations or -1 if the qualifier is not found.

CIMQualifier getQualifier(Uint32 pos)

getQualifier - Retrieves the qualifier object defined by the index input parameter. @ index for the qualifier object. The index to qualifier objects is zero-origin and continuous so that incrementing loops can be used to get all qualifier objects in a CIMInstnace.

Throws:

Throws the OutOfBounds exception if the index is out of bounds ATTN: What is effect of out of range index???

Returns:

: Returns qualifier object defined by index.

CIMConstQualifier getQualifier(Uint32 pos) const

getQualifier - Retrieves the qualifier object defined by the index input parameter. @ index for the qualifier object. The index to qualifier objects is zero-origin and continuous so that incrementing loops can be used to get all qualifier objects in a CIMInstnace.

Throws:

Throws the OutOfBounds exception if the index is out of bounds ATTN: What is effect of out of range index??? ATTN: Is the above statement correct???

Returns:

: Returns qualifier object defined by index.

Uint32 getQualifierCount() const

getQualifierCount - Gets the numbercount of CIMQualifierobjects defined for this CIMInstance.

Throws:

Throws the OutOfBounds exception if the index is out of bounds

Returns:

Count of the number of CIMQalifier objects in the CIMInstance.

CIMInstance& addProperty(const CIMProperty& x)

addProperty - Adds a property object defined by the input parameter to the CIMInstance

Throws:

Throws the exception AlreadyExists if the property already exists.

Parameters:

Property - Object to be added. See the CIM Property class for definition of the property object

Returns:

ATTN:

Uint32 findProperty(const String& name)

findProperty - Searches the CIMProperty objects installed in the CIMInstance for property objects with the name defined by the input.

Parameters:

- String with the name of the property object to be found

Returns:

Index in the CIM Instance to the property object if found or -1 if no property object found with the name defined by the input.

`CIMProperty getProperty(Uint32 pos)`

getProperty - Gets the CIMproperty object in the CIMInstance defined by the input index parameter.

Throws:

Throws the OutOfBounds exception if the index is out of bounds ATTN: What is the effect of out of range?

Parameters:

Index - to the property object in the CIMInstance. The index to qualifier objects is zero-origin and continuous so that incrementing loops can be used to get all qualifier objects in a CIMInstnace.

Returns:

CIMProperty object corresponding to the index.

`CIMConstProperty getProperty(Uint32 pos) const`

getProperty - Gets the CIMproperty object in the CIMInstance defined by the input index parameter.

Throws:

Throws the OutOfBounds exception if the index is out of bounds ATTN: What is the effect of out of range?

Parameters:

Index - to the property object in the CIMInstance. The index to qualifier objects is zero-origin and continuous so that incrementing loops can be used to get all qualifier objects in a CIMInstnace.

Returns:

CIMProperty object corresponding to the index.

`Uint32 getPropertyCount() const`

getPropertyCount - Gets the numbercount of CIMProperty objects defined for this CIMInstance.

Throws:

Throws the OutOfBounds exception if the index is out of bounds

Returns:

Count of the number of CIMProperty objects in the CIMInstance. Zero indicates that no CIMProperty objects are contained in the CIMInstance

- **`operator int() const`**

  operator int() - ATTN:

- **`void resolve(DeclContext* declContext, const String& nameSpace)`**

  resolve - ATTN:

- **`void toXml(Array<Sint8>& out) const`**

  toXml - Creates an XML transformation of the CIMInstance compatiblewith the DMTF CIM Operations over HTTP defintions.

  Returns:

  ATTN: This is incorrect and needs to be corrected.

- **`void print(std::ostream &o=std::cout) const`**

  CIMMethod

- **`Boolean identical(const CIMConstInstance& x) const`**

  identical - Compares the CIMInstance with another CIMInstance defined by the input parameter for equality of all components.

  Parameters:

  - CIMInstance to be compared

  Returns:

  Boolean true if they are identical

- **`CIMInstance clone() const`**

  CIMMethod

- **`CIMReference getInstanceName(const CIMConstClass& cimClass) const`**

  getInstnaceName - Get the instance name of this instance. The class argument is used to determine which fields are keys. The instance name has this from:

  ```
  ClassName.key1=value1,...,keyN=valueN
  ```

  The instance name is in standard form (the class name and key name is all lowercase; the keys-value pairs appear in sorted order by key name).

This class has no child classes.

Friends:

class CIMConstInstance

Alphabetic index *HTML hierarchy of classes* *or* *Java*

In file ..\..\src\Pegasus\Common\CIMMethod.h:

class PEGASUS_COMMON_LINKAGE CIMMethod

Class CIMMethod - This class defines the operations associated with manipulation of the Pegasus implementation of the CIM CIMMethod.

Documentation

Class CIMMethod - This class defines the operations associated with manipulation of the Pegasus implementation of the CIM CIMMethod. Within this class, methods are provides for creation, deletion, and manipulation of method declarations.

// ATTN: remove the classOrigin and propagated parameters.

Inheritance:

Public Methods

	CIMMethod () *Creates and instantiates a CIM method.*
	CIMMethod (const CIMMethod& x) *Creates and instantiates a CIM method from another method instance*
CIMMethod&	**operator=** (const CIMMethod& x) *Assignment operator*
	CIMMethod (const String& name, CIMType type, const String& classOrigin = String(), Boolean propagated = false) *Creates a CIM method with the specified name, type, and classOrigin*
	~CIMMethod () *Desctructor.*
const String&	**getName** () const *CIMMethod getName - Gets the name of the method*
void	**setName** (const String& name) *CIMMethod setName - Set the method name*
CIMType	**getType** () const *CIMMethod getType - gets the method type*
void	**setType** (CIMType type) *CIMMethod setType - Sets the method type to the specified CIM method type as defined in CIMType /Ref{TYPE}*

const String&	**getClassOrigin** () const *CIMMethod getClassOrigin - Returns the class in which this method was defined.*
void	**setClassOrigin** (const String& classOrigin) *CIMMethod setClassOrigin - ATTN:*
Boolean	**getPropagated** () const *method getPropagated - ATTN:*
void	**setPropagated** (Boolean propagated) *method setPropagated - ATTN:*
CIMMethod&	**addQualifier** (const CIMQualifier& x) *CIMMethod addQualifier - @parm CIMQualifier to add*
Uint32	**findQualifier** (const String& name) *CIMMethod findQualifier - returns the position of the qualifier with the given name.*
CIMQualifier	**getQualifier** (Uint32 pos) *CIMMethod getQualifier - Gets the CIMQualifier defined by the index input as a parameter.*
Uint32	**getQualifierCount** () const *CIMMethod getQualifierCount - Returns the number of Qualifiers attached to this method.*
CIMMethod&	**addParameter** (const CIMParameter& x) *CIMMethod addParameter - Adds the parameter defined by the input to the CIMMethod*
Uint32	**findParameter** (const String& name) *CIMMethod findParameter - Finds the parameter whose name is given by the name parameter.*
CIMParameter	**getParameter** (Uint32 pos) *CIMMethod getParameter - ATTN:*
CIMConstParameter	**getParameter** (Uint32 pos) const *CIMMethod getParameter - Gets the parameter defined by the index input as a parameter.*
Uint32	**getParameterCount** () const *CIMMethod getParameterCount - Gets the count of the numbeer of Parameters attached to the CIMMethod.*
void	**resolve** (DeclContext* declContext, const String& nameSpace, const CIMConstMethod& method) *method resolve - ATTN:*
void	**resolve** (DeclContext* declContext, const String& nameSpace) *CIMMethod resolve*
	operator int () const *Returns zero if CIMMethod refers to a null pointer*

void	**toXml** (Array<Sint8>& out) const *method toXML - placing XML encoding of this object into out arguemnt.*
void	**print** (std::ostream &o=std::cout) const *method print - prints this method (in CIM encoded form).*
Boolean	**identical** (const CIMConstMethod& x) const *CIMMethod identical - Returns true if this method is identical to the one given by the argument x*
CIMMethod	**clone** () const *CIMMethod clone - makes a distinct replica of this method*

- **`CIMMethod()`**

  Creates and instantiates a CIM method.

- **`CIMMethod(const CIMMethod& x)`**

  Creates and instantiates a CIM method from another method instance

  Returns:

  pointer to the new method instance

- **`CIMMethod& operator=(const CIMMethod& x)`**

  Assignment operator

- **`CIMMethod( const String& name, CIMType type, const String& classOrigin = String(), Boolean propagated = false)`**

  Creates a CIM method with the specified name, type, and classOrigin

  Parameters:

  name - for the method
  type - ATTN
  classOrigin -
  propagated -

  Returns:

  Throws IllegalName if name argument not legal CIM identifier.

- **`~CIMMethod()`**

  Desctructor.

- **`const String& getName() const`**

  CIMMethod getName - Gets the name of the method

  Returns:

  String with the name of the method

- **`void setName(const String& name)`**

  CIMMethod setName - Set the method name

  Throws:

IllegalName if name argument not legal CIM identifier.

Parameters:

name -

- **CIMType getType() const**

  CIMMethod getType - gets the method type

  Returns:

  The CIM method type for this method.

- **void setType(CIMType type)**

  CIMMethod setType - Sets the method type to the specified CIM method type as defined in CIMType /Ref{TYPE}

- **const String& getClassOrigin() const**

  CIMMethod getClassOrigin - Returns the class in which this method was defined.

  Returns:

  ATTN:

- **void setClassOrigin(const String& classOrigin)**

  CIMMethod setClassOrigin - ATTN:

- **Boolean getPropagated() const**

  method getPropagated - ATTN:

- **void setPropagated(Boolean propagated)**

  method setPropagated - ATTN:

- **CIMMethod& addQualifier(const CIMQualifier& x)**

  CIMMethod addQualifier - @parm CIMQualifier to add

  Throws:

  AlreadyExists exception

  Parameters:

  - CIMQualifier to be added

  Returns:

  Throws AlreadyExists excetpion if the qualifier already exists in the method

- **Uint32 findQualifier(const String& name)**

  CIMMethod findQualifier - returns the position of the qualifier with the given name.

  Parameters:

  name - - name of qualifier to be found.

  Returns:

  index of the parameter if found; otherwise Uint32(-1).

- **CIMQualifier getQualifier(Uint32 pos)**

  CIMMethod getQualifier - Gets the CIMQualifier defined by the index input as a parameter.

  Throws:

  OutOfBounds exception if the index is outside the range of parameters available from the CIMMethod.

  Parameters:

  Index - of the qualifier requested.

  Returns:

  CIMQualifier object or exception

- **Uint32 getQualifierCount() const**

  CIMMethod getQualifierCount - Returns the number of Qualifiers attached to this method.

  Returns:

  integer representing number of Qualifiers.

- **CIMMethod& addParameter(const CIMParameter& x)**

  CIMMethod addParameter - Adds the parameter defined by the input to the CIMMethod

- **Uint32 findParameter(const String& name)**

  CIMMethod findParameter - Finds the parameter whose name is given by the name parameter.

  Parameters:

  name - - name of parameter to be found.

  Returns:

  index of the parameter if found; otherwise Uint32(-1).

- **CIMParameter getParameter(Uint32 pos)**

  CIMMethod getParameter - ATTN:

- **CIMConstParameter getParameter(Uint32 pos) const**

  CIMMethod getParameter - Gets the parameter defined by the index input as a parameter.

  Parameters:

  index - for the parameter to be returned.

  Returns:

  CIMParameter requested. @Exception OutOfBounds exception is thrown if the index is outside the range of available parameters

- **Uint32 getParameterCount() const**

  CIMMethod getParameterCount - Gets the count of the numbeer of Parameters attached to the CIMMethod. @retrun - count of the number of parameters attached to the CIMMethod.

- **void resolve(DeclContext* declContext, const String& nameSpace, const CIMConstMethod& method)**

method resolve - ATTN:

- **`void resolve( DeclContext* declContext, const String& nameSpace)`**

  CIMMethod resolve

- **`operator int() const`**

  Returns zero if CIMMethod refers to a null pointer

- **`void toXml(Array<Sint8>& out) const`**

  method toXML - placing XML encoding of this object into out arguemnt.

- **`void print(std::ostream &o=std::cout) const`**

  method print - prints this method (in CIM encoded form).

- **`Boolean identical(const CIMConstMethod& x) const`**

  CIMMethod identical - Returns true if this method is identical to the one given by the argument x

- **`CIMMethod clone() const`**

  CIMMethod clone - makes a distinct replica of this method

This class has no child classes.

Friends:

class CIMConstMethod
class CIMClassRep

Alphabetic index *HTML hierarchy of classes* *or* *Java*

In file ..\..\src\Pegasus\Common\CIMName.h:

class PEGASUS_COMMON_LINKAGE CIMName

The name class defines static methods for handling CIM names.

Documentation

The name class defines static methods for handling CIM names. The names of classes, properties, qualifiers, and methods are all CIM names. A CIM name must match the following regular expression:

```
[A-Z-a-z_][A-Za-z_0-9]*
```

Notice that the definition of a name is the same as C, C++, and Java.

This class cannot be instantiated (since its only constructor is private).

Inheritance:

Public Methods

static Boolean	**legal** (const Char16* name) *CIMMethod legal - Determine if the name string input is legal as defnined in the CIMName class definition ATTN: Define what is legal*
static Boolean	**legal** (const String& name) *CIMMethod legal - Determine if the name string input is legal as defnined in the CIMName class definition*
static Boolean	**equal** (const String& name1, const String& name2) *CIMMethod equal - Compares two names.*

static Boolean legal(const Char16* name)

CIMMethod legal - Determine if the name string input is legal as defnined in the CIMName class definition ATTN: Define what is legal

Parameters:

- - String to test

Returns:

Returns true if the given name is legal. Throws NullPointer exception if name argument is null.

- **`static Boolean legal(const String& name)`**

  CIMMethod legal - Determine if the name string input is legal as defnined in the CIMName class definition

  Parameters:

  - - String to test

  Returns:

  Returns true if the given name is legal. Throws NullPointer exception if name argument is null.

- **`static Boolean equal(const String& name1, const String& name2)`**

  CIMMethod equal - Compares two names.

  Returns:

  Return true if the two names are equal. CIM names are case insensitive and so it this method.

This class has no child classes.

Alphabetic index HTML hierarchy of classes or Java

In file ..\..\src\Pegasus\Common\CIMProperty.h:

class PEGASUS_COMMON_LINKAGE CIMProperty

CIMProperty Class - ATTN:

Documentation

CIMProperty Class - ATTN:

Inheritance:

Public Methods

	CIMProperty () *CIMMethod CIMProperty*
	CIMProperty (const CIMProperty& x) *CIMMethod CIMProperty*
	CIMProperty (const String& name, const CIMValue& value, Uint32 arraySize = 0, const String& referenceClassName = String::EMPTY, const String& classOrigin = String(), Boolean propagated = false) *CIMMethod CIMProperty*
CIMProperty&	**operator=** (const CIMProperty& x) *CIMMethod*
const String&	**getName** () const *CIMMethod*
void	**setName** (const String& name) *CIMMethod setName - Set the property name.*
const CIMValue&	**getValue** () const *CIMMethod getValue - ATTN:*
void	**setValue** (const CIMValue& value) *CIMMethod setValue - ATTN*
Uint32	**getArraySize** () const *CIMMethod getArraySize - ATTN:*
const String&	**getReferenceClassName** () const *CIMMethod getReferenceClassName - ATTN:*
const String&	**getClassOrigin** () const *CIMMethod getClassOrigin - ATTN*

void	**setClassOrigin** (const String& classOrigin) *CIMMethod setClassOrigin*
Boolean	**getPropagated** () const *CIMMethod getPropagated - ATTN*
void	**setPropagated** (Boolean propagated) *CIMMethod setProgagated - ATTN*
CIMProperty&	**addQualifier** (const CIMQualifier& x) *CIMMethod addQualifier - ATTN Throws AlreadyExists*
Uint32	**findQualifier** (const String& name) *CIMMethod findQualifier - ATTN*
CIMQualifier	**getQualifier** (Uint32 pos) *CIMMethod getQualifier - ATTN*
CIMConstQualifier	**getQualifier** (Uint32 pos) const *CIMMethod getQualifier - ATTN*
Uint32	**getQualifierCount** () const *CIMMethod getQualifier - ATTN*
void	**resolve** (DeclContext* declContext, const String& nameSpace, Boolean isInstancePart, const CIMConstProperty& property) *CIMMethod resolve*
void	**resolve** (DeclContext* declContext, const String& nameSpace, Boolean isInstancePart) *CIMMethod resolve - ATTN*
	operator int () const *ATTN*
void	**toXml** (Array<Sint8>& out) const *mthod toXML*
void	**print** (std::ostream &o=std::cout) const *mthod print -ATTN*
Boolean	**identical** (const CIMConstProperty& x) const *CIMMethod identical - ATTN*
CIMProperty	**clone** () const *CIMMethod clone - ATTN*

- **`CIMProperty()`**

  CIMMethod CIMProperty

- **`CIMProperty(const CIMProperty& x)`**

  CIMMethod CIMProperty

- **`CIMProperty( const String& name, const CIMValue& value, Uint32 arraySize = 0, const String& referenceClassName = String::EMPTY, const String& classOrigin = String(), Boolean propagated = false)`**

CIMMethod CIMProperty

Returns:

Throws IllegalName if name argument not legal CIM identifier.

- **CIMProperty& operator=(const CIMProperty& x)**

  CIMMethod

- **const String& getName() const**

  CIMMethod

- **void setName(const String& name)**

  CIMMethod setName - Set the property name. Throws IllegalName if name argument not legal CIM identifier.

- **const CIMValue& getValue() const**

  CIMMethod getValue - ATTN:

- **void setValue(const CIMValue& value)**

  CIMMethod setValue - ATTN

- **Uint32 getArraySize() const**

  CIMMethod getArraySize - ATTN:

- **const String& getReferenceClassName() const**

  CIMMethod getReferenceClassName - ATTN:

- **const String& getClassOrigin() const**

  CIMMethod getClassOrigin - ATTN

- **void setClassOrigin(const String& classOrigin)**

  CIMMethod setClassOrigin

- **Boolean getPropagated() const**

  CIMMethod getPropagated - ATTN

- **void setPropagated(Boolean propagated)**

  CIMMethod setProgagated - ATTN

- **CIMProperty& addQualifier(const CIMQualifier& x)**

  CIMMethod addQualifier - ATTN Throws AlreadyExists

- **Uint32 findQualifier(const String& name)**

  CIMMethod findQualifier - ATTN

- **CIMQualifier getQualifier(Uint32 pos)**

  CIMMethod getQualifier - ATTN

- **CIMConstQualifier getQualifier(Uint32 pos) const**

CIMMethod getQualifier - ATTN

- **`Uint32 getQualifierCount() const`**

  CIMMethod getQualifier - ATTN

- **`void resolve( DeclContext* declContext, const String& nameSpace, Boolean isInstancePart, const CIMConstProperty& property)`**

  CIMMethod resolve

- **`void resolve( DeclContext* declContext, const String& nameSpace, Boolean isInstancePart)`**

  CIMMethod resolve - ATTN

- **`operator int() const`**

  ATTN

- **`void toXml(Array<Sint8>& out) const`**

  mthod toXML

- **`void print(std::ostream &o=std::cout) const`**

  mthod print -ATTN

- **`Boolean identical(const CIMConstProperty& x) const`**

  CIMMethod identical - ATTN

- **`CIMProperty clone() const`**

  CIMMethod clone - ATTN

This class has no child classes.

Friends:

class CIMConstProperty
class CIMClassRep
class CIMInstanceRep

Alphabetic index *HTML hierarchy of classes* *or* *Java*

In file ..\..\src\Pegasus\Common\CIMQualifier.h:

class PEGASUS_COMMON_LINKAGE CIMQualifier

Class CIMQualifier - This class defines the Pegasus implementation of the CIM CIMQualifier QUALIFIER

Documentation

Class CIMQualifier - This class defines the Pegasus implementation of the CIM CIMQualifier QUALIFIER

Inheritance:

Public Methods

	CIMQualifier () *Constructor instantiates a CIM qualifier with empty name value fieldsConstructor*
	CIMQualifier (const CIMQualifier& x) *Constructor - instantiates a CIM qualifier object from another qualifier object.*
	CIMQualifier (const String& name, const CIMValue& value, Uint32 flavor = CIMFlavor::DEFAULTS, Boolean propagated = false) *Constructor - Instantiates a CIM qualifier object with the parameters defined on input.*
	~CIMQualifier () *destructor*
CIMQualifier&	**operator=** (const CIMQualifier& x) *operator*
const String&	**getName** () const *CIMMethod*
void	**setName** (const String& name) *CIMMethod Throws IllegalName if name argument not legal CIM identifier*
CIMType	**getType** () const *CIMMethod*
Boolean	**isArray** () const *CIMMethod*
const CIMValue&	**getValue** () const *CIMMethod*

void	**setValue** (const CIMValue& value) *CIMMethod*
Uint32	**getFlavor** () const *CIMMethod*
const Uint32	**getPropagated** () const *CIMMethod*
void	**setPropagated** (Boolean propagated) *CIMMethod*
	operator int () const *CIMMethod*
void	**toXml** (Array<Sint8>& out) const *CIMMethod*
void	**print** (std::ostream &o=std::cout) const *CIMMethod*
Boolean	**identical** (const CIMConstQualifier& x) const *CIMMethod*
CIMQualifier	**clone** () const *CIMMethod*

- **`CIMQualifier()`**

  Constructor instantiates a CIM qualifier with empty name value fieldsConstructor

  Returns:

  instantiated empty qualifier object

- **`CIMQualifier(const CIMQualifier& x)`**

  Constructor - instantiates a CIM qualifier object from another qualifier object.

  Parameters:

  CIM - CIMQualifier object ATTN: What is differenc from clone?

  Returns:

  - Instantiated qualifier object

- **`CIMQualifier( const String& name, const CIMValue& value, Uint32 flavor = CIMFlavor::DEFAULTS, Boolean propagated = false)`**

  Constructor - Instantiates a CIM qualifier object with the parameters defined on input.

  Throws:

  Throws IllegalName if name argument not legal CIM identifier.

  Parameters:

  - String representing CIMName for the new qualifier

  value -
  flavor - - ATTN:
  propoagated - - ATTN:

Returns:

-Returns the instantiated qualifier object or throws an exception if the name argument is illegal

- **~CIMQualifier()**

  destructor

- **CIMQualifier& operator=(const CIMQualifier& x)**

  operator

- **const String& getName() const**

  CIMMethod

- **void setName(const String& name)**

  CIMMethod Throws IllegalName if name argument not legal CIM identifier

- **CIMType getType() const**

  CIMMethod

- **Boolean isArray() const**

  CIMMethod

- **const CIMValue& getValue() const**

  CIMMethod

- **void setValue(const CIMValue& value)**

  CIMMethod

- **Uint32 getFlavor() const**

  CIMMethod

- **const Uint32 getPropagated() const**

  CIMMethod

- **void setPropagated(Boolean propagated)**

  CIMMethod

- **operator int() const**

  CIMMethod

- **void toXml(Array<Sint8>& out) const**

  CIMMethod

- **void print(std::ostream &o=std::cout) const**

  CIMMethod

- **Boolean identical(const CIMConstQualifier& x) const**

  CIMMethod

CIMQualifier clone() const

CIMMethod

This class has no child classes.

Friends:

class CIMConstQualifier
class CIMClassRep

Alphabetic index *HTML hierarchy of classes* *or* *Java*

In file ..\..\src\Pegasus\Common\CIMQualifierDecl.h:

class PEGASUS_COMMON_LINKAGE CIMQualifierDecl

Class CIMQualifierDecl

Documentation

Class CIMQualifierDecl

NOTE: Clarify difference between qualifier and qualiferdeclaration ATTN: Important work required here.

Inheritance:

Public Methods

	CIMQualifierDecl () *Constructor - ATTN:*
	CIMQualifierDecl (const CIMQualifierDecl& x) *Constructor - ATTN:*
	CIMQualifierDecl (const String& name, const CIMValue& value, Uint32 scope, Uint32 flavor = CIMFlavor::DEFAULTS, Uint32 arraySize = 0) *Constructor Throws IllegalName if name argument not legal CIM identifier.*
	~CIMQualifierDecl () *Destructor*
CIMQualifierDecl&	**operator=** (const CIMQualifierDecl& x) *Operator*
const String&	**getName** () const *CIMMethod ATTN:*
void	**setName** (const String& name) *CIMMethod ATTN:*
CIMType	**getType** () const *CIMMethod ATTN:*
Boolean	**isArray** () const *CIMMethod ATTN:*
const CIMValue&	**getValue** () const *CIMMethod*

void	**setValue** (const CIMValue& value) *CIMMethod*
Uint32	**getScope** () const *CIMMethod*
Uint32	**getFlavor** () const *CIMMethod*
Uint32	**getArraySize** () const *CIMMethod*
	operator int () const *CIMMethod*
void	**toXml** (Array<Sint8>& out) const *CIMMethod*
void	**print** (std::ostream &o=std::cout) const *CIMMethod*
Boolean	**identical** (const CIMConstQualifierDecl& x) const *CIMMethod*
CIMQualifierDecl	**clone** () const *CIMMethod*

- **`CIMQualifierDecl()`**

  Constructor - ATTN:

- **`CIMQualifierDecl(const CIMQualifierDecl& x)`**

  Constructor - ATTN:

- **`CIMQualifierDecl( const String& name, const CIMValue& value, Uint32 scope, Uint32 flavor = CIMFlavor::DEFAULTS, Uint32 arraySize = 0)`**

  Constructor Throws IllegalName if name argument not legal CIM identifier. ATTN:

- **`~CIMQualifierDecl()`**

  Destructor

- **`CIMQualifierDecl& operator=(const CIMQualifierDecl& x)`**

  Operator

- **`const String& getName() const`**

  CIMMethod ATTN:

- **`void setName(const String& name)`**

  CIMMethod ATTN:

- **`CIMType getType() const`**

  CIMMethod ATTN:

- **`Boolean isArray() const`**

CIMMethod ATTN:

- **const CIMValue& getValue() const**

  CIMMethod

- **void setValue(const CIMValue& value)**

  CIMMethod

- **Uint32 getScope() const**

  CIMMethod

- **Uint32 getFlavor() const**

  CIMMethod

- **Uint32 getArraySize() const**

  CIMMethod

- **operator int() const**

  CIMMethod

- **void toXml(Array<Sint8>& out) const**

  CIMMethod

- **void print(std::ostream &o=std::cout) const**

  CIMMethod

- **Boolean identical(const CIMConstQualifierDecl& x) const**

  CIMMethod

- **CIMQualifierDecl clone() const**

  CIMMethod

This class has no child classes.

Friends:

class CIMConstQualifierDecl
class CIMClassRep

Alphabetic index HTML hierarchy of classes or Java

In file ..\..\src\Pegasus\Common\CIMReference.h:

class PEGASUS_COMMON_LINKAGE CIMReference

The CIMReference class represents the value of a reference.

Documentation

The CIMReference class represents the value of a reference. A reference is one of property types which an association may contain. Consider the following MOF for example:

```
[Association]
class MyAssociations
{
    MyClass ref from;
    MyClass ref to;
};
```

The value of the from and to properties are internally represented using the CIMReference class.

CIM references are used to uniquely identify a CIM class or CIM instance objects. CIMReference objects contain the following parts:

- Host - name of host whose repository contains the object
- NameSpace - the namespace which contains the object
- ClassName - name of objects class
- KeyBindings key/value pairs which uniquely identify an instance

CIM references may also be expressed as simple strings (as opposed to being represented by the CIMReference class). This string is known as the "Object Name". An object name has the following form:

```
<namespace-path>:<model-path>
```

The namespace-path is implementation dependent and has the following form in Pegasus:

```
//<hostname>>/<namespace>>
```

For example, suppose there is a host named "atp" with a CIM Server listening on port 9999 which has a CIM repository with a namespace called "root/cimv25". Then the namespace-path is given as:

```
//atp:9999/root/cimv25
```

As for the model-path mentioned above, its form is defined by the CIM Standard (more is defined by the "XML Mapping Specification v2.0.0" specification) as follows:

```
<Qualifyingclass>.<key-1>=<value-1>[,<key-n>=<value-n>]*
```

For example:

```
TennisPlayer.first="Patrick",last="Rafter"
```

This of course presupposes the existence of a class called "MyClass" that has key properties named "first" and "last". For example, here is what the MOF might look like:

```
class TennisPlayer : Person
{
    [key] string first;
    [key] string last;
};
```

All keys must be present in the model path.

Now the namespace-type and model-path are combined in the following string object name.

//atp:9999/root/cimv25:TennisPlayer.first="Patrick",last="Rafter"

Now suppose we wish to create a CIMReference from this above string. There are two constructors provided: one which takes the above string and the other that takes the constituent elements. Here are the signature of the two constructors:

```
CIMReference(const String& objectName);


CIMReference(
    const String& host,
    const String& nameSpace,
    const String& className,
    const KeyBindingArray& keyBindings);
```

Following our example, the above object name may be used to initialize a CIMReference like this:

```
    CIMReference ref =
        "//atp:9999/root/cimv25:TennisPlayer.first="Patrick",last="Rafter";
```

A CIMReference may also be initialized using the constituent elements of the object name (sometimes the object name is not available as a string: this is the case with CIM XML encodings). The arguments shown in that constructor above correspond elements of the object name in the following way:

- host = "atp:9999"
- nameSpace = "root/cimv25"
- className = "TennisPlayer"
- keyBindings = "first=\"Patrick\",last=\"Rafter\""

Note that the host and nameSpace argument may be empty since object names need not necessarily include a namespace path according to the standard.

Of course the key bindings must be built up by appending KeyBinding objects to a KeyBindingArray like this:

```
KeyBindingArray keyBindings;
keyBindings.append(KeyBinding("first", "Patrick", KeyBinding::STRING));
keyBindings.append(KeyBinding("last", "Rafter", KeyBinding::STRING));
```

The only key values that are supported are:

- KeyBinding::BOOLEAN
- KeyBinding::STRING
- KeyBinding::NUMERIC

This limitation is imposed by the "XML Mapping Specification v2.0.0" specification. The CIM types are encoded as one of these three in the following way:

```
boolean - BOOLEAN (the value must be "true" or "false")
uint8 - NUMERIC
sint8 - NUMERIC
uint16 - NUMERIC
sint16 - NUMERIC
uint32 - NUMERIC
sint32 - NUMERIC
uint64 - NUMERIC
sint64 - NUMERIC
char16 - NUMERIC
string - STRING
datetime - STRING
```

Notice that real32 and real64 are missing. Properties of these types cannot be used as keys.

Notice that the keys in the object name may appear in any order. That is the following object names refer to the same object:

```
TennisPlayer.first="Patrick",last="Rafter"
TennisPlayer.last="Rafter",first="Patrick"
```

And since CIM is not case sensitive, the following refer to the same object:

```
TennisPlayer.first="Patrick",last="Rafter"
tennisplayer.FIRST="Patrick",Last="Rafter"
```

Therefore, the CIMReferences::operator==() would return true for the last two examples.

The CIM standard leaves it an open question whether model paths may have spaces around delimiters (like '.', '=', and ','). We assume they cannot. So the following is an invalid model path:

```
TennisPlayer . first = "Patrick", last="Rafter"
```

We require that the '.', '=', and ',' have no spaces around them.

For reasons of efficiency, the key bindings are internally sorted during initialization. This allows the key bindings to be compared more easily. This means that when the string is converted back to string (by calling toString()) that the keys may have been rearranged.

There are two forms an object name can take:

```
<namespace-path>:<model-path>
```

```
<model-path>
```

In other words, the namespace-path is optional. Here is an example of each:

```
//atp:9999/root/cimv25:TennisPlayer.first="Patrick",last="Rafter"
TennisPlayer.first="Patrick",last="Rafter"
```

If it begins with "//" then we assume the namespace-path is present and process it that way.

It should also be noted that an object name may refer to an instance or a class. Here is an example of each:

```
TennisPlayer.first="Patrick",last="Rafter"
TennisPlayer
```

In the second case--when it refers to a class--the key bindings are omitted.

Inheritance:

Public Methods

	CIMReference () *Default constructor.*
	CIMReference (const CIMReference& x) *Copy constructor.*
	CIMReference (const String& objectName) *Initializes this object from a CIM object name.*
	CIMReference (const char* objectName) *Initializes this object from a CIM object name (char* version).*
	CIMReference (const String& host, const String& nameSpace, const String& className, const KeyBindingArray& keyBindings = KeyBindingArray()) *Constructs a CIMReference from constituent elements.*
virtual	**~CIMReference** () *Destructor*
CIMReference&	**operator=** (const CIMReference& x) *Assignment operator*
void	**clear** () *Clears out the internal fields of this object making it an empty (or unitialized reference).*
void	**set** (const String& host, const String& nameSpace, const String& className, const KeyBindingArray& keyBindings = KeyBindingArray()) *Sets this reference from constituent elements.*
void	**set** (const String& objectName) *Set the reference from an object name .*
CIMReference&	**operator=** (const String& objectName) *Same as set() above except that it is an assignment operator*

CIMReference&	**operator=** (const char* objectName) *Same as set() above except that it is an assignment operator*
const String&	**getHost** () const *Accessor.*
void	**setHost** (const String& host) *Modifier.*
const String&	**getNameSpace** () const *Accessor*
void	**setNameSpace** (const String& nameSpace) *Sets the namespace component.*
const String&	**getClassName** () const *Accessor.*
void	**setClassName** (const String& className) *Sets the class name component to the following string.*
const Array<KeyBinding> &	**getKeyBindings** () const *Accessor.*
void	**setKeyBindings** (const Array<KeyBinding>& keyBindings) *Modifier.*
String	**toString** () const *Returns the object name represented by this reference.*
Boolean	**identical** (const CIMReference& x) const *Returns true if this reference is identical to the one given by the x argument*
void	**toXml** (Array<Sint8>& out) const *Encodes this object as XML.*
void	**print** (std::ostream& os = std::cout) const *Prints the XML encoding of this objet*
Uint32	**makeHashCode** () const *Generates hash code for the given reference.*

- **`CIMReference()`**

  Default constructor.

- **`CIMReference(const CIMReference& x)`**

  Copy constructor.

- **`CIMReference(const String& objectName)`**

  Initializes this object from a CIM object name.

- **`CIMReference(const char* objectName)`**

  Initializes this object from a CIM object name (char* version).

- **`CIMReference( const String& host, const String& nameSpace, const String& className, const KeyBindingArray& keyBindings = KeyBindingArray())`**

  Constructs a CIMReference from constituent elements.

  Parameters:

  host - - name of host (e.g., "nemesis-8888").
  nameSpace - - namespace (e.g., "root/cimv20").
  className - - name of a class (e.g., "MyClass").
  keyBindings - - an array of KeyBinding objects.

- **`virtual ~CIMReference()`**

Destructor

- **CIMReference& operator=(const CIMReference& x)**

  Assignment operator

- **void clear()**

  Clears out the internal fields of this object making it an empty (or unitialized reference). The effect is the same as if the object was initialized with the default constructor.

- **void set(const String& host, const String& nameSpace, const String& className, const KeyBindingArray& keyBindings = KeyBindingArray())**

  Sets this reference from constituent elements. The effect is same as if the object was initialized using the constructor above that has the same arguments.

- **void set(const String& objectName)**

  Set the reference from an object name .

- **CIMReference& operator=(const String& objectName)**

  Same as set() above except that it is an assignment operator

- **CIMReference& operator=(const char* objectName)**

  Same as set() above except that it is an assignment operator

- **const String& getHost() const**

  Accessor.

- **void setHost(const String& host)**

  Modifier.

- **const String& getNameSpace() const**

  Accessor

- **void setNameSpace(const String& nameSpace)**

  Sets the namespace component.

  Throws:

  Throws IllegalName if form of the namespace is illegal.

  Parameters:

  - String representing the Namespace

- **const String& getClassName() const**

  Accessor.

- **void setClassName(const String& className)**

  Sets the class name component to the following string.

  Throws:

  Throws IllegalName if form of className is illegal.

- **const Array<KeyBinding> & getKeyBindings() const**

  Accessor.

- **void setKeyBindings(const Array<KeyBinding>& keyBindings)**

  Modifier.

- **String toString() const**

  Returns the object name represented by this reference.

- **`Boolean identical(const CIMReference& x) const`**

  Returns true if this reference is identical to the one given by the x argument

- **`void toXml(Array<Sint8>& out) const`**

  Encodes this object as XML.

  Parameters:

  out - - argument to place resulting XML into.

- **`void print(std::ostream& os = std::cout) const`**

  Prints the XML encoding of this objet

- **`Uint32 makeHashCode() const`**

  Generates hash code for the given reference. Two identical references generate the same hash code (despite any subtle differences such as the case of the classname and key names as well as the order of the keys).

This class has no child classes.

Friends:

XmlWriter

Alphabetic index HTML hierarchy of classes or Java

In file ..\..\src\Pegasus\Common\CIMType.h:

class PEGASUS_COMMON_LINKAGE CIMType

The CIMType Class defines the CIMType enumeration which introduces symbolic constants for the CIM data types.

Documentation

The CIMType Class defines the CIMType enumeration which introduces symbolic constants for the CIM data types.

The table below shows each CIM type, its symbolic constant, and its representation type.

CIM CIMType	Constant	C++ CIMType	
boolean	CIMType::BOOLEAN		Boolean
uint8	CIMType::UINT8	Uint8	
sint8	CIMType::SINT8	Sint8	
uint16	CIMType::UINT16	Uint16	
sint16	CIMType::SINT16	Sint16	
uint32	CIMType::UINT32	Uint32	
sint32	CIMType::SINT32	Sint32	
uint64	CIMType::UINT64	Sint64	
sint64	CIMType::SINT64	Sint64	
real32	CIMType::REAL32	Real32	
real64	CIMType::REAL64	Real64	
char16	CIMType::CHAR16	Char16	
string	CIMType::STRING	String	
datetime	CIMType::DATETIME		CIMDateTime
reference	CIMType::REFERENCE		CIMReference

Inheritance:

This class has no child classes.

Alphabetic index HTML hierarchy of classes or Java

In file ..\..\src\Pegasus\Common\CIMValue.h:

class PEGASUS_COMMON_LINKAGE CIMValue

The CIMValue class represents a value of any of the CIM data types (see CIMTypeh for a list of valid CIM data types).

Documentation

The CIMValue class represents a value of any of the CIM data types (see CIMTypeh for a list of valid CIM data types). This class encapsulates a union which holds the current value. The class also has a type field indicating the type of that value.

Inheritance:

Public Methods

	CIMValue () *Constructor*
	CIMValue (Boolean x) *Constructor*
	CIMValue (Uint8 x) *Constructor*
	CIMValue (Sint8 x) *Constructor*
	CIMValue (Uint16 x) *Constructor*
	CIMValue (Sint16 x) *Constructor*
	CIMValue (Uint32 x) *Constructor*
	CIMValue (Sint32 x) *Constructor*
	CIMValue (Uint64 x) *Constructor*
	CIMValue (Sint64 x) *Constructor*

	CIMValue (Real32 x) *Constructor*
	CIMValue (Real64 x) *Constructor*
	CIMValue (const Char16& x) *Constructor*
	CIMValue (const String& x) *Constructor*
	CIMValue (const char* x) *Constructor*
	CIMValue (const CIMDateTime& x) *Constructor*
	CIMValue (const CIMReference& x) *Constructor*
	CIMValue (const Array<Boolean>& x) *Constructor*
	CIMValue (const Array<Uint8>& x) *Constructor*
	CIMValue (const Array<Sint8>& x) *Constructor*
	CIMValue (const Array<Uint16>& x) *Constructor*
	CIMValue (const Array<Sint16>& x) *Constructor*
	CIMValue (const Array<Uint32>& x) *Constructor*
	CIMValue (const Array<Sint32>& x) *Constructor*
	CIMValue (const Array<Uint64>& x) *Constructor*
	CIMValue (const Array<Sint64>& x) *Constructor*
	CIMValue (const Array<Real32>& x) *Constructor*
	CIMValue (const Array<Real64>& x) *Constructor*
	CIMValue (const Array<Char16>& x) *Constructor*

	CIMValue (const Array<String>& x) *Constructor*
	CIMValue (const Array<CIMDateTime>& x) *Constructor*
	CIMValue (const CIMValue& x) *Constructor*
	~CIMValue () *Destrustructor*
CIMValue&	**operator=** (const CIMValue& x) *Operator =*
void	**assign** (const CIMValue& x) *CIMMethod assign*
void	**clear** () *CIMMethod clear*
Boolean	**typeCompatible** (const CIMValue& x) const *CIMMethod typeCompatible - Compares the types of two values.*
Boolean	**isArray** () const *CIMMethod isArray - Determines if the value is an array*
Uint32	**getArraySize** () const *CIMMethod getArraySize*
void	**setNullValue** (CIMType type, Boolean isArray, Uint32 arraySize = 0) *method setNullvalue - ATTN:*
void	**set** (Boolean x) *method set - ATTN:*
void	**set** (Uint8 x) *CIMMethod Set*
void	**get** (Boolean& x) const *CIMMethod get - ATTN*
void	**toXml** (Array<Sint8>& out) const *CIMMethod toXML - ATTN*
void	**print** (std::ostream &o=std::cout) const *CIMMethod print - ATTN*
String	**toString** () const *CIMMethod toString - ATTN*

- **`CIMValue()`**

  Constructor

- **`CIMValue(Boolean x)`**

  Constructor

- **`CIMValue(Uint8 x)`**

Constructor

- **CIMValue(Sint8 x)**

  Constructor

- **CIMValue(Uint16 x)**

  Constructor

- **CIMValue(Sint16 x)**

  Constructor

- **CIMValue(Uint32 x)**

  Constructor

- **CIMValue(Sint32 x)**

  Constructor

- **CIMValue(Uint64 x)**

  Constructor

- **CIMValue(Sint64 x)**

  Constructor

- **CIMValue(Real32 x)**

  Constructor

- **CIMValue(Real64 x)**

  Constructor

- **CIMValue(const Char16& x)**

  Constructor

- **CIMValue(const String& x)**

  Constructor

- **CIMValue(const char* x)**

  Constructor

- **CIMValue(const CIMDateTime& x)**

  Constructor

- **CIMValue(const CIMReference& x)**

  Constructor

- **CIMValue(const Array<Boolean>& x)**

  Constructor

- **CIMValue(const Array<Uint8>& x)**

Constructor

- **CIMValue(const Array<Sint8>& x)**

  Constructor

- **CIMValue(const Array<Uint16>& x)**

  Constructor

- **CIMValue(const Array<Sint16>& x)**

  Constructor

- **CIMValue(const Array<Uint32>& x)**

  Constructor

- **CIMValue(const Array<Sint32>& x)**

  Constructor

- **CIMValue(const Array<Uint64>& x)**

  Constructor

- **CIMValue(const Array<Sint64>& x)**

  Constructor

- **CIMValue(const Array<Real32>& x)**

  Constructor

- **CIMValue(const Array<Real64>& x)**

  Constructor

- **CIMValue(const Array<Char16>& x)**

  Constructor

- **CIMValue(const Array<String>& x)**

  Constructor

- **CIMValue(const Array<CIMDateTime>& x)**

  Constructor

- **CIMValue(const CIMValue& x)**

  Constructor

- **~CIMValue()**

  Destrustructor

- **CIMValue& operator=(const CIMValue& x)**

  Operator =

- **void assign(const CIMValue& x)**

CIMMethod assign

- **void clear()**

  CIMMethod clear

- **Boolean typeCompatible(const CIMValue& x) const**

  CIMMethod typeCompatible - Compares the types of two values.

  Returns:

  true if compatible.

- **Boolean isArray() const**

  CIMMethod isArray - Determines if the value is an array

  Returns:

  TRUE if the value is an array

- **Uint32 getArraySize() const**

  CIMMethod getArraySize

  Returns:

  The number of entries in the array

- **void setNullValue(CIMType type, Boolean isArray, Uint32 arraySize = 0)**

  method setNullvalue - ATTN:

- **void set(Boolean x)**

  method set - ATTN:

- **void set(Uint8 x)**

  CIMMethod Set

- **void get(Boolean& x) const**

  CIMMethod get - ATTN

- **void toXml(Array<Sint8>& out) const**

  CIMMethod toXML - ATTN

- **void print(std::ostream &o=std::cout) const**

  CIMMethod print - ATTN

- **String toString() const**

  CIMMethod toString - ATTN

This class has no child classes.

Friends:

class CIMMethodRep
class CIMParameterRep

class CIMPropertyRep
class CIMQualifierRep
class CIMQualifierDeclRep
Boolean operator==(

Alphabetic index *HTML hierarchy of classes* *or* *Java*

Pegasus Documentation

In file ..\..\src\Pegasus\Common\Char16.h:

class PEGASUS_COMMON_LINKAGE Char16

The Char16 class represents a CIM sixteen bit character (char16).

Documentation

The Char16 class represents a CIM sixteen bit character (char16). This class is a trivial wrapper for a sixteen bit integer. It is used as the element type in the String class (used to represent the CIM string type). Ordinarily Uint16 could be used; however, a distinguishable type was needed for the purposes of function overloaded which occurs in the CIMValue class.

Inheritance:

Public Methods

	Char16 () *Constructor Char16*
	Char16 (Uint16 x) *Constructor Char16*
	Char16 (const Char16& x) *Constructor Char16*
Char16&	**operator=** (Uint16 x) *Constructor Char16*
Char16&	**operator=** (const Char16& x) *Constructor Char16*
	operator Uint16 () const *Implicit converter from Char16 to Uint16*
Uint16	**getCode** () const *Accessor for internal code member*

- **`Char16()`**

  Constructor Char16

- **`Char16(Uint16 x)`**

  Constructor Char16

- **`Char16(const Char16& x)`**

Constructor Char16

- **`Char16& operator=(Uint16 x)`**

  Constructor Char16

- **`Char16& operator=(const Char16& x)`**

  Constructor Char16

- **`operator Uint16() const`**

  Implicit converter from Char16 to Uint16

- **`Uint16 getCode() const`**

  Accessor for internal code member

This class has no child classes.

Alphabetic index *HTML hierarchy of classes* *or* *Java*

In file ..\..\src\Pegasus\Common\CIMReference.h:

class PEGASUS_COMMON_LINKAGE KeyBinding

The KeyBinding class associates a key name, value, and type.

Documentation

The KeyBinding class associates a key name, value, and type. It is used by the reference class to represent key bindings. See the CIMReference class to see how they are used.

Inheritance:

Public Methods

	KeyBinding () *Default constructor*
	KeyBinding (const KeyBinding& x) *Copy constructor*
	KeyBinding (const String& name, const String& value, Type type) *Construct a KeyBinding with a name, value, and type*
	~KeyBinding () *Destructor*
KeyBinding&	**operator=** (const KeyBinding& x) *Assignment operator*
const String&	**getName** () const *Accessor*
void	**setName** (const String& name) *Modifier*
const String&	**getValue** () const *Accessor*
void	**setValue** (const String& value) *Modifier*
Type	**getType** () const *Accessor*
void	**setType** (Type type) *Modifier*

static const char*	**typeToString** (Type type) *Converts the given type to one of the following: "boolean", "string", or "numeric"*

- **KeyBinding()**

  Default constructor

- **KeyBinding(const KeyBinding& x)**

  Copy constructor

- **KeyBinding(const String& name, const String& value, Type type)**

  Construct a KeyBinding with a name, value, and type

- **~KeyBinding()**

  Destructor

- **KeyBinding& operator=(const KeyBinding& x)**

  Assignment operator

- **const String& getName() const**

  Accessor

- **void setName(const String& name)**

  Modifier

- **const String& getValue() const**

  Accessor

- **void setValue(const String& value)**

  Modifier

- **Type getType() const**

  Accessor

- **void setType(Type type)**

  Modifier

- **static const char* typeToString(Type type)**

  Converts the given type to one of the following: "boolean", "string", or "numeric"

This class has no child classes.

Friends:

Boolean operator==(const KeyBinding& x, const KeyBinding& y)
CIMReference

Alphabetic index *HTML hierarchy of classes* *or* *Java*

In file ..\..\src\Pegasus\Common\Exception.h:

class PEGASUS_COMMON_LINKAGE AddedReferenceToClass

ATTN:

Documentation

ATTN:

Inheritance:

Inherited from Exception:

This class has no child classes.

Alphabetic index HTML hierarchy of classes or Java

In file ..\..\src\Pegasus\Common\Exception.h:

class PEGASUS_COMMON_LINKAGE AlreadyExists

ATTN:

Documentation

ATTN:

Inheritance:

Inherited from Exception:

This class has no child classes.

Alphabetic index HTML hierarchy of classes or Java

In file ..\..\src\Pegasus\Common\Exception.h:

class PEGASUS_COMMON_LINKAGE AssertionFailureException

Class AssertionFailureException This is an Exception class tied to the definiton of an assert named PEGASUS_ASSERT.

Documentation

Class AssertionFailureException This is an Exception class tied to the definiton of an assert named PEGASUS_ASSERT. This assertion can be included at any point in Pegasus code

---

Inheritance:

Inherited from Exception:

This class has no child classes.

Alphabetic index HTML hierarchy of classes or Java

In file ..\..\src\Pegasus\Common\Exception.h:

class PEGASUS_COMMON_LINKAGE BadQualifierOverride

ATTN:

Documentation

ATTN:

Inheritance:

Inherited from Exception:

This class has no child classes.

Alphabetic index HTML hierarchy of classes or Java

In file ..\..\src\Pegasus\Common\Exception.h:

class PEGASUS_COMMON_LINKAGE BadQualifierScope

ATTN:

Documentation

ATTN:

Inheritance:

Inherited from Exception:

This class has no child classes.

Alphabetic index HTML hierarchy of classes or Java

In file ..\..\src\Pegasus\Common\Exception.h:

class PEGASUS_COMMON_LINKAGE BadReference

ATTN:

Documentation

ATTN:

Inheritance:

Inherited from Exception:

This class has no child classes.

Alphabetic index HTML hierarchy of classes or Java

In file ..\..\src\Pegasus\Common\Exception.h:

class PEGASUS_COMMON_LINKAGE CIMException

The CIMException defines the CIM exceptions that are formally defined in the CIM Operations over HTTP specification.

Documentation

The CIMException defines the CIM exceptions that are formally defined in the CIM Operations over HTTP specification. @example

```
throw CIMException(CIMException::NOT_SUPPORTED);
```

Inheritance:

Inherited from Exception:

This class has no child classes.

Alphabetic index HTML hierarchy of classes or Java

In file ..\..\src\Pegasus\Common\Exception.h:

class PEGASUS_COMMON_LINKAGE CannotCreateDirectory

ATTN:

Documentation

ATTN:

Inheritance:

Inherited from Exception:

This class has no child classes.

Alphabetic index HTML hierarchy of classes or Java

In file ..\..\src\Pegasus\Common\Exception.h:

class PEGASUS_COMMON_LINKAGE CannotOpenFile

ATTN:

Documentation

ATTN:

Inheritance:

Inherited from Exception:

This class has no child classes.

Alphabetic index HTML hierarchy of classes or Java

In file ..\..\src\Pegasus\Common\Exception.h:

class PEGASUS_COMMON_LINKAGE ClassAlreadyResolved

ATTN:

Documentation

ATTN:

Inheritance:

Inherited from Exception:

This class has no child classes.

Alphabetic index HTML hierarchy of classes or Java

In file ..\..\src\Pegasus\Common\Exception.h:

class PEGASUS_COMMON_LINKAGE ClassNotResolved

ATTN:

Documentation

ATTN:

Inheritance:

Inherited from Exception:

This class has no child classes.

Alphabetic index HTML hierarchy of classes or Java

In file ..\..\src\Pegasus\Common\Exception.h:

class PEGASUS_COMMON_LINKAGE Exception

Class Exception

Documentation

Class Exception

Inheritance:

Direct child classes:

UnitializedHandle
UndeclaredQualifier
TypeMismatch
TruncatedCharacter
OutOfBounds
NullType
NullPointer
NotImplemented
NoSuchProperty
NoSuchNameSpace
NoSuchFile
NoSuchDirectory
MissingReferenceClassName
InvalidPropertyOverride
InvalidMethodOverride
InstantiatedAbstractClass
InstanceAlreadyResolved
IllegalTypeTag
IllegalName
FailedToRemoveFile
FailedToRemoveDirectory
ExpectedReferenceValue
ClassNotResolved
ClassAlreadyResolved
CannotOpenFile
CannotCreateDirectory

CIMException
BadReference
BadQualifierScope
BadQualifierOverride
AssertionFailureException
AlreadyExists
AddedReferenceToClass

Alphabetic index *HTML hierarchy of classes* *or* *Java*

Pegasus Documentation

In file ..\..\src\Pegasus\Common\Exception.h:

class PEGASUS_COMMON_LINKAGE ExpectedReferenceValue

ATTN:

Documentation

ATTN:

Inheritance:

Inherited from Exception:

This class has no child classes.

Alphabetic index HTML hierarchy of classes or Java

In file ..\..\src\Pegasus\Common\Exception.h:

class PEGASUS_COMMON_LINKAGE FailedToRemoveDirectory

ATTN:

Documentation

ATTN:

Inheritance:

Inherited from Exception:

This class has no child classes.

Alphabetic index HTML hierarchy of classes or Java

In file ..\..\src\Pegasus\Common\Exception.h:

class PEGASUS_COMMON_LINKAGE FailedToRemoveFile

ATTN:

Documentation

ATTN:

Inheritance:

Inherited from Exception:

This class has no child classes.

Alphabetic index HTML hierarchy of classes or Java

In file ..\..\src\Pegasus\Common\Exception.h:

class PEGASUS_COMMON_LINKAGE IllegalName

ATTN:

Documentation

ATTN:

Inheritance:

Inherited from Exception:

This class has no child classes.

Alphabetic index HTML hierarchy of classes or Java

In file ..\..\src\Pegasus\Common\Exception.h:

class PEGASUS_COMMON_LINKAGE IllegalTypeTag

ATTN:

Documentation

ATTN:

Inheritance:

Inherited from Exception:

This class has no child classes.

Alphabetic index HTML hierarchy of classes or Java

In file ..\..\src\Pegasus\Common\Exception.h:

class PEGASUS_COMMON_LINKAGE InstanceAlreadyResolved

ATTN:

Documentation

ATTN:

Inheritance:

Inherited from Exception:

This class has no child classes.

Alphabetic index HTML hierarchy of classes or Java

In file ..\..\src\Pegasus\Common\Exception.h:

class PEGASUS_COMMON_LINKAGE InstantiatedAbstractClass

ATTN:

Documentation

ATTN:

Inheritance:

Inherited from Exception:

This class has no child classes.

Alphabetic index HTML hierarchy of classes or Java

In file ..\..\src\Pegasus\Common\Exception.h:

class PEGASUS_COMMON_LINKAGE InvalidMethodOverride

ATTN:

Documentation

ATTN:

Inheritance:

Inherited from Exception:

This class has no child classes.

Alphabetic index HTML hierarchy of classes or Java

In file ..\..\src\Pegasus\Common\Exception.h:

class PEGASUS_COMMON_LINKAGE InvalidPropertyOverride

ATTN:

Documentation

ATTN:

Inheritance:

Inherited from Exception:

This class has no child classes.

Alphabetic index HTML hierarchy of classes or Java

In file ..\..\src\Pegasus\Common\Exception.h:

class PEGASUS_COMMON_LINKAGE MissingReferenceClassName

ATTN:

Documentation

ATTN:

Inheritance:

Inherited from Exception:

This class has no child classes.

Alphabetic index HTML hierarchy of classes or Java

In file ..\..\src\Pegasus\Common\Exception.h:

class PEGASUS_COMMON_LINKAGE NoSuchDirectory

ATTN:

Documentation

ATTN:

Inheritance:

Inherited from Exception:

This class has no child classes.

Alphabetic index HTML hierarchy of classes or Java

In file ..\..\src\Pegasus\Common\Exception.h:

class PEGASUS_COMMON_LINKAGE NoSuchFile

ATTN:

Documentation

ATTN:

Inheritance:

Inherited from Exception:

This class has no child classes.

Alphabetic index HTML hierarchy of classes or Java

In file ..\..\src\Pegasus\Common\Exception.h:

class PEGASUS_COMMON_LINKAGE NoSuchNameSpace

ATTN:

Documentation

ATTN:

Inheritance:

Inherited from Exception:

This class has no child classes.

Alphabetic index HTML hierarchy of classes or Java

In file ..\..\src\Pegasus\Common\Exception.h:

class PEGASUS_COMMON_LINKAGE NoSuchProperty

ATTN:

Documentation

ATTN:

Inheritance:

Inherited from Exception:

This class has no child classes.

Alphabetic index HTML hierarchy of classes or Java

In file ..\..\src\Pegasus\Common\Exception.h:

class PEGASUS_COMMON_LINKAGE NotImplemented

ATTN:

Documentation

ATTN:

Inheritance:

Inherited from Exception:

This class has no child classes.

Alphabetic index HTML hierarchy of classes or Java

In file ..\..\src\Pegasus\Common\Exception.h:

class PEGASUS_COMMON_LINKAGE NullPointer

ATTN:

Documentation

ATTN:

Inheritance:

Inherited from Exception:

This class has no child classes.

Alphabetic index HTML hierarchy of classes or Java

In file ..\..\src\Pegasus\Common\Exception.h:

class PEGASUS_COMMON_LINKAGE NullType

ATTN:

Documentation

ATTN:

Inheritance:

Inherited from Exception:

This class has no child classes.

Alphabetic index HTML hierarchy of classes or Java

In file ..\..\src\Pegasus\Common\Exception.h:

class PEGASUS_COMMON_LINKAGE OutOfBounds

ATTN:

Documentation

ATTN:

Inheritance:

Inherited from Exception:

This class has no child classes.

Alphabetic index HTML hierarchy of classes or Java

In file ..\..\src\Pegasus\Common\Exception.h:

class PEGASUS_COMMON_LINKAGE TruncatedCharacter

ATTN:

Documentation

ATTN:

Inheritance:

Inherited from Exception:

This class has no child classes.

Alphabetic index HTML hierarchy of classes or Java

In file ..\..\src\Pegasus\Common\Exception.h:

class PEGASUS_COMMON_LINKAGE TypeMismatch

ATTN:

Documentation

ATTN:

Inheritance:

Inherited from Exception:

This class has no child classes.

Alphabetic index HTML hierarchy of classes or Java

In file ..\..\src\Pegasus\Common\Exception.h:

class PEGASUS_COMMON_LINKAGE UndeclaredQualifier

ATTN:

Documentation

ATTN:

Inheritance:

Inherited from Exception:

This class has no child classes.

Alphabetic index HTML hierarchy of classes or Java

In file ..\..\src\Pegasus\Common\Exception.h:

class PEGASUS_COMMON_LINKAGE UnitializedHandle

ATTN:

Documentation

ATTN:

Inheritance:

Inherited from Exception:

This class has no child classes.

Alphabetic index HTML hierarchy of classes or Java

In file ..\..\src\Pegasus\Common\StopWatch.h:

class PEGASUS_COMMON_LINKAGE Stopwatch

Stopwatch - A class for measuring elapsed time Stopwatch is a class for measuring time intervals within the environment.

Documentation

Stopwatch - A class for measuring elapsed time Stopwatch is a class for measuring time intervals within the environment. It is intended to be a developers tool primarily.

Inheritance:

Public Methods

	Stopwatch () *stopwatch constructor.*
void	**reset** () *Reset Stopwatch resets an existing Stopwatch object to the current time value*
double	**getElapsed** () const *getElapsed - Get the elapsed time for the defined stopwatch.*
void	**printElapsed** () *printElapsed method sends the current value of the timer and sends it to standardout as a string with the word seconds attached*

- **`Stopwatch()`**

  stopwatch constructor. The constructor creates the object and starts the timer @example Stopwatch time;

- **`void reset()`**

  Reset Stopwatch resets an existing Stopwatch object to the current time value

- **`double getElapsed() const`**

  getElapsed - Get the elapsed time for the defined stopwatch.

  Returns:

  Returns the elapsed time value as a double

- **`void printElapsed()`**

  printElapsed method sends the current value of the timer and sends it to standardout as a string with the word seconds attached

This class has no child classes.

Alphabetic index HTML hierarchy of classes or Java

Pegasus Documentation

THE Open GROUP
... enabling enterprise integration

In file ..\..\src\Pegasus\Common\String.h:

class PEGASUS_COMMON_LINKAGE String

The Pegasus String C++ Class implements the CIM string type.

Documentation

The Pegasus String C++ Class implements the CIM string type. This class is based on the general handle/representation pattern defined for all the Pegasus objects. However, it differes from most in that it implements "copy on assign" all of the others implement "no copy on assign" semantics. The string class uses the Array class and implements an array of characters.

Inheritance:

Public Fields

static const String	**EMPTY** *EMPTY - Represent an empty string.*

Public Methods

	String () *Default constructor without parameters.*
	String (const String& x) *Copy constructor*
	String (const String& x, Uint32 n) *Initialize with first n characters from x*
	String (const Char16* x) *Initialize with x*
	String (const Char16* x, Uint32 n) *Initialize with first n characters of x*
	String (const char* x) *Initialize from a plain old C-String:*
	String (const char* x, Uint32 n) *Initialize from the first n characters of a plain old C-String:*
	~String () *String destructor.*
String&	**operator=** (const String& x) *Assign this string with x.*

String&	**operator=** (const Char16* x) *Assign this string with x*
String&	**assign** (const String& x) *Assign this string with x*
String&	**assign** (const Char16* x) *Assign this string with x*
String&	**assign** (const Char16* x, Uint32 n) *Assign this string with first n characters of x*
String&	**assign** (const char* x) *Assign this string with the plain old C-String x*
String&	**assign** (const char* x, Uint32 n) *Assign this string with first n characters of the plain old C-String x*
void	**clear** () *Clear this string.*
void	**reserve** (Uint32 capacity) *Reserves memory for capacity characters.*
Uint32	**size** () const *Returns the length of the String object.*
const Char16*	**getData** () const *Returns a pointer to the first character in the null-terminated string string.*
char*	**allocateCString** (Uint32 extraBytes = 0, Boolean noThrow = false) const *Allocates an 8 bit representation of this string.*
void	**appendToCString** (char* str, Uint32 length = Uint32(-1), Boolean noThrow = false) const *Append the given string to a C-string.*
Char16&	**operator[]** (Uint32 i) *Returns the Ith character of the String object.*
const Char16	**operator[]** (Uint32 i) const *Returns the Ith character of the String (const version).*
String&	**append** (const Char16& c) *Append the given character to the string.*
String&	**append** (const Char16* str, Uint32 n) *Append n characters from str to this String object*
String&	**append** (const String& str) *Append the characters of str to this String object*
String&	**operator+=** (const String& x) *Overload operator += appends the parameter String to this String.*
String&	**operator+=** (Char16 c) *Append the character given by c to this String object.*
String&	**operator+=** (char c) *Append the character given by c to this string.*
void	**remove** (Uint32 pos, Uint32 size = Uint32(-1)) *Remove size characters from the string starting at the given position.*

String	**subString** (Uint32 pos, Uint32 length = Uint32(-1)) const *Return a new String which is initialzed with `length` characters from this string starting at `pos`.*
Uint32	**find** (Char16 c) const *Find the position of the first occurence of the character c.*
Uint32	**find** (const String& s) const *Find the position of the first occurence of the string object.*
Uint32	**find** (const Char16* s) const *Find substring @ param - 16 bit character pointer*
Uint32	**find** (const char* s) const *find substring*
Uint32	**reverseFind** (Char16 c) const *Same as find() but start looking in reverse (last character first).*
void	**toLower** () *Converts all characters in this string to lower case*
void	**translate** (Char16 fromChar, Char16 toChar) *Translate any occurences of fromChar to toChar*
static int	**compare** (const Char16* s1, const Char16* s2, Uint32 n) *Compare the first n characters of the two strings.*
static int	**compare** (const Char16* s1, const Char16* s2) *Compare two null-terminated strings.*
static Boolean	**equal** (const String& x, const String& y) *Compare two String objects for equality.*
static Boolean	**equal** (const String& x, const Char16* y) *Return true if the two strings are equal*
static Boolean	**equal** (const Char16* x, const String& y) *Return true if the two strings are equal*
static Boolean	**equal** (const String& x, const char* y) *Return true if the two strings are equal*
static Boolean	**equal** (const char* x, const String& y) *Return true if the two strings are equal*
static void	**toLower** (char* str) *Convert the plain old C-string to lower case:*

- **`String()`**

  Default constructor without parameters. This constructor creates a null string

```
String test;
```

- **`String(const String& x)`**

  Copy constructor

- **`String(const String& x, Uint32 n)`**

Initialize with first n characters from x

- **String(const Char16* x)**

  Initialize with x

- **String(const Char16* x, Uint32 n)**

  Initialize with first n characters of x

- **String(const char* x)**

  Initialize from a plain old C-String:

- **String(const char* x, Uint32 n)**

  Initialize from the first n characters of a plain old C-String:

- **~String()**

  String destructor. Used by the representation of the String object

- **String& operator=(const String& x)**

  Assign this string with x.

```
String t1 = "abc";
String t2 = t1;
```

- **String& operator=(const Char16* x)**

  Assign this string with x

- **String& assign(const String& x)**

  Assign this string with x

- **String& assign(const Char16* x)**

  Assign this string with x

- **String& assign(const Char16* x, Uint32 n)**

  Assign this string with first n characters of x

- **String& assign(const char* x)**

  Assign this string with the plain old C-String x

- **String& assign(const char* x, Uint32 n)**

  Assign this string with first n characters of the plain old C-String x

- **void clear()**

  Clear this string. After calling clear(), size() will return 0.

```
String test = "abc";
test.clear();        String test is now NULL (length == 0)
```

- **void reserve(Uint32 capacity)**

Reserves memory for capacity characters. Notice that this does not change the size of the string (size() returns what it did before). If the capacity of the string is already greater or equal to the capacity argument, this method has no effect. After calling reserve(), getCapicty() returns a value which is greater or equal to the capacity argument.

Parameters:

capacity - defines the capacity in characters to reserve.

Uint32 size() const

Returns the length of the String object.

Returns:

Length of the string in characters.

```
String s = "abcd";
assert(s.size() == 4);
```

const Char16* getData() const

Returns a pointer to the first character in the null-terminated string string.

Parameters:

-

Returns:

Pointer to the first character of the String object

```
String t1 = "abc";
const Char16* q = t1.getData();
```

char* allocateCString(Uint32 extraBytes = 0, Boolean noThrow = false) const

Allocates an 8 bit representation of this string. The user is responsible for freeing the result. If any characters are truncated, a TruncatedCharacter exception is thrown. This exception may be suppressed by passing true as the noThrow argument. Extra characters may be allocated at the end of the new string by passing a non-zero value to the extraBytes argument.

Throws:

Throws TruncatedCharacter exception if any characters are truncated

```
String test = "abc";
char* p = test.allocateCString();
```

Parameters:

extraBytes - - Defines the number of extra characters to be allocated at the end of the new string. Default is zero.
noThrow - - If true, no exception will be thrown if characters are truncated

Returns:

pointer to the new representation of the string

- **void appendToCString(char* str, Uint32 length = Uint32(-1), Boolean noThrow = false) const**

  Append the given string to a C-string. If the length is not Uint32(-1), then the lesser of the the length argument and the length of this string is truncated. Otherwise, the entire string is trunctated. The TruncatedCharacter exception is thrown if any characters are truncated.

```
const char STR0[] = "one two three four";
String s = STR0;
const char STR1[] = "zero ";
char* tmp = new char[strlen(STR1) + s.size() + 1];
strcpy(tmp, STR1);
s.appendToCString(tmp, 7);
assert(strcmp(tmp, "zero one two") == 0);
```

- **Char16& operator[](Uint32 i)**

  Returns the Ith character of the String object.

  Throws:

  - Throws exception "OutofBounds" if the index is outside the length of the string.

```
String t1 = "abc;
Char16 c = t1[1];        character b
```

- **const Char16 operator[](Uint32 i) const**

  Returns the Ith character of the String (const version).

  Throws:

  - Throws exception "OutofBounds" if the index is outside the length of the string.

- **String& append(const Char16& c)**

  Append the given character to the string.

```
 String s4 = "Hello";
s4.append(Char16(0x0000))
```

- **String& append(const Char16* str, Uint32 n)**

  Append n characters from str to this String object

- **String& append(const String& str)**

  Append the characters of str to this String object

- **String& operator+=(const String& x)**

  Overload operator += appends the parameter String to this String. @parm String to append.

  Returns:

This String

```
String test = "abc";
test += "def";
assert(test == "abcdef");
```

String& operator+=(Char16 c)

Append the character given by c to this String object.

Parameters:

c - - Single character

String& operator+=(char c)

Append the character given by c to this string.

```
String t1 = "abc";
t1 += 'd'
assert(t1 == "abcd");
```

void remove(Uint32 pos, Uint32 size = Uint32(-1))

Remove size characters from the string starting at the given position. If size is -1, then all characters after pos are removed.

Throws:

throws "OutOfBounds" exception if size is greater than length of String plus starting position for remove.

Parameters:

pos - - Position in string to start remove
- size - Number of characters to remove. Default is -1 which causes all characters after pos to be removed

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

String subString(Uint32 pos, Uint32 length = Uint32(-1)) const

Return a new String which is initialzed with `length` characters from this string starting at `pos`.

Parameters:

- `pos` is the positon in string to start getting the substring.

- `length` is the number of characters to get. If length is -1, then all characters after pos are added to the new string.

Returns:

String with the defined substring.

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

- **`Uint32 find(Char16 c) const`**

  Find the position of the first occurence of the character c. If the character is not found, -1 is returned.

  Parameters:

  c - - Char to be found in the String

  Returns:

  Position of the character in the string or -1 if not found.

- **`Uint32 find(const String& s) const`**

  Find the position of the first occurence of the string object. This function finds one string inside another If the matching substring is not found, -1 is returned.

  Parameters:

  s - - String object to be found in the String

  Returns:

  Position of the substring in the String or -1 if not found.

- **`Uint32 find(const Char16* s) const`**

  Find substring @ param - 16 bit character pointer

  See Also:

  also find

- **`Uint32 find(const char* s) const`**

  find substring

  Parameters:

  char - * to substring

- **`Uint32 reverseFind(Char16 c) const`**

  Same as find() but start looking in reverse (last character first). @Seealso find

  Returns:

  Position of the character in the string or -1 if not found. NOTE: This function is defined only for char* input, not for String.

- **`void toLower()`**

  Converts all characters in this string to lower case

- **`void translate(Char16 fromChar, Char16 toChar)`**

Translate any occurences of fromChar to toChar

- **static int compare(const Char16* s1, const Char16* s2, Uint32 n)**

  Compare the first n characters of the two strings.

  Parameters:

  s1 - - First null-terminated string for the comparison
  s2 - - Second null-terminated string for the comparison
  n - - Number of characters to compare

  Returns:

  Return -1 if s1 is lexographically less than s2. If they are equavalent return 0. Otherwise return 1.

- **static int compare(const Char16* s1, const Char16* s2)**

  Compare two null-terminated strings.

  Parameters:

  s1 - - First null-terminated string for the comparison
  s2 - - Second null-terminated string for the comparison

  Returns:

  If s1 is less than s2, return -1; if equal return 0; otherwise, return 1. NOTE: Use the comparison operators <,<= > >= to compare String objects.

- **static Boolean equal(const String& x, const String& y)**

  Compare two String objects for equality.

  Parameters:

  s1 - - First `String` for comparison.
  s2 - - Second `String` for comparison.

  Returns:

  Boolean true if the two strings are equal.

```
String s1 = "Hello World";
String s2 = s1;
String s3(s2);
assert(String::equal(s1, s3));
```

- **static Boolean equal(const String& x, const Char16* y)**

  Return true if the two strings are equal

- **static Boolean equal(const Char16* x, const String& y)**

  Return true if the two strings are equal

- **static Boolean equal(const String& x, const char* y)**

  Return true if the two strings are equal

- **static Boolean equal(const char* x, const String& y)**

  Return true if the two strings are equal

- **`static void toLower(char* str)`**

  Convert the plain old C-string to lower case:

- **`static const String EMPTY`**

  EMPTY - Represent an empty string. This member is used to represent empty strings. Using this member avoids an expensive construction of an empty string (e.g., String()).

This class has no child classes.

Alphabetic index *HTML hierarchy of classes* *or* *Java*

In file ..\..\src\Pegasus\Common\TimeValue.h:

class PEGASUS_COMMON_LINKAGE TimeValue

The TimeValue class represents time expressed in seconds plus microseconds

---

Documentation

The TimeValue class represents time expressed in seconds plus microseconds

---

Inheritance:

This class has no child classes.

Alphabetic index HTML hierarchy of classes or Java

In file ..\..\src\Pegasus\Common\String.h:

PEGASUS_COMMON_LINKAGE int CompareIgnoreCase (const char* s1, const char* s2)

Compare two strings but ignore any case differences

Documentation

Compare two strings but ignore any case differences

Alphabetic index *HTML hierarchy of classes* *or* *Java*

In file ..\..\src\Pegasus\Common\String.h:

PEGASUS_COMMON_LINKAGE Boolean GetLine (std::istream& is, String& line)

Get the next line from the input file

Documentation

Get the next line from the input file

Alphabetic index *HTML hierarchy of classes* *or* *Java*

In file ..\..\src\Pegasus\Common\String.h:

PEGASUS_COMMON_LINKAGE String ToLower (const String& str)

Return a version of this string whose characters have been shifted to lower case

---

Documentation

Return a version of this string whose characters have been shifted to lower case

Alphabetic index *HTML hierarchy of classes* *or* *Java*

In file ..\..\src\Pegasus\Common\String.h:

inline Boolean operator!=(const String& x, const String& y)

String operator ==.

Documentation

String operator ==. Test for equality between two strings

Alphabetic index *HTML hierarchy of classes* *or* *Java*

In file ..\..\src\Pegasus\Common\String.h:

inline String operator+ (const String& x, const String& y)

overload operator + - Concatenates String objects.

Documentation

overload operator + - Concatenates String objects.

```
String t1 = "abc";
String t2;
t2 = t1 + "def"
assert(t2 == "abcdef");
```

Alphabetic index *HTML hierarchy of classes* *or* *Java*

In file ..\..\src\Pegasus\Common\String.h:

inline Boolean operator<(const String& x, const String& y)

overload operator < - Compares String obects.

Documentation

overload operator < - Compares String obects.

```
String t1 = "def";
String t2 = "a";
assert (t2 < t1);
```

Alphabetic index HTML hierarchy of classes or Java

In file ..\..\src\Pegasus\Common\String.h:

inline Boolean operator<=(const String& x, const String& y)

overload operator <= compares String objects.

Documentation

overload operator <= compares String objects.

Alphabetic index *HTML hierarchy of classes* *or* *Java*

In file ..\..\src\Pegasus\Common\String.h:

inline Boolean operator==(const char* x, const String& y)

String operator ==.

Documentation

String operator ==. Test for equality between two strings

Alphabetic index *HTML hierarchy of classes* *or* *Java*

In file ..\..\src\Pegasus\Common\String.h:

inline Boolean operator==(const String& x, const String& y)

String operator ==.

Documentation

String operator ==. Test for equality between two strings of any of the types String or char*.

Returns:

Boolean - True if the strings are equal

Alphabetic index HTML hierarchy of classes or Java

In file ..\..\src\Pegasus\Common\String.h:

inline Boolean operator==(const String& x, const char* y)

String operator ==.

Documentation

String operator ==. Test for equality between two strings

Alphabetic index *HTML hierarchy of classes* *or* *Java*

In file ..\..\src\Pegasus\Common\String.h:

inline Boolean operator>(const String& x, const String& y)

Overload operator > compares String objects

Documentation

Overload operator > compares String objects

Alphabetic index *HTML hierarchy of classes* *or* *Java*

In file ..\..\src\Pegasus\Common\String.h:

inline Boolean operator>=(const String& x, const String& y)

overload operator >= - Compares String objects

Documentation

overload operator >= - Compares String objects

Alphabetic index *HTML hierarchy of classes* *or* *Java*

In file ..\..\src\Pegasus\Common\Exception.h:

#define PEGASUS_ASSERT (COND)

define PEGASUS_ASSERT assertion statement.

Documentation

define PEGASUS_ASSERT assertion statement. This statement tests the condition defined by the parameters and if not True executes a

```
throw AssertionFailureException
```

defining the file, line and condition that was tested.

Alphabetic index *HTML hierarchy of classes* *or* *Java*

In file ..\..\src\Pegasus\Common\Exception.h:

#define Pegasus_Exception_h

Programming with Exceptions

Documentation

Programming with Exceptions

The Pegaus programming model uses exceptions extensively.

It is important that these exceptions be handled in any code that is created.

Where exceptions are used in Pegasus Classes and Methods, they are noted in the documentation with an indication that the method or class generates an exception.

Alphabetic index *HTML hierarchy of classes* *or* *Java*

In file ..\..\src\Pegasus\Common\CIMFlavor.h:

struct PEGASUS_COMMON_LINKAGE CIMFlavor

CIMQualifier flavor constants

Documentation

CIMQualifier flavor constants

Alphabetic index *HTML hierarchy of classes* *or* *Java*

Hierarchy of classes

alphabetic index
