AAF Object Manager Documentation

This documentation was generated using autoduck on 21-Aug-01

Class Members

- OMContainerProperty::~OMContainerProperty
- OMDataStream::OMDataStream
- OMDataStream::~OMDataStream
- OMDataStreamProperty::bitsSize
- OMDataStreamProperty::byteOrder
- OMDataStreamProperty::clearByteOrder
- OMDataStreamProperty::getBits
- OMDataStreamProperty::hasByteOrder
- OMDataStreamProperty::position
- OMDataStreamProperty::read
- OMDataStreamProperty::readTypedElements
- OMDataStreamProperty::restore
- OMDataStreamProperty::save
- OMDataStreamProperty::setBits
- OMDataStreamProperty::setByteOrder
- OMDataStreamProperty::setPosition
- OMDataStreamProperty::setSize
- OMDataStreamProperty::size
- OMDataStreamProperty::write
- OMDataStreamProperty::writeTypedElements
- OMDiskRawStorage::extend
- OMDiskRawStorage::extent
- OMDiskRawStorage::isExtendible
- OMDiskRawStorage::isPositionable
- OMDiskRawStorage::isReadable
- OMDiskRawStorage::isWritable
- OMDiskRawStorage::OMDiskRawStorage
- OMDiskRawStorage::openExistingModify
- OMDiskRawStorage::openExistingRead
- OMDiskRawStorage::openNewModify
- OMDiskRawStorage::position
- OMDiskRawStorage::read
- OMDiskRawStorage::readAt
- OMDiskRawStorage::setPosition
- OMDiskRawStorage::size
- OMDiskRawStorage::synchronize
- OMDiskRawStorage::write
- OMDiskRawStorage::writeAt
- OMDiskRawStorage::~OMDiskRawStorage
- OMFile::accessMode
- OMFile::byteOrder
- OMFile::clientRoot
- OMFile::close
- OMFile::compatibleRawStorage

- OMFile::encoding
- OMFile::fileName
- OMFile::findPropertyPath
- OMFile::isClosed
- OMFile::isOMFile
- OMFile::isOpen
- OMFile::isRecognized
- OMFile::isRecognized
- OMFile::isRecognized
- OMFile::loadMode
- OMFile::objectDirectory
- OMFile::OMFile
- OMFile::OMFile
- OMFile::OMFile
- OMFile::OMFile
- OMFile::open
- OMFile::openExistingModify
- OMFile::openExistingRead
- OMFile::openNewModify
- OMFile::rawStorage
- OMFile::referencedProperties
- OMFile::restore
- OMFile::revert
- OMFile::saveAsFile
- OMFile::saveFile
- OMFile::signature
- OMFile::validSignature
- OMFile::~OMFile
- OMFixedSizeProperty::getValue
- OMFixedSizeProperty::operator PropertyType
- OMFixedSizeProperty::operator&
- OMFixedSizeProperty::operator=
- OMFixedSizeProperty::reference
- OMFixedSizeProperty::restore
- OMFixedSizeProperty::setValue
- OMKLVStoredObject::byteOrder
- OMKLVStoredObject::close
- OMKLVStoredObject::create
- OMKLVStoredObject::createModify
- OMKLVStoredObject::createStoredStream
- OMKLVStoredObject::createWrite
- OMKLVStoredObject::isRecognized
- OMKLVStoredObject::isRecognized
- OMKLVStoredObject::isRecognized

- OMKLVStoredObject::OMKLVStoredObject
- OMKLVStoredObject::open
- OMKLVStoredObject::openModify
- OMKLVStoredObject::openRead
- OMKLVStoredObject::openStoredStream
- OMKLVStoredObject::restore
- OMKLVStoredObject::restore
- OMKLVStoredObject::restore
- OMKLVStoredObject::restore
- OMKLVStoredObject::restore
- OMKLVStoredObject::restore
- OMKLVStoredObject::restore
- OMKLVStoredObject::restore
- OMKLVStoredObject::restore
- OMKLVStoredObject::restore
- OMKLVStoredObject::restore
- OMKLVStoredObject::save
- OMKLVStoredObject::save
- OMKLVStoredObject::save
- OMKLVStoredObject::save
- OMKLVStoredObject::save
- OMKLVStoredObject::save
- OMKLVStoredObject::save
- OMKLVStoredObject::save
- OMKLVStoredObject::save
- OMKLVStoredObject::save
- OMKLVStoredObject::~OMKLVStoredObject
- OMMappedFileRawStorage::extend
- OMMappedFileRawStorage::extent
- OMMappedFileRawStorage::isExtendible
- OMMappedFileRawStorage::isPositionable
- OMMappedFileRawStorage::isReadable
- OMMappedFileRawStorage::isWritable
- OMMappedFileRawStorage::OMMappedFileRawStorage
- OMMappedFileRawStorage::openExistingModify
- OMMappedFileRawStorage::openExistingRead
- OMMappedFileRawStorage::openNewModify
- OMMappedFileRawStorage::position
- OMMappedFileRawStorage::read
- OMMappedFileRawStorage::readAt
- OMMappedFileRawStorage::setPosition
- OMMappedFileRawStorage::size
- OMMappedFileRawStorage::synchronize

- OMMappedFileRawStorage::write
- OMMappedFileRawStorage::writeAt
- OMMappedFileRawStorage::~OMMappedFileRawStorage
- OMMemoryRawStorage::extend
- OMMemoryRawStorage::extent
- OMMemoryRawStorage::isExtendible
- OMMemoryRawStorage::isPositionable
- OMMemoryRawStorage::isReadable
- OMMemoryRawStorage::isWritable
- OMMemoryRawStorage::OMMemoryRawStorage
- OMMemoryRawStorage::openNewModify
- OMMemoryRawStorage::position
- OMMemoryRawStorage::read
- OMMemoryRawStorage::read
- OMMemoryRawStorage::readAt
- OMMemoryRawStorage::setPosition
- OMMemoryRawStorage::size
- OMMemoryRawStorage::synchronize
- OMMemoryRawStorage::write
- OMMemoryRawStorage::write
- OMMemoryRawStorage::writeAt
- OMMemoryRawStorage::~OMMemoryRawStorage
- OMMSSStoredObject::close
- OMMSSStoredObject::closeStream
- OMMSSStoredObject::collectionIndexStreamName
- OMMSSStoredObject::create
- OMMSSStoredObject::createModify
- OMMSSStoredObject::createModify
- OMMSSStoredObject::createStoredStream
- OMMSSStoredObject::createStream
- OMMSSStoredObject::createWrite
- OMMSSStoredObject::isRecognized
- OMMSSStoredObject::isRecognized
- OMMSSStoredObject::isRecognized
- OMMSSStoredObject::OMMSSStoredObject
- OMMSSStoredObject::open
- OMMSSStoredObject::openModify
- OMMSSStoredObject::openModify
- OMMSSStoredObject::openRead
- OMMSSStoredObject::openRead
- OMMSSStoredObject::openStoredStream
- OMMSSStoredObject::openStream
- OMMSSStoredObject::read
- OMMSSStoredObject::readFromStream

- OMMSSStoredObject::readFromStream
- OMMSSStoredObject::readUInt16FromStream
- OMMSSStoredObject::readUInt32FromStream
- OMMSSStoredObject::readUInt64FromStream
- OMMSSStoredObject::readUInt8FromStream
- OMMSSStoredObject::readUniqueMaterialIdentificationFromStream
- OMMSSStoredObject::readUniqueObjectIdentificationFromStream
- OMMSSStoredObject::reorderUInt16
- OMMSSStoredObject::reorderUInt32
- OMMSSStoredObject::reorderUInt64
- OMMSSStoredObject::reorderUniqueMaterialIdentification
- OMMSSStoredObject::reorderUniqueObjectIdentification
- OMMSSStoredObject::restore
- OMMSSStoredObject::restore
- OMMSSStoredObject::restore
- OMMSSStoredObject::restore
- OMMSSStoredObject::restore
- OMMSSStoredObject::restore
- OMMSSStoredObject::restore
- OMMSSStoredObject::restore
- OMMSSStoredObject::restore
- OMMSSStoredObject::restore
- OMMSSStoredObject::restore
- OMMSSStoredObject::restore
- OMMSSStoredObject::restore
- OMMSSStoredObject::restore
- OMMSSStoredObject::restoreName
- OMMSSStoredObject::save
- OMMSSStoredObject::save
- OMMSSStoredObject::save
- OMMSSStoredObject::save
- OMMSSStoredObject::save
- OMMSSStoredObject::save
- OMMSSStoredObject::save
- OMMSSStoredObject::save
- OMMSSStoredObject::save
- OMMSSStoredObject::save
- OMMSSStoredObject::save
- OMMSSStoredObject::save
- OMMSSStoredObject::save
- OMMSSStoredObject::save
- OMMSSStoredObject::saveName

- OMMSSStoredObject::streamPosition
- OMMSSStoredObject::streamSetPosition
- OMMSSStoredObject::streamSetSize
- OMMSSStoredObject::streamSize
- OMMSSStoredObject::validate
- OMMSSStoredObject::write
- OMMSSStoredObject::writeSignature
- OMMSSStoredObject::writeSignature
- OMMSSStoredObject::writeToStream
- OMMSSStoredObject::writeToStream
- OMMSSStoredObject::writeUInt16ToStream
- OMMSSStoredObject::writeUInt32ToStream
- OMMSSStoredObject::writeUInt64ToStream
- OMMSSStoredObject::writeUInt8ToStream
- OMMSSStoredObject::writeUniqueMaterialIdentificationToStream
- OMMSSStoredObject::writeUniqueObjectIdentificationToStream
- OMMSSStoredObject::~OMMSSStoredObject
- OMObjectReference::isVoid
- OMObjectReference::OMObjectReference
- OMObjectReference::OMObjectReference
- OMObjectReference::OMObjectReference
- OMObjectReference::operator=
- OMObjectReference::operator==
- OMObjectReference::pointer
- OMObjectReference::~OMObjectReference
- OMOStream::endLine
- OMOStream::operator<<
- OMOStream::operator<<
- OMOStream::operator<<
- OMOStream::operator<<
- OMOStream::put
- OMOStream::put
- OMOStream::put
- OMOStream::putLine
- OMProperty::address
- OMProperty::clearPresent
- OMProperty::close
- OMProperty::container
- OMProperty::definition
- OMProperty::detach
- OMProperty::file
- OMProperty::initialize
- OMProperty::isOptional
- OMProperty::isPresent

- OMProperty::isVoid
- OMProperty::name
- OMProperty::OMProperty
- OMProperty::propertyId
- OMProperty::propertySet
- OMProperty::removeProperty
- OMProperty::setPresent
- OMProperty::setPropertySet
- OMProperty::storable
- OMProperty::store
- OMProperty::type
- OMProperty::~OMProperty
- OMPropertySet::container
- OMPropertySet::count
- OMPropertySet::get
- OMPropertySet::get
- OMPropertySet::isAllowed
- OMPropertySet::isPresent
- OMPropertySet::isPresent
- OMPropertySet::isRequired
- OMPropertySet::put
- OMPropertySet::setContainer
- OMPropertySetIterator::after
- OMPropertySetIterator::before
- OMPropertySetIterator::count
- OMPropertySetIterator::OMPropertySetIterator
- OMPropertySetIterator::operator++
- OMPropertySetIterator::operator--
- OMPropertySetIterator::property
- OMPropertySetIterator::propertyId
- OMPropertySetIterator::reset
- OMPropertySetIterator::valid
- OMPropertySetIterator::~OMPropertySetIterator
- OMPropertyTable::count
- OMPropertyTable::insert
- OMPropertyTable::isValid
- OMPropertyTable::OMPropertyTable
- OMPropertyTable::valueAt
- OMPropertyTable::~OMPropertyTable
- OMRawStorageLockBytes::Flush
- OMRawStorageLockBytes::LockRegion
- OMRawStorageLockBytes::OMRawStorageLockBytes
- OMRawStorageLockBytes::ReadAt
- OMRawStorageLockBytes::SetSize

- OMRawStorageLockBytes::Stat
- OMRawStorageLockBytes::UnlockRegion
- OMRawStorageLockBytes::WriteAt
- OMRawStorageLockBytes::~OMRawStorageLockBytes
- OMRedBlackTree::clear
- OMRedBlackTree::contains
- OMRedBlackTree::count
- OMRedBlackTree::find
- OMRedBlackTree::find
- OMRedBlackTree::height
- OMRedBlackTree::insert
- OMRedBlackTree::remove
- OMRedBlackTree::traverseInOrder
- OMRedBlackTree::traverseInPostOrder
- OMRedBlackTree::traverseInPreOrder
- OMRedBlackTreeIterator::after
- OMRedBlackTreeIterator::before
- OMRedBlackTreeIterator::count
- OMRedBlackTreeIterator::key
- OMRedBlackTreeIterator::OMRedBlackTreeIterator
- OMRedBlackTreeIterator::operator++
- OMRedBlackTreeIterator::operator--
- OMRedBlackTreeIterator::reset
- OMRedBlackTreeIterator::setValue
- OMRedBlackTreeIterator::value
- OMRedBlackTreeIterator::~OMRedBlackTreeIterator
- OMReferenceProperty::bitsSize
- OMReferenceSet::appendValue
- OMReferenceSet::contains
- OMReferenceSet::contains
- OMReferenceSet::containsObject
- OMReferenceSet::containsValue
- OMReferenceSet::count
- OMReferenceSet::createIterator
- OMReferenceSet::ensureAbsent
- OMReferenceSet::ensureAbsent
- OMReferenceSet::ensurePresent
- OMReferenceSet::find
- OMReferenceSet::findObject
- OMReferenceSet::insert
- OMReferenceSet::insertObject
- OMReferenceSet::OMReferenceSet
- OMReferenceSet::remove
- OMReferenceSet::remove

- OMReferenceSet::removeAllObjects
- OMReferenceSet::removeObject
- OMReferenceSet::removeValue
- OMReferenceSet::value
- OMReferenceSet::~OMReferenceSet
- OMReferenceSetIterator::after
- OMReferenceSetIterator::before
- OMReferenceSetIterator::copy
- OMReferenceSetIterator::count
- OMReferenceSetIterator::currentObject
- OMReferenceSetIterator::identification
- OMReferenceSetIterator::OMReferenceSetIterator
- OMReferenceSetIterator::OMReferenceSetIterator
- OMReferenceSetIterator::operator++
- OMReferenceSetIterator::operator--
- OMReferenceSetIterator::reset
- OMReferenceSetIterator::setValue
- OMReferenceSetIterator::valid
- OMReferenceSetIterator::value
- OMReferenceSetIterator::~OMReferenceSetIterator
- OMReferenceSetProperty::OMReferenceSetProperty
- OMReferenceSetProperty::referenceContainer
- OMReferenceSetProperty::~OMReferenceSetProperty
- OMReferenceVector::
- OMReferenceVector::appendObject
- OMReferenceVector::appendValue
- OMReferenceVector::clearValueAt
- OMReferenceVector::containsIndex
- OMReferenceVector::containsObject
- OMReferenceVector::containsValue
- OMReferenceVector::count
- OMReferenceVector::countOfValue
- OMReferenceVector::createIterator
- OMReferenceVector::find
- OMReferenceVector::findIndex
- OMReferenceVector::getObjectAt
- OMReferenceVector::getValueAt
- OMReferenceVector::grow
- OMReferenceVector::indexOfValue
- OMReferenceVector::insert
- OMReferenceVector::insertAt
- OMReferenceVector::insertObject
- OMReferenceVector::insertObjectAt
- OMReferenceVector::prependObject

- OMReferenceVector::prependValue
- OMReferenceVector::removeAllObjects
- OMReferenceVector::removeAt
- OMReferenceVector::removeFirst
- OMReferenceVector::removeLast
- OMReferenceVector::removeObject
- OMReferenceVector::removeObjectAt
- OMReferenceVector::removeValue
- OMReferenceVector::setObjectAt
- OMReferenceVector::setValueAt
- OMReferenceVector::valueAt
- OMReferenceVector::~OMReferenceVector
- OMReferenceVectorIterator::after
- OMReferenceVectorIterator::before
- OMReferenceVectorIterator::copy
- OMReferenceVectorIterator::count
- OMReferenceVectorIterator::currentObject
- OMReferenceVectorIterator::index
- OMReferenceVectorIterator::OMReferenceVectorIterator
- OMReferenceVectorIterator::OMReferenceVectorIterator
- OMReferenceVectorIterator::operator++
- OMReferenceVectorIterator::operator--
- OMReferenceVectorIterator::reset
- OMReferenceVectorIterator::setValue
- OMReferenceVectorIterator::valid
- OMReferenceVectorIterator::value
- OMReferenceVectorIterator::~OMReferenceVectorIterator
- OMReferenceVectorProperty::OMReferenceVectorProperty
- OMReferenceVectorProperty::referenceContainer
- OMReferenceVectorProperty::~OMReferenceVectorProperty
- OMRootStorable::close
- OMRootStorable::restoreContents
- OMRootStorable::save
- OMSet::append
- OMSet::clear
- OMSet::contains
- OMSet::count
- OMSet::find
- OMSet::find
- OMSet::insert
- OMSet::remove
- OMSetElement::identification
- OMSetElement::OMSetElement
- OMSetElement::OMSetElement

- OMSetElement::OMSetElement
- OMSetElement::operator=
- OMSetElement::operator==
- OMSetElement::~OMSetElement
- OMSetIterator::after
- OMSetIterator::before
- OMSetIterator::count
- OMSetIterator::key
- OMSetIterator::OMSetIterator
- OMSetIterator::operator++
- OMSetIterator::operator--
- OMSetIterator::reset
- OMSetIterator::setValue
- OMSetIterator::value
- OMSetIterator::~OMSetIterator
- OMSimpleProperty::bitsSize
- OMSimpleProperty::get
- OMSimpleProperty::getBits
- OMSimpleProperty::OMSimpleProperty
- OMSimpleProperty::OMSimpleProperty
- OMSimpleProperty::restore
- OMSimpleProperty::save
- OMSimpleProperty::set
- OMSimpleProperty::setBits
- OMSimpleProperty::setSize
- OMSimpleProperty::size
- OMSimpleProperty::~OMSimpleProperty
- OMStorable::attach
- OMStorable::attached
- OMStorable::classFactory
- OMStorable::close
- OMStorable::definition
- OMStorable::detach
- OMStorable::file
- OMStorable::find
- OMStorable::findProperty
- OMStorable::inFile
- OMStorable::isDirty
- OMStorable::isRoot
- OMStorable::name
- OMStorable::onCopy
- OMStorable::onRestore
- OMStorable::onSave
- OMStorable::pathName

- OMStorable::persistent
- OMStorable::propertySet
- OMStorable::restoreContents
- OMStorable::restoreFrom
- OMStorable::save
- OMStorable::setClassFactory
- OMStorable::setDefinition
- OMStorable::setName
- OMStorable::setStore
- OMStorable::store
- OMStoredObject::~OMStoredObject
- OMStoredPropertySetIndex::entries
- OMStoredPropertySetIndex::find
- OMStoredPropertySetIndex::insert
- OMStoredPropertySetIndex::isValid
- OMStoredPropertySetIndex::iterate
- OMStoredSetIndex::entries
- OMStoredSetIndex::firstFreeKey
- OMStoredSetIndex::insert
- OMStoredSetIndex::isValid
- OMStoredSetIndex::iterate
- OMStoredSetIndex::lastFreeKey
- OMStoredSetIndex::OMStoredSetIndex
- OMStoredSetIndex::setFirstFreeKey
- OMStoredSetIndex::setLastFreeKey
- OMStoredSetIndex::~OMStoredSetIndex
- OMStoredVectorIndex::entries
- OMStoredVectorIndex::firstFreeKey
- OMStoredVectorIndex::insert
- OMStoredVectorIndex::isValid
- OMStoredVectorIndex::iterate
- OMStoredVectorIndex::lastFreeKey
- OMStoredVectorIndex::OMStoredVectorIndex
- OMStoredVectorIndex::setFirstFreeKey
- OMStoredVectorIndex::setLastFreeKey
- OMStoredVectorIndex::~OMStoredVectorIndex
- OMStreamProperty::appendElement
- OMStreamProperty::appendElements
- OMStreamProperty::elementCount
- OMStreamProperty::index
- OMStreamProperty::OMStreamProperty
- OMStreamProperty::readElement
- OMStreamProperty::readElement
- OMStreamProperty::readElements

- OMStreamProperty::readElements
- OMStreamProperty::setElementCount
- OMStreamProperty::setIndex
- OMStreamProperty::writeElement
- OMStreamProperty::writeElement
- OMStreamProperty::writeElements
- OMStreamProperty::writeElements
- OMStreamProperty::~OMStreamProperty
- OMStrongObjectReference::clearLoaded
- OMStrongObjectReference::close
- OMStrongObjectReference::detach
- OMStrongObjectReference::getValue
- OMStrongObjectReference::isLoaded
- OMStrongObjectReference::isVoid
- OMStrongObjectReference::load
- OMStrongObjectReference::OMStrongObjectReference
- OMStrongObjectReference::OMStrongObjectReference
- OMStrongObjectReference::OMStrongObjectReference
- OMStrongObjectReference::operator=
- OMStrongObjectReference::operator==
- OMStrongObjectReference::restore
- OMStrongObjectReference::save
- OMStrongObjectReference::setLoaded
- OMStrongObjectReference::setValue
- OMStrongObjectReference::~OMStrongObjectReference
- OMStrongReference::OMStrongReference
- OMStrongReference::~OMStrongReference
- OMStrongReferenceProperty::clearValue
- OMStrongReferenceProperty::close
- OMStrongReferenceProperty::detach
- OMStrongReferenceProperty::getBits
- OMStrongReferenceProperty::getObject
- OMStrongReferenceProperty::getValue
- OMStrongReferenceProperty::isVoid
- OMStrongReferenceProperty::operator ReferencedObject*
- OMStrongReferenceProperty::operator->
- OMStrongReferenceProperty::operator->
- OMStrongReferenceProperty::operator=
- OMStrongReferenceProperty::removeProperty
- OMStrongReferenceProperty::restore
- OMStrongReferenceProperty::save
- OMStrongReferenceProperty::setBits
- OMStrongReferenceProperty::setObject
- OMStrongReferenceProperty::setValue

- OMStrongReferenceProperty::storable
- OMStrongReferenceSet::OMStrongReferenceSet
- OMStrongReferenceSet::~OMStrongReferenceSet
- OMStrongReferenceSetElement::identification
- OMStrongReferenceSetElement::OMStrongReferenceSetElement
- OMStrongReferenceSetElement::OMStrongReferenceSetElement
- OMStrongReferenceSetElement::OMStrongReferenceSetElement
- OMStrongReferenceSetElement::operator=
- OMStrongReferenceSetElement::operator==
- OMStrongReferenceSetElement::referenceCount
- OMStrongReferenceSetElement::setValue
- OMStrongReferenceSetElement::~OMStrongReferenceSetElement
- OMStrongReferenceSetIterator::after
- OMStrongReferenceSetIterator::before
- OMStrongReferenceSetIterator::clearValue
- OMStrongReferenceSetIterator::copy
- OMStrongReferenceSetIterator::count
- OMStrongReferenceSetIterator::currentObject
- OMStrongReferenceSetIterator::identification
- OMStrongReferenceSetIterator::OMStrongReferenceSetIterator
- OMStrongReferenceSetIterator::OMStrongReferenceSetIterator
- OMStrongReferenceSetIterator::operator++
- OMStrongReferenceSetIterator::operator--
- OMStrongReferenceSetIterator::reset
- OMStrongReferenceSetIterator::setValue
- OMStrongReferenceSetIterator::valid
- OMStrongReferenceSetIterator::value
- OMStrongReferenceSetIterator::~OMStrongReferenceSetIterator
- OMStrongReferenceSetProperty::appendValue
- OMStrongReferenceSetProperty::bitsSize
- OMStrongReferenceSetProperty::close
- OMStrongReferenceSetProperty::contains
- OMStrongReferenceSetProperty::contains
- OMStrongReferenceSetProperty::containsObject
- OMStrongReferenceSetProperty::containsValue
- OMStrongReferenceSetProperty::count
- OMStrongReferenceSetProperty::createIterator
- OMStrongReferenceSetProperty::detach
- OMStrongReferenceSetProperty::ensureAbsent
- OMStrongReferenceSetProperty::ensureAbsent
- OMStrongReferenceSetProperty::ensurePresent
- OMStrongReferenceSetProperty::find
- OMStrongReferenceSetProperty::findObject
- OMStrongReferenceSetProperty::getBits

- OMStrongReferenceSetProperty::insert
- OMStrongReferenceSetProperty::insertObject
- OMStrongReferenceSetProperty::isVoid
- OMStrongReferenceSetProperty::OMStrongReferenceSetProperty
- OMStrongReferenceSetProperty::remove
- OMStrongReferenceSetProperty::remove
- OMStrongReferenceSetProperty::removeAllObjects
- OMStrongReferenceSetProperty::removeObject
- OMStrongReferenceSetProperty::removeProperty
- OMStrongReferenceSetProperty::removeValue
- OMStrongReferenceSetProperty::restore
- OMStrongReferenceSetProperty::save
- OMStrongReferenceSetProperty::setBits
- OMStrongReferenceSetProperty::value
- OMStrongReferenceSetProperty::~OMStrongReferenceSetProperty
- OMStrongReferenceVector::OMStrongReferenceVector
- OMStrongReferenceVector::~OMStrongReferenceVector
- OMStrongReferenceVectorElement::localKey
- OMStrongReferenceVectorElement::OMStrongReferenceVectorElement
- OMStrongReferenceVectorElement::OMStrongReferenceVectorElement
- OMStrongReferenceVectorElement::OMStrongReferenceVectorElement
- OMStrongReferenceVectorElement::operator=
- OMStrongReferenceVectorElement::operator==
- OMStrongReferenceVectorElement::setValue
- OMStrongReferenceVectorElement::~OMStrongReferenceVectorElement
- OMStrongReferenceVectorIterator::
- OMStrongReferenceVectorIterator::
- OMStrongReferenceVectorIterator::
- OMStrongReferenceVectorIterator::after
- OMStrongReferenceVectorIterator::before
- OMStrongReferenceVectorIterator::clearValue
- OMStrongReferenceVectorIterator::copy
- OMStrongReferenceVectorIterator::count
- OMStrongReferenceVectorIterator::currentObject
- OMStrongReferenceVectorIterator::index
- OMStrongReferenceVectorIterator::operator++
- OMStrongReferenceVectorIterator::operator--
- OMStrongReferenceVectorIterator::reset
- OMStrongReferenceVectorIterator::setValue
- OMStrongReferenceVectorIterator::valid
- OMStrongReferenceVectorIterator::value
- OMStrongReferenceVectorProperty::
- OMStrongReferenceVectorProperty::
- OMStrongReferenceVectorProperty::appendObject

- OMStrongReferenceVectorProperty::appendValue
- OMStrongReferenceVectorProperty::bitsSize
- OMStrongReferenceVectorProperty::clearValueAt
- OMStrongReferenceVectorProperty::close
- OMStrongReferenceVectorProperty::containsIndex
- OMStrongReferenceVectorProperty::containsObject
- OMStrongReferenceVectorProperty::containsValue
- OMStrongReferenceVectorProperty::count
- OMStrongReferenceVectorProperty::countOfValue
- OMStrongReferenceVectorProperty::createIterator
- OMStrongReferenceVectorProperty::detach
- OMStrongReferenceVectorProperty::find
- OMStrongReferenceVectorProperty::findIndex
- OMStrongReferenceVectorProperty::getBits
- OMStrongReferenceVectorProperty::getObjectAt
- OMStrongReferenceVectorProperty::getValueAt
- OMStrongReferenceVectorProperty::grow
- OMStrongReferenceVectorProperty::indexOfValue
- OMStrongReferenceVectorProperty::insert
- OMStrongReferenceVectorProperty::insertAt
- OMStrongReferenceVectorProperty::insertObject
- OMStrongReferenceVectorProperty::insertObjectAt
- OMStrongReferenceVectorProperty::isVoid
- OMStrongReferenceVectorProperty::prependObject
- OMStrongReferenceVectorProperty::prependValue
- OMStrongReferenceVectorProperty::removeAllObjects
- OMStrongReferenceVectorProperty::removeAt
- OMStrongReferenceVectorProperty::removeFirst
- OMStrongReferenceVectorProperty::removeLast
- OMStrongReferenceVectorProperty::removeObject
- OMStrongReferenceVectorProperty::removeObjectAt
- OMStrongReferenceVectorProperty::removeProperty
- OMStrongReferenceVectorProperty::removeValue
- OMStrongReferenceVectorProperty::restore
- OMStrongReferenceVectorProperty::save
- OMStrongReferenceVectorProperty::setBits
- OMStrongReferenceVectorProperty::setObjectAt
- OMStrongReferenceVectorProperty::setValueAt
- OMStrongReferenceVectorProperty::valueAt
- OMType::contract
- OMType::copy
- OMType::expand
- OMType::reorderInteger
- OMVariableSizeProperty::appendValue

- OMVariableSizeProperty::copyElementsToBuffer
- OMVariableSizeProperty::copyToBuffer
- OMVariableSizeProperty::count
- OMVariableSizeProperty::getValue
- OMVariableSizeProperty::getValueAt
- OMVariableSizeProperty::prependValue
- OMVariableSizeProperty::restore
- OMVariableSizeProperty::setElementValues
- OMVariableSizeProperty::setValue
- OMVariableSizeProperty::setValueAt
- OMVector::append
- OMVector::capacity
- OMVector::clear
- OMVector::containsValue
- OMVector::count
- OMVector::countValue
- OMVector::empty
- OMVector::full
- OMVector::getAt
- OMVector::getAt
- OMVector::grow
- OMVector::indexOfValue
- OMVector::insert
- OMVector::insertAt
- OMVector::nextHigherCapacity
- OMVector::OMVector
- OMVector::prepend
- OMVector::removeAt
- OMVector::removeFirst
- OMVector::removeLast
- OMVector::removeValue
- OMVector::setAt
- OMVector::shrink
- OMVector::valueAt
- OMVector::~OMVector
- OMVectorElement::getValue
- OMVectorElement::OMVectorElement
- OMVectorElement::OMVectorElement
- OMVectorElement::OMVectorElement
- OMVectorElement::operator=
- OMVectorElement::operator==
- OMVectorElement::pointer
- OMVectorElement::setValue
- OMVectorElement::~OMVectorElement

- OMVectorIterator::after
- OMVectorIterator::before
- OMVectorIterator::count
- OMVectorIterator::index
- OMVectorIterator::OMVectorIterator
- OMVectorIterator::operator++
- OMVectorIterator::operator--
- OMVectorIterator::reset
- OMVectorIterator::setValue
- OMVectorIterator::value
- OMVectorIterator::~OMVectorIterator
- OMWeakObjectReference::close
- OMWeakObjectReference::detach
- OMWeakObjectReference::getValue
- OMWeakObjectReference::isVoid
- OMWeakObjectReference::OMWeakObjectReference
- OMWeakObjectReference::OMWeakObjectReference
- OMWeakObjectReference::OMWeakObjectReference
- OMWeakObjectReference::OMWeakObjectReference
- OMWeakObjectReference::operator=
- OMWeakObjectReference::operator==
- OMWeakObjectReference::restore
- OMWeakObjectReference::save
- OMWeakObjectReference::setValue
- OMWeakObjectReference::~OMWeakObjectReference
- OMWeakReference::OMWeakReference
- OMWeakReference::~OMWeakReference
- OMWeakReferenceProperty::clearValue
- OMWeakReferenceProperty::close
- OMWeakReferenceProperty::getBits
- OMWeakReferenceProperty::getObject
- OMWeakReferenceProperty::getValue
- OMWeakReferenceProperty::isVoid
- OMWeakReferenceProperty::OMWeakReferenceProperty
- OMWeakReferenceProperty::OMWeakReferenceProperty
- OMWeakReferenceProperty::operator ReferencedObject*
- OMWeakReferenceProperty::operator->
- OMWeakReferenceProperty::operator=
- OMWeakReferenceProperty::restore
- OMWeakReferenceProperty::save
- OMWeakReferenceProperty::setBits
- OMWeakReferenceProperty::setObject
- OMWeakReferenceProperty::setValue
- OMWeakReferenceSet::OMWeakReferenceSet

- OMWeakReferenceSet::~OMWeakReferenceSet
- OMWeakReferenceSetElement::identification
- OMWeakReferenceSetElement::OMWeakReferenceSetElement
- OMWeakReferenceSetElement::OMWeakReferenceSetElement
- OMWeakReferenceSetElement::OMWeakReferenceSetElement
- OMWeakReferenceSetElement::operator=
- OMWeakReferenceSetElement::operator==
- OMWeakReferenceSetElement::setValue
- OMWeakReferenceSetElement::~OMWeakReferenceSetElement
- OMWeakReferenceSetIterator::after
- OMWeakReferenceSetIterator::before
- OMWeakReferenceSetIterator::clearValue
- OMWeakReferenceSetIterator::copy
- OMWeakReferenceSetIterator::count
- OMWeakReferenceSetIterator::currentObject
- OMWeakReferenceSetIterator::identification
- OMWeakReferenceSetIterator::OMWeakReferenceSetIterator
- OMWeakReferenceSetIterator::OMWeakReferenceSetIterator
- OMWeakReferenceSetIterator::operator++
- OMWeakReferenceSetIterator::operator--
- OMWeakReferenceSetIterator::reset
- OMWeakReferenceSetIterator::setValue
- OMWeakReferenceSetIterator::valid
- OMWeakReferenceSetIterator::value
- OMWeakReferenceSetIterator::~OMWeakReferenceSetIterator
- OMWeakReferenceSetProperty::
- OMWeakReferenceSetProperty::appendValue
- OMWeakReferenceSetProperty::bitsSize
- OMWeakReferenceSetProperty::close
- OMWeakReferenceSetProperty::contains
- OMWeakReferenceSetProperty::contains
- OMWeakReferenceSetProperty::containsObject
- OMWeakReferenceSetProperty::containsValue
- OMWeakReferenceSetProperty::count
- OMWeakReferenceSetProperty::createIterator
- OMWeakReferenceSetProperty::detach
- OMWeakReferenceSetProperty::ensureAbsent
- OMWeakReferenceSetProperty::ensureAbsent
- OMWeakReferenceSetProperty::ensurePresent
- OMWeakReferenceSetProperty::find
- OMWeakReferenceSetProperty::findObject
- OMWeakReferenceSetProperty::getBits
- OMWeakReferenceSetProperty::insert
- OMWeakReferenceSetProperty::insertObject

- OMWeakReferenceSetProperty::isVoid
- OMWeakReferenceSetProperty::OMWeakReferenceSetProperty
- OMWeakReferenceSetProperty::remove
- OMWeakReferenceSetProperty::remove
- OMWeakReferenceSetProperty::removeAllObjects
- OMWeakReferenceSetProperty::removeObject
- OMWeakReferenceSetProperty::removeProperty
- OMWeakReferenceSetProperty::removeValue
- OMWeakReferenceSetProperty::restore
- OMWeakReferenceSetProperty::save
- OMWeakReferenceSetProperty::setBits
- OMWeakReferenceSetProperty::value
- OMWeakReferenceSetProperty::~OMWeakReferenceSetProperty
- OMWeakReferenceVector::OMWeakReferenceVector
- OMWeakReferenceVector::~OMWeakReferenceVector
- OMWeakReferenceVectorElement::identification
- OMWeakReferenceVectorElement::OMWeakReferenceVectorElement
- OMWeakReferenceVectorElement::OMWeakReferenceVectorElement
- OMWeakReferenceVectorElement::OMWeakReferenceVectorElement
- OMWeakReferenceVectorElement::operator=
- OMWeakReferenceVectorElement::operator==
- OMWeakReferenceVectorElement::setValue
- OMWeakReferenceVectorElement::~OMWeakReferenceVectorElement
- OMWeakReferenceVectorIterator::
- OMWeakReferenceVectorIterator::after
- OMWeakReferenceVectorIterator::before
- OMWeakReferenceVectorIterator::clearValue
- OMWeakReferenceVectorIterator::copy
- OMWeakReferenceVectorIterator::count
- OMWeakReferenceVectorIterator::currentObject
- OMWeakReferenceVectorIterator::identification
- OMWeakReferenceVectorIterator::index
- OMWeakReferenceVectorIterator::OMWeakReferenceVectorIterator
- OMWeakReferenceVectorIterator::OMWeakReferenceVectorIterator
- OMWeakReferenceVectorIterator::operator++
- OMWeakReferenceVectorIterator::operator--
- OMWeakReferenceVectorIterator::reset
- OMWeakReferenceVectorIterator::setValue
- OMWeakReferenceVectorIterator::valid
- OMWeakReferenceVectorIterator::value
- OMWeakReferenceVectorProperty::
- OMWeakReferenceVectorProperty::
- OMWeakReferenceVectorProperty::appendObject
- OMWeakReferenceVectorProperty::appendValue

- OMWeakReferenceVectorProperty::bitsSize
- OMWeakReferenceVectorProperty::clearValueAt
- OMWeakReferenceVectorProperty::close
- OMWeakReferenceVectorProperty::containsIndex
- OMWeakReferenceVectorProperty::containsObject
- OMWeakReferenceVectorProperty::containsValue
- OMWeakReferenceVectorProperty::count
- OMWeakReferenceVectorProperty::countOfValue
- OMWeakReferenceVectorProperty::createIterator
- OMWeakReferenceVectorProperty::detach
- OMWeakReferenceVectorProperty::find
- OMWeakReferenceVectorProperty::findIndex
- OMWeakReferenceVectorProperty::getBits
- OMWeakReferenceVectorProperty::getObjectAt
- OMWeakReferenceVectorProperty::getValueAt
- OMWeakReferenceVectorProperty::grow
- OMWeakReferenceVectorProperty::indexOfValue
- OMWeakReferenceVectorProperty::insert
- OMWeakReferenceVectorProperty::insertAt
- OMWeakReferenceVectorProperty::insertObject
- OMWeakReferenceVectorProperty::insertObjectAt
- OMWeakReferenceVectorProperty::isVoid
- OMWeakReferenceVectorProperty::OMWeakReferenceVectorProperty
- OMWeakReferenceVectorProperty::prependObject
- OMWeakReferenceVectorProperty::prependValue
- OMWeakReferenceVectorProperty::removeAllObjects
- OMWeakReferenceVectorProperty::removeAt
- OMWeakReferenceVectorProperty::removeFirst
- OMWeakReferenceVectorProperty::removeLast
- OMWeakReferenceVectorProperty::removeObject
- OMWeakReferenceVectorProperty::removeObjectAt
- OMWeakReferenceVectorProperty::removeProperty
- OMWeakReferenceVectorProperty::removeValue
- OMWeakReferenceVectorProperty::restore
- OMWeakReferenceVectorProperty::save
- OMWeakReferenceVectorProperty::setBits
- OMWeakReferenceVectorProperty::setObjectAt
- OMWeakReferenceVectorProperty::setValueAt
- OMWeakReferenceVectorProperty::valueAt
- OMWideStringProperty::OMWideStringProperty
- OMWideStringProperty::operator=
- OMWideStringProperty::~OMWideStringProperty
- OMXMLStoredObject::byteOrder
- OMXMLStoredObject::close

- OMXMLStoredObject::create
- OMXMLStoredObject::createModify
- OMXMLStoredObject::createStoredStream
- OMXMLStoredObject::createWrite
- OMXMLStoredObject::isRecognized
- OMXMLStoredObject::isRecognized
- OMXMLStoredObject::isRecognized
- OMXMLStoredObject::OMXMLStoredObject
- OMXMLStoredObject::open
- OMXMLStoredObject::openModify
- OMXMLStoredObject::openRead
- OMXMLStoredObject::openStoredStream
- OMXMLStoredObject::restore
- OMXMLStoredObject::restore
- OMXMLStoredObject::restore
- OMXMLStoredObject::restore
- OMXMLStoredObject::restore
- OMXMLStoredObject::restore
- OMXMLStoredObject::restore
- OMXMLStoredObject::restore
- OMXMLStoredObject::restore
- OMXMLStoredObject::restore
- OMXMLStoredObject::restore
- OMXMLStoredObject::save
- OMXMLStoredObject::save
- OMXMLStoredObject::save
- OMXMLStoredObject::save
- OMXMLStoredObject::save
- OMXMLStoredObject::save
- OMXMLStoredObject::save
- OMXMLStoredObject::save
- OMXMLStoredObject::save
- OMXMLStoredObject::save
- OMXMLStoredObject::~OMXMLStoredObject

Classes

- OMAssertionViolation
- OMBuiltinPropertyDefinition
- OMCharacterStringProperty
- OMClassFactory
- OMContainer

- OMContainerElement
- OMContainerIterator
- OMContainerProperty
- OMDataStream
- OMDataStreamProperty
- OMDefinition
- OMDiskRawStorage
- OMFile
- OMFixedSizeProperty
- OMIdentitySet
- OMKLVStoredObject
- OMKLVStoredStream
- OMMappedFileRawStorage
- OMMemoryRawStorage
- OMMSSStoredObject
- OMMSSStoredStream
- OMObject
- OMObjectDirectory
- OMObjectReference
- OMObjectSet
- OMObjectVector
- OMOStream
- OMProperty
- OMPropertyDefinition
- OMPropertySet
- OMPropertySetIterator
- OMPropertyTable
- OMRawStorage
- OMRawStorageLockBytes
- OMRedBlackTree
- OMRedBlackTreeIterator
- OMReferenceContainer
- OMReferenceContainerIterator
- OMReferenceProperty
- OMReferenceSet
- OMReferenceSetIterator
- OMReferenceSetProperty
- OMReferenceVector
- OMReferenceVectorIterator
- OMReferenceVectorProperty
- OMRootStorable
- OMSet
- OMSetElement
- OMSetIterator

- OMSimpleProperty
- OMSingleton
- OMStorable
- OMStoredObject
- OMStoredPropertySetIndex
- OMStoredSetIndex
- OMStoredStream
- OMStoredVectorIndex
- OMStreamProperty
- OMStrongObjectReference
- OMStrongReference
- OMStrongReferenceProperty
- OMStrongReferenceSet
- OMStrongReferenceSetElement
- OMStrongReferenceSetIterator
- OMStrongReferenceSetProperty
- OMStrongReferenceVector
- OMStrongReferenceVectorElement
- OMStrongReferenceVectorIterator
- OMStrongReferenceVectorProperty
- OMType
- OMUniqueObjectIdentificationType
- OMVariableSizeProperty
- OMVector
- OMVectorElement
- OMVectorIterator
- OMWeakObjectReference
- OMWeakReference
- OMWeakReferenceProperty
- OMWeakReferenceSet
- OMWeakReferenceSetElement
- OMWeakReferenceSetIterator
- OMWeakReferenceSetProperty
- OMWeakReferenceVector
- OMWeakReferenceVectorElement
- OMWeakReferenceVectorIterator
- OMWeakReferenceVectorProperty
- OMWideStringProperty
- OMXMLStoredObject
- OMXMLStoredStream

Enums

- OMFile::OMAccessMode
- OMFile::OMFileEncoding
- OMFile::OMLoadMode
- OMIteratorPosition

Macros and Functions

- ANAME
- ASSERT
- concatenateWideString
- copyWideString
- endl
- endl
- finalizeObjectManager
- FORALL
- FOREACH
- hostByteOrder
- IMPLIES
- initializeObjectManager
- INVARIANT
- lengthOfWideString
- NNAME
- OBSOLETE
- obsolete
- OLD
- POSTCONDITION
- PRECONDITION
- reportAssertionViolation
- SAVE
- SAVE_EXPRESSION
- saveString
- saveWideString
- squeezeWideString
- stringSize
- toWideString
- TRACE
- trace
- validOMString
- validString
- validWideString
- wfopen

Modules

- OMAssertions
- OMDataTypes
- OMMSStructuredStorage
- OMObjectManager
- OMPortability
- OMUtilities

OMAssertions

Filename: OMAssertions.h

Description

Functions and macros to implement run-time monitoring of assertions.

References ...

[1] "Object Oriented Software Construction", Bertrand Meyer, 1997 Prentice Hall PTR, ISBN 0-13-629155-4

Author

Tim Bingham - tjb - (Avid Technology, Inc.)

OMDataTypes

Filename: OMDataTypes.h

Description

Host independent data type definitions used by the Object Manager.

Author

Tim Bingham - tjb - (Avid Technology, Inc.)

OMMSStructuredStorage

Filename: OMMSStructuredStorage.h

Description

Interface to various implementations of Microsoft Structured Storage.

Author

Tim Bingham - tjb - (Avid Technology, Inc.)

OMObjectManager

Filename: OMObjectManager.h

Description

Object Manager global functions.

Author

Tim Bingham - tjb - (Avid Technology, Inc.)

OMPortability

Filename: OMPortability.h

Description

Definitions supporting the portability of the Object Manager.

Author

Tim Bingham - tjb - (Avid Technology, Inc.)

OMUtilities

Filename: OMUtilities.h

Description

Utility functions including error handling, obtaining information about the host computer, wide character string manipulation, property path manipulation and accessing disk files with wide character names.

Author

Tim Bingham - tjb - (Avid Technology, Inc.)

ANAME

define ANAME(*name***)**

Define a name only when assertions are enabled. Use to avoid compiler warnings.

Defined in: OMAssertions.h

Parameters

name
: The name to (conditionally) define.

ASSERT

define ASSERT(*name*, *expression***)**

Assert (when enabled with OM_ENABLE_DEBUG) that the condition described by *name* and *expression* is true. An invocation of this macro must be preceeded by an invocation of the TRACE macro.

Defined in: OMAssertions.h

Parameters

name
: The name of the condition. The condition name is a description of the condition that makes sense from the internal point of view (that of someone reading the source text). The name comprises a portion of the message that is printed if the condition is violated. The message that is printed makes sense from the external point of view.

expression
: The condition expression. The expression should be free of side effects.

---

concatenateWideString

wchar_t* concatenateWideString(wchar_t* *destination*, **const wchar_t*** *source*, **const size_t** *length***)**

Concatenate wide character strings. Same as strncat(), but for wide characters. Append up to *length* characters from *source* to the end of *destination*. If the null character that terminates *source* is encountered before *length* characters have been copied, then the null character is copied but no more. If no null character appears among the first *length* characters of *source*, then the first *length* characters are copied and a null character is supplied to terminate *destination*, that is, *length* + 1 characters in all are written.

Defined in: OMUtilities.h

Return Value

The resulting wide character string.

Parameters

destination
: The destination buffer.

source
: The wide character string to copy.

length
: The number of characters to copy.

---

copyWideString

wchar_t* copyWideString(wchar_t* *destination*, **const wchar_t*** *source*, **const size_t** *length***)**

Copy a wide character string. Same as strncpy(), but for wide characters. Exactly *length* characters are always written to the *destination* buffer. The *destination* buffer must be at least *length* characters in size. If the buffer is too small this error is not detected. If *length* is greater than the length of *source* then then *destination* is padded with nulls and *destination* will be properly null terminated. If *length* is less than the length of *source* only length characters will be copied and *destination* will not be properly null terminated. If the *source* and *destination* wide character strings overlap this error is not detected,.

Defined in: OMUtilities.h

Return Value

The resulting wide character string.

Parameters

destination
: The destination buffer.

source
: The wide character string to copy.

length
: The number of characters to copy.

---

endl

OMOStream& endl(OMOStream& *s***)**

OMOStream end of line manipulator.

Defined in: OMOStream.cpp

Return Value

The modified OMOStream

Parameters

s

The OMOStream in which to inset the new line.

Global Variables

OMOStream **omlog**
Global OMOStream for Object Manager logging. Debug use only.

Developer Notes

If your platform doesn't have iostream.h you'll need to implement the following functions differently.

---

endl

OMOStream& endl(OMOStream& *s*)

OMOStream end of line manipulator.

Defined in: OMOStream.h

Return Value

The modified OMOStream.

Parameters

s
The OMOStream in which to inset the new line.

Global Variables

extern OMOStream omlog
Global OMOStream for Object Manager logging. Debug use only.

---

finalizeObjectManager

void finalizeObjectManager(void)

Finalize the Object Manager.

Defined in: OMObjectManager.h

FORALL

define FORALL(*index*, *elementCount*, *expression***)**

Universal quantifier. Evaluate *expression* for all elements, 0 .. *elementCount* of a collection. Use *index* as the name of the index. The *expression* is most usefully one of the assertion macros such as PRECONDITION, POSTCONDITION or ASSERT.

Defined in: OMAssertions.h

Parameters

index
: The index name.

elementCount
: The number of elements in the collection.

expression
: The expression to evaluate for each element.

FOREACH

define FOREACH(*index*, *start*, *elementCount*, *expression***)**

Evaluate *expression* for each element, *start* .. *elementCount* of a collection. Use *index* as the name of the index. The *expression* is most usefully one of the assertion macros such as PRECONDITION, POSTCONDITION or ASSERT.

Defined in: OMAssertions.h

Parameters

index
: The index name.

start
: The starting index.

elementCount
: The number of elements.

expression
: The expression to evaluate for each element.

---

hostByteOrder

OMByteOrder hostByteOrder(void)

Get the byte order used on the host computer.

Defined in: OMUtilities.h

Return Value

The host byte order.

IMPLIES

define IMPLIES(*a*, *b***)**

Boolean implication - use **IMPLIES** in construction of other assertions. Read 'IMPLIES(a, b)' as 'a => b', or 'a implies b'. 'ASSERT("...", IMPLIES(a, b))' is the expression form of 'if (a) ASSERT("...", b)'. However, IMPLIES() hides the 'if' statement so that it can be 'compiled away'.

Defined in: OMAssertions.h

Parameters

a
: An expression.

b
: An expression.

---

initializeObjectManager

void initializeObjectManager(void)

Initialize the Object Manager.

Defined in: OMObjectManager.h

INVARIANT

define INVARIANT(void)

Assert (when enabled with OM_ENABLE_DEBUG) that the invariant for the class of the current object is true.

Defined in: OMAssertions.h

---

lengthOfWideString

size_t lengthOfWideString(const wchar_t* *string***)**

The length of the wide character string *string* in characters excluding the terminating null character. Same as strlen(), but for wide characters.

Defined in: OMUtilities.h

Return Value

The wide character string length in characters.

Parameters

string
The wide character string.

NNAME

define NNAME(*name***)**

Never define a name. Use to avoid compiler warnings.

Defined in: OMAssertions.h

Parameters

name
The name not to define.

OBSOLETE

define OBSOLETE(*newRoutineName***)**

Print a message (when enabled with OM_ENABLE_DEBUG and OM_ENABLE_OBSOLETE) indicating that the current routine is obsolete and that *newRoutineName* should be used instead. OBSOLETE is provided to aid clients in migrating from one Object Manager version to the next. Routines are made obsolete before they are removed.

Defined in: OMAssertions.h

Parameters

newRoutineName
The name of the routine that should be called instead.

---

obsolete

void obsolete(const char* *routineName***, const char*** *newRoutineName***)**

Output a message indicating that the *routineName* is obsolete and that *newRoutineName* should be used instead

Defined in: OMAssertions.h

Parameters

routineName
The name of the obsolete routine.
newRoutineName

The name of the routine that should be called instead.

OLD

define OLD(*name***)**

Retrieve the value of a variable or expression saved on entry to a routine with SAVE or with SAVE_EXPRESSION. For use in postconditions.

Defined in: OMAssertions.h

Parameters

name
The name of the saved variable or expression.

OMAssertionViolation class

OMAssertionViolation **class OMAssertionViolation**

Object Manager assertion violation. An instance of this class is thrown when an assertion violation occurs.

Defined in: OMAssertions.h

OMBuiltinPropertyDefinition class

OMBuiltinPropertyDefinition **class OMBuiltinPropertyDefinition**

Definitions of persistent properties supported by the Object Manager.

Defined in: OMPropertyDefinition.h

Author

Tim Bingham - tjb - (Avid Technology, Inc.)

Developer Notes

This is a temporary class and will be merged into OMPropertyDefinition. This will require changes to code in Object Manager clients.

Class Members

Public members.
OMBuiltinPropertyDefinition**(const OMType* type, const wchar_t* name, const OMPropertyId propertyId, const bool isOptional)**

Constructor.

~OMBuiltinPropertyDefinition**(void)**

Destructor.

virtual const OMType* type**(void) const**

The type of the OMProperty defined by this **OMBuiltinPropertyDefinition.**

virtual const wchar_t* name**(void) const**

The name of the OMProperty defined by this **OMBuiltinPropertyDefinition.**

virtual OMPropertyId localIdentification**(void) const**

The locally unique identification of the OMProperty defined by this **OMBuiltinPropertyDefinition.**

virtual bool isOptional**(void) const**

Is the OMProperty defined by this **OMBuiltinPropertyDefinition** optional?

Class Members

Private members.

OMBuiltinPropertyDefinition::isOptional

bool OMBuiltinPropertyDefinition::isOptional(void)

Is the OMProperty defined by this OMBuiltinPropertyDefinition optional?

Defined in: OMPropertyDefinition.cpp

Back to OMBuiltinPropertyDefinition

OMBuiltinPropertyDefinition::localIdentification

OMPropertyId OMBuiltinPropertyDefinition::localIdentification(void)

The locally unique identification of the OMProperty defined by this OMBuiltinPropertyDefinition.

Defined in: OMPropertyDefinition.cpp

Back to OMBuiltinPropertyDefinition

OMBuiltinPropertyDefinition::name

const wchar_t* OMBuiltinPropertyDefinition::name(void)

The name of the OMProperty defined by this OMBuiltinPropertyDefinition.

Defined in: OMPropertyDefinition.cpp

Back to OMBuiltinPropertyDefinition

OMBuiltinPropertyDefinition::OMBuiltinPropertyDefinition

OMBuiltinPropertyDefinition::OMBuiltinPropertyDefinition(void)

Constructor.

Defined in: OMPropertyDefinition.cpp

Back to OMBuiltinPropertyDefinition

OMBuiltinPropertyDefinition::type

const OMType* OMBuiltinPropertyDefinition::type(void)

The type of the OMProperty defined by this OMBuiltinPropertyDefinition.

Defined in: OMPropertyDefinition.cpp

Back to OMBuiltinPropertyDefinition

OMBuiltinPropertyDefinition::~OMBuiltinPropertyDefinition

OMBuiltinPropertyDefinition::~OMBuiltinPropertyDefinition(void)

Destructor.

Defined in: OMPropertyDefinition.cpp

Back to OMBuiltinPropertyDefinition

OMCharacterStringProperty class

OMCharacterStringProperty **class OMCharacterStringProperty: public OMVariableSizeProperty**

Abstract base class for persistent character string properties supported by the Object Manager.

Defined in: OMCharacterStringProperty.h

Class Template Arguments

CharacterType
: The type of the characters that comprise the string.

Author

Tim Bingham - tjb - (Avid Technology, Inc.)

Class Members

Public members.

OMCharacterStringProperty(const OMPropertyId propertyId, const wchar_t* name)
Constructor.

virtual ~OMCharacterStringProperty(void)
Destructor.

operator const CharacterType* () const
Type conversion. Convert an **OMCharacterStringProperty** into a string of *CharacterType* characters.

void assign**(const CharacterType* characterString)**
Assign the string *characterString* to this **OMCharacterStringProperty.**

size_t length**(void) const**
The length of this **OMCharacterStringProperty** in characters (not counting the null terminating character).

static size_t stringLength**(const CharacterType* characterString)**
Utility function for computing the length, in characters, of the string of *CharacterType* characters *characterString*.

OMCharacterStringProperty::assign

template <class *CharacterType***>**
void OMCharacterStringProperty<*CharacterType***>::assign(const CharacterType*** *characterString***)**

Assign the string *characterString* to this OMCharacterStringProperty.

Defined in: OMCharacterStringPropertyT.h

Parameters

characterString
The string of *CharacterType* characters to assign.

Class Template Arguments

CharacterType
The type of the characters that comprise the string.

Back to OMCharacterStringProperty

OMCharacterStringProperty::length

template <class *CharacterType***>**
size_t OMCharacterStringProperty<*CharacterType***>::length(void) const**

The length of this OMCharacterStringProperty in characters (not counting the null terminating character).

Defined in: OMCharacterStringPropertyT.h

Return Value

The length of this OMCharacterStringProperty.

Class Template Arguments

CharacterType
The type of the characters that comprise the string.

Back to OMCharacterStringProperty

OMCharacterStringProperty::operator const CharacterType*

template <class *CharacterType***>**
OMCharacterStringProperty<*CharacterType***>::operator const CharacterType*(void)**

Type conversion. Convert an OMCharacterStringProperty into a string of *CharacterType* characters.

Defined in: OMCharacterStringPropertyT.h

Return Value

The result of the conversion as a value of type pointer to *CharacterType*.

Class Template Arguments

CharacterType
The type of the characters that comprise the string.

Back to OMCharacterStringProperty

OMCharacterStringProperty::stringLength

template <class *CharacterType***>**
size_t OMCharacterStringProperty<*CharacterType***>::stringLength(const CharacterType*** *characterString***)**

Utility function for computing the length, in characters, of the string of *CharacterType* characters *characterString*.

Defined in: OMCharacterStringPropertyT.h

Return Value

The length of the the string of *CharacterType* characters *characterString*.

Parameters

characterString
A string of *CharacterType* characters.

Class Template Arguments

CharacterType
The type of the characters that comprise the string.

Back to OMCharacterStringProperty

OMClassFactory class

OMClassFactory **class OMClassFactory**

Abstract base class decribing the class factory used by the Object Manager and provided by Object Manager clients.

Defined in: OMClassFactory.h

Author

Tim Bingham - tjb - (Avid Technology, Inc.)

Class Members

Public members.
virtual ~OMClassFactory(void)
Destructor.
virtual OMStorable* create(const OMClassId& classId) const
Create an instance of the appropriate derived class, given the class id.

OMContainer class

OMContainer **class OMContainer**

Abstract base class for collections of elements.

Defined in: OMContainer.h

Class Template Arguments

Element
The type of an **OMContainer** element. This type must support = and ==.

Author

Tim Bingham - tjb - (Avid Technology, Inc.)

Class Members

Public members.
virtual size_t count(void) const
The number of elements in this **OMContainer. count** returns the actual number of elements in the **OMContainer.**
virtual void clear(void)
Remove all elements from this **OMContainer.**

OMContainerElement class

OMContainerElement **class OMContainerElement**

Elements of Object Manager reference containers.

Defined in: OMContainerElement.h

Class Template Arguments

ObjectReference
The type of the contained object reference

Author

Tim Bingham - tjb - (Avid Technology, Inc.)

Class Members

Public members.
OMContainerElement**(void)**
Constructor.
OMContainerElement**(const ObjectReference& reference)**
Constructor.
OMContainerElement**(const OMContainerElement&ltObjectReference>& rhs)**
Copy constructor.
~OMContainerElement**(void)**
Destructor.
OMContainerElement&ltObjectReference>& operator=
Assignment. This operator provides value semantics for OMContainer. This operator does not provide assignment of object references.
bool operator==**(const OMContainerElement&ltObjectReference>& rhs) const**
Equality. This operator provides value semantics for OMContainer. This operator does not provide equality of object references.
ObjectReference& reference**(void)**
The contained ObjectReference.
void setReference**(const ObjectReference& reference)**
Set the contained ObjectReference.
void save**(void)**
Save this **OMContainerElement.**
void close**(void)**

Close this **OMContainerElement.**

void detach**(void)**

Detach this **OMContainerElement.**

void restore**(void)**

Restore this **OMContainerElement.**

OMStorable* getValue**(void) const**

Get the value of this **OMContainerElement.**

OMStorable* pointer**(void) const**

The value of this **OMContainerElement** as a pointer. This function provides low-level access. If the object exits but has not yet been loaded then the value returned is 0.

Class Members

Protected members.

ObjectReference _reference

The actual object reference.

OMContainerElement::close

template <class *ObjectReference***>**
void OMContainerElement<*ObjectReference***>::close(void)**

Close this OMContainerElement.

Defined in: OMContainerElementT.h

Class Template Arguments

ObjectReference

The type of the contained object reference

Back to OMContainerElement

OMContainerElement::detach

template <class *ObjectReference***>**
void OMContainerElement<*ObjectReference***>::detach(void)**

Detach this OMContainerElement.

Defined in: OMContainerElementT.h

Class Template Arguments

ObjectReference

The type of the contained object reference

Back to OMContainerElement

OMContainerElement::getValue

template <class *ObjectReference***>**
OMStorable* OMContainerElement<*ObjectReference***>::getValue(void) const**

Get the value of this OMContainerElement.

Defined in: OMContainerElementT.h

Return Value

A pointer to the *ReferencedObject*.

Class Template Arguments

ObjectReference
The type of the contained object reference

Back to OMContainerElement

OMContainerElement::OMContainerElement

template <class *ObjectReference***>**
OMContainerElement<*ObjectReference***>::OMContainerElement(const OMContainerElement&ltObjectReference>&** *rhs***)**

Copy constructor.

Defined in: OMContainerElementT.h

Parameters

rhs
The OMContainerElement to copy.

Class Template Arguments

ObjectReference
The type of the contained object reference

Back to OMContainerElement

OMContainerElement::OMContainerElement

template <class *ObjectReference***>**
OMContainerElement<*ObjectReference***>::OMContainerElement(const ObjectReference&** *reference***)**

Constructor.

Defined in: OMContainerElementT.h

Parameters

reference
The *ObjectReference* for this OMContainerElement.

Class Template Arguments

ObjectReference
The type of the contained object reference

Back to OMContainerElement

OMContainerElement::OMContainerElement

template <class *ObjectReference***>**
OMContainerElement<*ObjectReference***>::OMContainerElement(void)**

Constructor.

Defined in: OMContainerElementT.h

Class Template Arguments

ObjectReference
The type of the contained object reference

Back to OMContainerElement

OMContainerElement::operator=

template <class *ObjectReference***>**
OMContainerElement&ltObjectReference>& OMContainerElement<*ObjectReference***>::operator=(const OMContainerElement&ltObjectReference>&** *rhs***)**

Assignment. This operator provides value semantics for OMContainer. This operator does not provide assignment of object references.

Defined in: OMContainerElementT.h

Return Value

The OMContainerElement resulting from the assignment.

Parameters

rhs
The OMContainerElement to be assigned.

Class Template Arguments

ObjectReference
The type of the contained object reference

Back to OMContainerElement

OMContainerElement::operator==

template <class *ObjectReference***>**
bool OMContainerElement<*ObjectReference***>::operator==(const OMContainerElement&ltObjectReference>&** *rhs***)**

Equality. This operator provides value semantics for OMContainer. This operator does not provide equality of object references.

Defined in: OMContainerElementT.h

Return Value

True if the values are the same, false otherwise.

Parameters

rhs
The OMContainerElement to be compared.

Class Template Arguments

ObjectReference
The type of the contained object reference

Back to OMContainerElement

OMContainerElement::pointer

template <class *ObjectReference***>**
OMStorable* OMContainerElement<*ObjectReference***>::pointer(void)**

The value of this OMContainerElement as a pointer. This function provides low-level access. If the object exits but has not yet been loaded then the value returned is 0.

Defined in: OMContainerElementT.h

Return Value

A pointer to the reference OMStorable, if loaded.

Class Template Arguments

ObjectReference
The type of the contained object reference
Back to OMContainerElement

OMContainerElement::reference

template <class *ObjectReference***>**
ObjectReference& OMContainerElement<*ObjectReference***>::reference(void)**

The contained ObjectReference.

Defined in: OMContainerElementT.h

Return Value

The contained *ObjectReference*.

Class Template Arguments

ObjectReference
The type of the contained object reference
Back to OMContainerElement

OMContainerElement::restore

template <class *ObjectReference***>**
void OMContainerElement<*ObjectReference***>::restore(void)**

Restore this OMContainerElement.

Defined in: OMContainerElementT.h

Class Template Arguments

ObjectReference
The type of the contained object reference
Back to OMContainerElement

OMContainerElement::save

template <class *ObjectReference***>**
void OMContainerElement<*ObjectReference***>::save(void)**

Save this OMContainerElement.

Defined in: OMContainerElementT.h

Class Template Arguments

ObjectReference
The type of the contained object reference

Back to OMContainerElement

OMContainerElement::setReference

template <class *ObjectReference***>**
void OMContainerElement<*ObjectReference***>::setReference(const ObjectReference&** *reference***)**

Set the contained ObjectReference.

Defined in: OMContainerElementT.h

Parameters

reference
The new contained *ObjectReference*.

Class Template Arguments

ObjectReference
The type of the contained object reference

Back to OMContainerElement

OMContainerElement::~OMContainerElement

template <class *ObjectReference***>**
OMContainerElement<*ObjectReference***>::~OMContainerElement(void)**

Destructor.

Defined in: OMContainerElementT.h

Class Template Arguments

ObjectReference
The type of the contained object reference

Back to OMContainerElement

OMContainerIterator class

OMContainerIterator **class OMContainerIterator**

Abstract base class for iterators over Object Manager containers. The elements of an Object Manager container have a well defined order. An Object Manager container may be traversed in either the forward or reverse direction.

Defined in: OMContainerIterator.h

Class Template Arguments

Element
The type of the contained elements.

Author

Tim Bingham - tjb - (Avid Technology, Inc.)

Class Members

Public members.

virtual void reset(OMIteratorPosition initialPosition)
Reset this **OMContainerIterator** to the given *initialPosition*. If *initialPosition* is specified as OMBefore then this **OMContainerIterator** is made ready to traverse the associated OMContainer in the forward direction. If *initialPosition* is specified as OMAfter then this **OMContainerIterator** is made ready to traverse the associated OMContainer in the reverse direction.

virtual bool before(void) const
Is this **OMContainerIterator** positioned before the first *Element* ?

virtual bool after(void) const
Is this **OMContainerIterator** positioned after the last *Element* ?

virtual bool valid(void) const
Is this **OMContainerIterator** validly positioned on an *Element* ?

virtual bool operator++()
Advance this **OMContainerIterator** to the next *Element*, if any. If the end of the associated OMContainer is not reached then the result is **true** , **valid** becomes **true** and **after** becomes **false** . If the end of the associated OMContainer is reached then the result is **false** , **valid** becomes **false** and **after** becomes **true** .

virtual bool operator--()
Retreat this **OMContainerIterator** to the previous *Element*, if any. If the beginning of the associated OMContainer is not reached then the result is **true** , **valid** becomes **true** and **before** becomes **false** . If the beginning of the associated OMContainer is reached then the result is **false** , **valid** becomes **false** and **before** becomes **true** .

virtual Element& value(void) const
Return the *Element* in the associated OMContainer at the position currently designated by this **OMContainerIterator**.

OMContainerProperty class

OMContainerProperty **class OMContainerProperty: public OMProperty**

Abstract base class for persistent object reference container properties supported by the Object Manager.

Defined in: OMContainerProperty.h

Author

Tim Bingham - tjb - (Avid Technology, Inc.)

Class Members

Public members.

OMContainerProperty**(const OMPropertyId propertyId, const OMStoredForm storedForm, const wchar_t* name)**
: Constructor.

virtual ~OMContainerProperty**(void)**
: Destructor.

virtual OMReferenceContainer* referenceContainer(void)
: Convert to OMReferenceContainer.

OMUInt32 localKey**(void) const**
: The current local key.

void setLocalKey**(OMUInt32 newLocalKey)**
: Set the current local key. Used on restore to restart local key assignment.

Class Members

Protected members.

wchar_t* elementName**(OMUInt32 localKey)**
: Compute the name of an element in this **OMContainter** given the element's *localKey*.

OMUInt32 nextLocalKey**(void)**
: Obtain the next available local key.

Class Members

Private members.

OMUInt32 _localKey
: The next available local key.

OMContainerProperty::elementName

wchar_t* OMContainerProperty::elementName(OMUInt32 *localKey***)**

Compute the name of an element in this OMContainer given the element's *localKey*.

Defined in: OMContainerProperty.cpp

Parameters

localKey
: The element's local key.

Back to OMContainerProperty

OMContainerProperty::localKey

OMUInt32 OMContainerProperty::localKey(void) const

The current local key.

Defined in: OMContainerProperty.cpp

Return Value

The current local key.

Back to OMContainerProperty

OMContainerProperty::nextLocalKey

OMUInt32 OMContainerProperty::nextLocalKey(void)

Obtain the next available local key.

Defined in: OMContainerProperty.cpp

Return Value

The next available local key.

Back to OMContainerProperty

OMContainerProperty::OMContainerProperty

OMContainerProperty::OMContainerProperty(const OMPropertyId *propertyId*, **const OMStoredForm** *storedForm*, **const wchar_t*** *name***)**

Constructor.

Defined in: OMContainerProperty.cpp

Parameters

propertyId
: The property id.

storedForm
: The stored form of this property.

name

The name of this property.

Back to OMContainerProperty

OMContainerProperty::setLocalKey

void OMContainerProperty::setLocalKey(OMUInt32 *newLocalKey***)**

Set the current local key. Used on restore to restart local key assignment.

Defined in: OMContainerProperty.cpp

Parameters

newLocalKey
The new local key.

Back to OMContainerProperty

OMContainerProperty::~OMContainerProperty

OMContainerProperty::~OMContainerProperty(void)

Destructor.

Defined in: OMContainerProperty.cpp

Back to OMContainerProperty

OMDataStream class

OMDataStream **class OMDataStream: public OMProperty.**

Persistent data stream properties supported by the Object Manager.

Defined in: OMDataStream.h

Author

Tim Bingham - tjb - (Avid Technology, Inc.)

Class Members

Public members.

OMDataStream**(const OMPropertyId propertyId, const wchar_t* name)**
Constructor.

~OMDataStream**(void)**
Destructor.

virtual OMUInt64 size(void) const

The size, in bytes, of the data in this OMDataStreamProperty.

virtual void setPosition(const OMUInt64 offset) const

Set the current position for **read()** and **write()**, as an offset in bytes from the begining of the data stream.

virtual void read(OMByte* buffer, const OMUInt32 bytes, OMUInt32& bytesRead) const

Attempt to read the number of bytes given by *bytes* from the data stream into the buffer at address *buffer*. The actual number of bytes read is returned in *bytesRead*.

OMDataStream::OMDataStream

OMDataStream::OMDataStream(void)

Constructor.

Defined in: OMDataStream.cpp

Back to OMDataStream

OMDataStream::~OMDataStream

OMDataStream::~OMDataStream(void)

Destructor.

Defined in: OMDataStream.cpp

Back to OMDataStream

OMDataStreamProperty class

OMDataStreamProperty **class OMDataStreamProperty: public OMDataStream.**

Persistent data stream properties supported by the Object Manager.

Defined in: OMDataStreamProperty.h

Author

Tim Bingham - tjb - (Avid Technology, Inc.)

Class Members

Public members.

OMDataStreamProperty(const OMPropertyId propertyId, const wchar_t* name)

Constructor.

virtual ~OMDataStreamProperty(void)

Destructor.

virtual void save**(void) const**

Save this **OMDataStreamProperty.**

virtual void restore**(size_t externalSize)**

Restore this **OMDataStreamProperty**, the size of the **OMDataStreamProperty** is *externalSize*.

virtual void close(void)

Close this **OMDataStreamProperty.**

virtual OMUInt64 size**(void) const**

The size, in bytes, of the data in this **OMDataStreamProperty.**

void setSize**(const OMUInt64 newSize)**

Set the size, in bytes, of the data in this **OMDataStreamProperty.**

OMUInt64 position**(void) const**

The current position for **read()** and **write()**, as an offset in bytes from the begining of the data stream.

virtual void setPosition**(const OMUInt64 offset) const**

Set the current position for **read()** and **write()**, as an offset in bytes from the begining of the data stream.

virtual void read**(OMByte* buffer, const OMUInt32 bytes, OMUInt32& bytesRead) const**

Attempt to read the number of bytes given by *bytes* from the data stream into the buffer at address *buffer*. The actual number of bytes read is returned in *bytesRead*.

void write**(const OMByte* buffer, const OMUInt32 bytes, OMUInt32& bytesWritten)**

Attempt to write the number of bytes given by *bytes* to the data stream from the buffer at address *buffer*. The actual number of bytes written is returned in *bytesWritten*.

void readTypedElements**(const OMType* elementType, size_t externalElementSize, OMByte* elements, OMUInt32 elementCount, OMUInt32& elementsRead) const**

Attempt to read the number of elements given by *elementCount* and described by *elementType* and *externalElementSize* from the data stream into the buffer at address *elements*. The actual number of elements read is returned in *elementsRead*.

void writeTypedElements**(const OMType* elementType, size_t internalElementSize, const OMByte* elements, OMUInt32 elementCount, OMUInt32& elementsWritten)**

Attempt to write the number of elements given by *elementCount* and described by *elementType* and *internalElementSize* to the data stream from the buffer at address *elements*. The actual number of elements written is returned in *elementsWritten*.

virtual size_t bitsSize**(void) const**

The size of the raw bits of this **OMDataStreamProperty**. The size is given in bytes.

virtual void getBits**(OMByte* bits, size_t size) const**

Get the raw bits of this **OMDataStreamProperty**. The raw bits are copied to the buffer at address *bits* which is *size* bytes in size.

virtual void setBits**(const OMByte* bits, size_t size)**

Set the raw bits of this **OMDataStreamProperty**. The raw bits are copied from the buffer at address *bits* which is *size* bytes in size.

virtual bool hasByteOrder**(void) const**

Is a byte order specifed for this stream ?

virtual void setByteOrder**(OMByteOrder byteOrder)**

Specify a byte order for this stream.

virtual OMByteOrder byteOrder**(void) const**

The byte order of this stream.

virtual void clearByteOrder**(void)**

Clear the byte order of this stream

OMDataStreamProperty::bitsSize

size_t OMDataStreamProperty::bitsSize(void) const

The size of the raw bits of this OMDataStreamProperty. The size is given in bytes.

Defined in: OMDataStreamProperty.cpp

Return Value

The size of the raw bits of this OMDataStreamProperty in bytes.

Back to OMDataStreamProperty

OMDataStreamProperty::byteOrder

OMByteOrder OMDataStreamProperty::byteOrder(void)

The byte order of this stream.

Defined in: OMDataStreamProperty.cpp

Back to OMDataStreamProperty

OMDataStreamProperty::clearByteOrder

void OMDataStreamProperty::clearByteOrder(void)

Clear the byte order of this stream

Defined in: OMDataStreamProperty.cpp

Back to OMDataStreamProperty

OMDataStreamProperty::getBits

void OMDataStreamProperty::getBits(OMByte* *bits*, **size_t** *size***) const**

Get the raw bits of this OMDataStreamProperty. The raw bits are copied to the buffer at address *bits* which is *size* bytes in size.

Defined in: OMDataStreamProperty.cpp

Parameters

bits
The address of the buffer into which the raw bits are copied.
size
The size of the buffer.
Back to OMDataStreamProperty

OMDataStreamProperty::hasByteOrder

bool OMDataStreamProperty::hasByteOrder(void)

Is a byte order specifed for this stream ?

Defined in: OMDataStreamProperty.cpp

Back to OMDataStreamProperty

OMDataStreamProperty::position

OMUInt64 OMDataStreamProperty::position(void) const

The current position for **read()** and **write()**, as an offset in bytes from the begining of the data stream.

Defined in: OMDataStreamProperty.cpp

Return Value

The current position for **read()** and **write()**, as an offset in bytes from the begining of the data stream.

Back to OMDataStreamProperty

OMDataStreamProperty::read

void OMDataStreamProperty::read(OMByte* *buffer***, const OMUInt32** *bytes***, OMUInt32&** *bytesRead***) const**

Attempt to read the number of bytes given by *bytes* from the data stream into the buffer at address *buffer*. The actual number of bytes read is returned in *bytesRead*.

Defined in: OMDataStreamProperty.cpp

Parameters

buffer
The address of the buffer into which the bytes should be read.
bytes

The number of bytes to read.

bytesRead

The actual number of bytes that were read.

Back to OMDataStreamProperty

OMDataStreamProperty::readTypedElements

void OMDataStreamProperty::readTypedElements(const OMType* *elementType*, **size_t** *externalElementSize*, **OMByte*** *elements*, **OMUInt32** *elementCount*, **OMUInt32&** *elementsRead***) const**

Attempt to read the number of elements given by *elementCount* and described by *elementType* and *externalElementSize* from the data stream into the buffer at address *elements*. The actual number of elements read is returned in *elementsRead*.

Defined in: OMDataStreamProperty.cpp

Parameters

elementType

The element type

externalElementSize

The external element size

elements

The address of the buffer into which the elements should be read.

elementCount

The number of elements to read.

elementsRead

The actual number of elements that were read.

Back to OMDataStreamProperty

OMDataStreamProperty::restore

void OMDataStreamProperty::restore(size_t *externalSize***)**

Restore this OMDataStreamProperty, the size of the OMDataStreamProperty is *externalSize*.

Defined in: OMDataStreamProperty.cpp

Parameters

externalSize

The size of the OMDataStreamProperty.

Back to OMDataStreamProperty

OMDataStreamProperty::save

void OMDataStreamProperty::save(void) const

Save this OMDataStreamProperty.

Defined in: OMDataStreamProperty.cpp

Back to OMDataStreamProperty

OMDataStreamProperty::setBits

void OMDataStreamProperty::setBits(const OMByte* *bits*, **size_t** *size***)**

Set the raw bits of this OMDataStreamProperty. The raw bits are copied from the buffer at address *bits* which is *size* bytes in size.

Defined in: OMDataStreamProperty.cpp

Parameters

bits
: The address of the buffer from which the raw bits are copied.

size
: The size of the buffer.

Back to OMDataStreamProperty

OMDataStreamProperty::setByteOrder

void OMDataStreamProperty::setByteOrder(void)

Specify a byte order for this stream.

Defined in: OMDataStreamProperty.cpp

Back to OMDataStreamProperty

OMDataStreamProperty::setPosition

void OMDataStreamProperty::setPosition(const OMUInt64 *offset***) const**

Set the current position for **read()** and **write()**, as an offset in bytes from the begining of the data stream.

Defined in: OMDataStreamProperty.cpp

Parameters

offset

The position to use for subsequent calls to read() and write() on this stream. The position is specified as an offset in bytes from the begining of the data stream.

Back to OMDataStreamProperty

OMDataStreamProperty::setSize

void OMDataStreamProperty::setSize(void)

Set the size, in bytes, of the data in this OMDataStreamProperty.

Defined in: OMDataStreamProperty.cpp

Return Value

The new size, in bytes, of the data in this OMDataStreamProperty.

Back to OMDataStreamProperty

OMDataStreamProperty::size

OMUInt64 OMDataStreamProperty::size(void) const

The size, in bytes, of the data in this OMDataStreamProperty.

Defined in: OMDataStreamProperty.cpp

Return Value

The size, in bytes, of the data in this OMDataStreamProperty.

Back to OMDataStreamProperty

OMDataStreamProperty::write

void OMDataStreamProperty::write(const OMByte* *buffer*, **const OMUInt32** *bytes*, **OMUInt32&** *bytesWritten***)**

Attempt to write the number of bytes given by *bytes* to the data stream from the buffer at address *buffer*. The actual number of bytes written is returned in *bytesWritten*.

Defined in: OMDataStreamProperty.cpp

Parameters

buffer
The address of the buffer from which the bytes should be written.
bytes
The number of bytes to write.
bytesWritten
The actual number of bytes that were written.
Back to OMDataStreamProperty

OMDataStreamProperty::writeTypedElements

void OMDataStreamProperty::writeTypedElements(const OMType* *elementType*, **size_t** *internalElementSize*, **const OMByte*** *elements*, **OMUInt32** *elementCount*, **OMUInt32&** *elementsWritten***)**

Attempt to write the number of elements given by *elementCount* and described by *elementType* and *internalElementSize* to the data stream from the buffer at address *elements*. The actual number of elements written is returned in *elementsWritten*.

Defined in: OMDataStreamProperty.cpp

Parameters

elementType
The element type
internalElementSize
The internal element size
elements
The address of the buffer from which the elements should be written.
elementCount
The number of elements to write.
elementsWritten
The actual number of elements that were written.
Back to OMDataStreamProperty

OMDefinition class

OMDefinition **class OMDefinition**

Abstract base class used to define persistent entities supported by the Object Manager.

Defined in: OMDefinition.h

Author

Tim Bingham - tjb - (Avid Technology, Inc.)

Class Members

Public members.
virtual ~OMDefinition(void)
Destructor.
virtual const OMUniqueObjectIdentification& identification(void) const
The unique identification of the entity defined by this **OMDefinition.**
virtual const wchar_t* name(void) const
The name of the entity defined by this **OMDefinition.**

OMDiskRawStorage class

OMDiskRawStorage **class OMDiskRawStorage: public OMRawStorage**

Class supporting access to the raw bytes of disk files supported by the Object Manager.

This is an Object Manager built-in implementation of the OMRawStorage interface. The implementation uses ANSI file functions only.

Defined in: OMDiskRawStorage.h

Author

Tim Bingham - tjb - (Avid Technology, Inc.)

Class Members

Static members.
static OMDiskRawStorage* openExistingRead**(const wchar_t* fileName)**
Create an **OMDiskRawStorage** object by opening an existing file for read-only access, the file is named *fileName*. The file must already exist.
static OMDiskRawStorage* openExistingModify**(const wchar_t* fileName)**
Create an **OMDiskRawStorage** object by opening an existing file for modify access, the file is named *fileName*. The file must already exist.
static OMDiskRawStorage* openNewModify**(const wchar_t* fileName)**
Create an **OMDiskRawStorage** object by creating a new file for modify access, the file is named *fileName*. The file must not already exist.

Class Members

Public members.
virtual ~OMDiskRawStorage**(void)**
Destructor.
virtual bool isReadable**(void) const**
Is it possible to read from this **OMDiskRawStorage** ?
virtual void read**(OMByte* bytes, OMUInt32 byteCount, OMUInt32& bytesRead) const**
Attempt to read the number of bytes given by *byteCount* from the current position in this **OMDiskRawStorage** into the buffer at address *bytes*. The actual number of bytes read is returned in *bytesRead*. Reading from positions greater than **size** causes *bytesRead* to be less than *byteCount*. Reading bytes that have never been written returns undefined data in *bytes*.
virtual void readAt**(OMUInt64 position, OMByte* bytes, OMUInt32 byteCount, OMUInt32& bytesRead) const**

Attempt to read the number of bytes given by *byteCount* from offset *position* in this **OMDiskRawStorage** into the buffer at address *bytes*. The actual number of bytes read is returned in *bytesRead*. Reading from positions greater than **size** causes *bytesRead* to be less than *byteCount*. Reading bytes that have never been written returns undefined data in *bytes*.

preconditions

isReadable() && **isPositionable()**

virtual bool isWritable**(void) const**

Is it possible to write to this **OMDiskRawStorage** ?

virtual void write**(const OMByte* bytes, OMUInt32 byteCount, OMUInt32& bytesWritten)**

Attempt to write the number of bytes given by *byteCount* to the current position in this **OMDiskRawStorage** from the buffer at address *bytes*. The actual number of bytes written is returned in *bytesWritten*. Writing to positions greater than **size** causes this **OMDiskRawStorage** to be extended, however such extension can fail, causing *bytesWritten* to be less than *byteCount*.

virtual void writeAt**(OMUInt64 position, const OMByte* bytes, OMUInt32 byteCount, OMUInt32& bytesWritten)**

Attempt to write the number of bytes given by *byteCount* to offset *position* in this **OMDiskRawStorage** from the buffer at address *bytes*. The actual number of bytes written is returned in *bytesWritten*. Writing to positions greater than **size** causes this **OMDiskRawStorage** to be extended, however such extension can fail, causing *bytesWritten* to be less than *byteCount*.

preconditions

isWritable() && **isPositionable()**

Developer Notes

How is failure to extend indicated ?

virtual bool isExtendible**(void) const**

May this **OMDiskRawStorage** be changed in size ?

virtual OMUInt64 extent**(void) const**

The current extent of this **OMDiskRawStorage** in bytes. The **extent()** is the allocated size, while the **size()** is the valid size. precondition - isPositionable()

virtual void extend**(OMUInt64 newSize)**

Set the size of this **OMDiskRawStorage** to *newSize* bytes. If *newSize* is greater than **size** then this **OMDiskRawStorage** is extended. If *newSize* is less than **size** then this **OMDiskRawStorage** is truncated. Truncation may also result in the current position for **read()** and **write()** being set to **size**. precondition - isExtendible()

virtual OMUInt64 size**(void) const**

The current size of this **OMDiskRawStorage** in bytes. The **size()** is the valid size, while the **extent()** is the allocated size. precondition - isPositionable()

virtual bool isPositionable**(void) const**

May the current position, for **read()** and **write()**, of this **OMDiskRawStorage** be changed ?

virtual OMUInt64 position**(void) const**

The current position for **read()** and **write()**, as an offset in bytes from the beginning of this **OMDiskRawStorage**. precondition - isPositionable()

virtual void setPosition**(OMUInt64 newPosition) const**

Set the current position for **read()** and **write()**, as an offset in bytes from the beginning of this **OMDiskRawStorage**. precondition - isPositionable()

virtual void synchronize**(void)**

Synchronize this **OMDiskRawStorage** with its external representation.

Class Members

Protected members.

OMDiskRawStorage**(FILE* file, OMFile::OMAccessMode accessMode)**

Constructor.

Class Members

Private members.

OMDiskRawStorage::extend

void OMDiskRawStorage::extend(OMUInt64 *newSize***)**

Set the size of this OMDiskRawStorage to *newSize* bytes. If *newSize* is greater than **size** then this OMDiskRawStorage is extended. If *newSize* is less than **size** then this OMDiskRawStorage is truncated. Truncation may also result in the current position for **read()** and **write()** being set to **size**. precondition - isExtendible()

Defined in: OMDiskRawStorage.cpp

Parameters

newSize

The new size of this OMDiskRawStorage in bytes.

Developer Notes

There is no ISO/ANSI way of truncating a file in place.

Back to OMDiskRawStorage

OMDiskRawStorage::extent

OMUInt64 OMDiskRawStorage::extent(void) const

The current extent of this OMDiskRawStorage in bytes. precondition - isPositionable()

Defined in: OMDiskRawStorage.cpp

Return Value

The current extent of this OMDiskRawStorage in bytes.

Back to OMDiskRawStorage

OMDiskRawStorage::isExtendible

bool OMDiskRawStorage::isExtendible(void) const

May this OMDiskRawStorage be changed in size ?

Defined in: OMDiskRawStorage.cpp

Return Value

Always **true** .

Back to OMDiskRawStorage

OMDiskRawStorage::isPositionable

bool OMDiskRawStorage::isPositionable(void) const

May the current position, for **read()** and **write()**, of this OMDiskRawStorage be changed ?

Defined in: OMDiskRawStorage.cpp

Return Value

Always **true** .

Back to OMDiskRawStorage

OMDiskRawStorage::isReadable

bool OMDiskRawStorage::isReadable(void) const

Is it possible to read from this OMDiskRawStorage ?

Defined in: OMDiskRawStorage.cpp

Return Value

True if this OMDiskRawStorage is readable, false otherwise.

Back to OMDiskRawStorage

OMDiskRawStorage::isWritable

bool OMDiskRawStorage::isWritable(void) const

Is it possible to write to this OMDiskRawStorage ?

Defined in: OMDiskRawStorage.cpp

Return Value

True if this OMDiskRawStorage is writable, false otherwise.

Back to OMDiskRawStorage

OMDiskRawStorage::OMDiskRawStorage

OMDiskRawStorage::OMDiskRawStorage(FILE* *file*, OMFile::OMAccessMode *accessMode*)

Constructor.

Defined in: OMDiskRawStorage.cpp

Parameters

file
: The file.

accessMode
: The access mode.

Back to OMDiskRawStorage

OMDiskRawStorage::openExistingModify

OMDiskRawStorage* OMDiskRawStorage::openExistingModify(const wchar_t* *fileName*)

Create an OMDiskRawStorage object by opening an existing file for modify access, the file is named *fileName*. The file must already exist.

Defined in: OMDiskRawStorage.cpp

Return Value

The newly created OMDiskRawStorage object.

Parameters

fileName
The file name.
Back to OMDiskRawStorage

OMDiskRawStorage::openExistingRead

OMDiskRawStorage* OMDiskRawStorage::openExistingRead(const wchar_t* *fileName*)

Create an OMDiskRawStorage object by opening an existing file for read-only access, the file is named *fileName*. The file must already exist.

Defined in: OMDiskRawStorage.cpp

Return Value

The newly created OMDiskRawStorage object.

Parameters

fileName
The file name.
Back to OMDiskRawStorage

OMDiskRawStorage::openNewModify

OMDiskRawStorage* OMDiskRawStorage::openNewModify(const wchar_t* *fileName*)

Create an OMDiskRawStorage object by creating a new file for modify access, the file is named *fileName*. The file must not already exist.

Defined in: OMDiskRawStorage.cpp

Return Value

The newly created OMDiskRawStorage object.

Parameters

fileName
The file name.
Back to OMDiskRawStorage

OMDiskRawStorage::position

OMUInt64 OMDiskRawStorage::position(void) const

The current position for **read()** and **write()**, as an offset in bytes from the beginning of this OMDiskRawStorage.
precondition - isPositionable()

Defined in: OMDiskRawStorage.cpp

Return Value

The current position for **read()** and **write()**.

Back to OMDiskRawStorage

OMDiskRawStorage::read

void OMDiskRawStorage::read(OMByte* *bytes*, **OMUInt32** *byteCount*, **OMUInt32&** *bytesRead***) const**

Attempt to read the number of bytes given by *byteCount* from the current position in this OMDiskRawStorage into the buffer at address *bytes*. The actual number of bytes read is returned in *bytesRead*. Reading from positions greater than **size** causes *bytesRead* to be less than *byteCount*. Reading bytes that have never been written returns undefined data in *bytes*.

Defined in: OMDiskRawStorage.cpp

Parameters

bytes
: The buffer into which the bytes are to be read.

byteCount
: The number of bytes to read.

bytesRead
: The number of bytes actually read.

Back to OMDiskRawStorage

OMDiskRawStorage::readAt

void OMDiskRawStorage::readAt(OMUInt64 *position*, **OMByte*** *bytes*, **OMUInt32** *byteCount*, **OMUInt32&** *bytesRead***) const**

Attempt to read the number of bytes given by *byteCount* from offset *position* in this OMDiskRawStorage into the buffer at address *bytes*. The actual number of bytes read is returned in *bytesRead*. Reading from positions greater than **size** causes *bytesRead* to be less than *byteCount*. Reading bytes that have never been written returns undefined data in *bytes*.

Defined in: OMDiskRawStorage.cpp

Parameters

position

The position from which the bytes are to be read.

bytes

The buffer into which the bytes are to be read.

byteCount

The number of bytes to read.

bytesRead

The number of bytes actually read.

Back to OMDiskRawStorage

OMDiskRawStorage::setPosition

void OMDiskRawStorage::setPosition(OMUInt64 *newPosition***) const**

Set the current position for **read()** and **write()**, as an offset in bytes from the beginning of this OMDiskRawStorage. precondition - isPositionable()

Defined in: OMDiskRawStorage.cpp

Parameters

newPosition

The new position.

Developer Notes

fseek takes a long int for offset this may not be sufficient for 64-bit offsets.

Back to OMDiskRawStorage

OMDiskRawStorage::size

OMUInt64 OMDiskRawStorage::size(void) const

The current size of this OMDiskRawStorage in bytes. precondition - isPositionable()

Defined in: OMDiskRawStorage.cpp

Return Value

The current size of this OMDiskRawStorage in bytes.

Back to OMDiskRawStorage

OMDiskRawStorage::synchronize

void OMDiskRawStorage::synchronize(void)

Synchronize this OMDiskRawStorage with its external representation.

Defined in: OMDiskRawStorage.cpp

Back to OMDiskRawStorage

OMDiskRawStorage::write

void OMDiskRawStorage::write(const OMByte* *bytes*, **OMUInt32** *byteCount*, **OMUInt32&** *bytesWritten***)**

Attempt to write the number of bytes given by *byteCount* to the current position in this OMDiskRawStorage from the buffer at address *bytes*. The actual number of bytes written is returned in *bytesWritten*. Writing to positions greater than **size** causes this OMDiskRawStorage to be extended, however such extension can fail, causing *bytesWritten* to be less than *byteCount*.

Defined in: OMDiskRawStorage.cpp

Parameters

bytes
: The buffer from which the bytes are to be written.

byteCount
: The number of bytes to write.

bytesWritten
: The actual number of bytes written.

Back to OMDiskRawStorage

OMDiskRawStorage::writeAt

void OMDiskRawStorage::writeAt(OMUInt64 *position*, **const OMByte*** *bytes*, **OMUInt32** *byteCount*, **OMUInt32&** *bytesWritten***)**

Attempt to write the number of bytes given by *byteCount* to offset *position* in this OMDiskRawStorage from the buffer at address *bytes*. The actual number of bytes written is returned in *bytesWritten*. Writing to positions greater than **size** causes this OMDiskRawStorage to be extended, however such extension can fail, causing *bytesWritten* to be less than *byteCount*.

Defined in: OMDiskRawStorage.cpp

Parameters

position
: The position to which the bytes are to be written.

bytes
: The buffer from which the bytes are to be written.

byteCount

The number of bytes to write.

bytesWritten

The actual number of bytes written.

Back to OMDiskRawStorage

OMDiskRawStorage::~OMDiskRawStorage

OMDiskRawStorage::~OMDiskRawStorage(void)

Destructor.

Defined in: OMDiskRawStorage.cpp

Back to OMDiskRawStorage

OMFile class

OMFile **class OMFile: public OMStorable**

Files supported by the Object Manager.

Defined in: OMFile.h

Author

Tim Bingham - tjb - (Avid Technology, Inc.)

enum OMAccessMode

File access modes.

enum OMLoadMode

Lazy loading modes (degrees of indolence).

enum OMFileEncoding

Supported file encodings.

Class Members

Static members.

static OMFile* openExistingRead**(const wchar_t* fileName, const OMClassFactory* factory, void* clientOnRestoreContext, const OMLoadMode loadMode, OMDictionary* dictionary = 0)**

Open an existing **OMFile** for read-only access, the **OMFile** is named *fileName*, use the OMClassFactory *factory* to create the objects. The file must already exist.

static OMFile* openExistingModify**(const wchar_t* fileName, const OMClassFactory* factory, void* clientOnRestoreContext, const OMLoadMode loadMode, OMDictionary* dictionary = 0)**

Open an existing **OMFile** for modify access, the **OMFile** is named *fileName*, use the OMClassFactory *factory* to create the objects. The file must already exist.

static OMFile* openNewModify**(const wchar_t* fileName, const OMClassFactory* factory, void* clientOnRestoreContext, const OMByteOrder byteOrder, OMStorable* clientRoot, const OMFileSignature& signature, OMDictionary* dictionary = 0)**

Open a new **OMFile** for modify access, the **OMFile** is named *fileName*, use the OMClassFactory *factory* to create the objects. The file must not already exist. The byte ordering on the newly created file is given by *byteOrder*. The client root OMStorable in the newly created file is given by *clientRoot*.

static bool compatibleRawStorage**(const OMRawStorage* rawStorage, const OMAccessMode accessMode, const OMFileSignature& signature)**

Is the given OMRawStorage compatible with the given file access mode and signature ? Can a file of the encoding specified by *signature* be created successfully on *rawStorage* and then accessed successfully in the mode specified by *accessMode* ?

static OMFile* openExistingRead**(OMRawStorage* rawStorage, const OMClassFactory* factory, void* clientOnRestoreContext, const OMLoadMode loadMode, OMDictionary* dictionary = 0)**

Open an existing **OMFile** for read-only access.

Developer Notes

Will superceed openExistingRead() above.

static OMFile* openExistingModify**(OMRawStorage* rawStorage, const OMClassFactory* factory, void* clientOnRestoreContext, const OMLoadMode loadMode, OMDictionary* dictionary = 0)**

Open an existing **OMFile** for modify access.

Developer Notes

Will superceed openExistingModify() above.

static OMFile* openNewWrite(OMRawStorage* rawStorage, const OMClassFactory* factory, void* clientOnRestoreContext, const OMByteOrder byteOrder, OMStorable* clientRoot, const OMFileSignature& signature, OMDictionary* dictionary = 0)

Open a new **OMFile** for write access.

static OMFile* openNewModify**(OMRawStorage* rawStorage, const OMClassFactory* factory, void* clientOnRestoreContext, const OMByteOrder byteOrder, OMStorable* clientRoot, const OMFileSignature& signature, OMDictionary* dictionary = 0)**

Open a new **OMFile** for modify access.

Developer Notes

Will superceed openNewModify() above.

static bool validSignature**(const OMFileSignature& signature)**

Is *signature* a valid signature for an **OMFile** ?

static bool isRecognized**(const wchar_t* fileName, OMFileSignature& signature, OMFileEncoding& encoding)**

Is the file named *fileName* a recognized file ? If so, the result is true, and the signature is returned in *signature* and the encoding in *encoding*.

static bool isRecognized**(OMRawStorage* rawStorage, OMFileSignature& signature, OMFileEncoding& encoding)**

Does *rawStorage* contain a recognized file ? If so, the result is true, and the signature is returned in *signature* and the encoding in *encoding*.

static bool isRecognized**(const OMFileSignature& signature, OMFileEncoding& encoding)**

Is *signature* recognized ? If so, the result is true, and the encoding is returned in *encoding*.

Class Members

Public members.

~OMFile**(void)**

Destructor.

void saveFile**(void* clientOnSaveContext = 0)**

Save all changes made to the contents of this **OMFile**. It is not possible to save read-only or transient files.

preconditions

isOpen()

void saveAsFile**(OMFile* destFile) const**

Save the entire contents of this **OMFile** as well as any unsaved changes in the new file *destFile*. *destFile* must be open, writeable and not yet contain any objects. **saveAsFile** may be called for files opened in modify mode and for files opened in read-only and transient modes.

void revert**(void)**

Discard all changes made to this **OMFile** since the last **save** or **open**.

OMStorable* restore**(void)**

Restore the client root OMStorable object from this **OMFile**.

preconditions

isOpen()

void open**(void)**

Open this **OMFile**.

preconditions

!isOpen()
!isClosed()

postconditions

isOpen()

void close**(void)**

Close this **OMFile**, any unsaved changes are discarded.

preconditions

isOpen()

postconditions

!isOpen()
isClosed()

bool isOpen**(void) const**

Is this **OMFile** open ?

bool isClosed**(void) const**

Is this **OMFile** closed ? Note that **isClosed()** is not the same as !**isOpen()** since before open() is called, **isClosed()** is false. That is, **isClosed()** means "was once open and is now closed".

OMStorable* clientRoot**(void)**

Retrieve the client root OMStorable from this **OMFile**.

OMStorable* root(void)

Retrieve the root OMStorable from this **OMFile**.

OMPropertyTable* referencedProperties**(void)**

Retrieve the OMPropertyTable from this **OMFile.**

OMObjectDirectory* objectDirectory**(void)**

Retrieve the OMObjectDirectory from this **OMFile.**

OMByteOrder byteOrder**(void) const**

The byte order of this **OMFile.**

OMLoadMode loadMode**(void) const**

The loading mode (eager or lazy) of this **OMFile.**

OMAccessMode accessMode**(void) const**

The access mode of this **OMFile.**

bool isReadable(void) const

Is it possible to read from this **OMFile** ?

bool isWritable(void) const

Is it possible to write to this **OMFile** ?

bool isOMFile**(void) const**

Is this file recognized by the Object Manager ?

const wchar_t* fileName**(void) const**

The name of this **OMFile.**

Developer Notes

Soon to be obsolete.

OMFileSignature signature**(void) const**

The signature of this **OMFile.**

OMFileEncoding encoding**(void) const**

The encoding of this **OMFile.**

OMRawStorage* rawStorage**(void) const**

The raw storage on which this **OMFile** is stored.

virtual OMProperty* findPropertyPath**(const wchar_t* propertyPathName) const**

Find the property instance in this **OMFile** named by *propertyPathName*.

Class Members

Private members.

OMFile**(const wchar_t* fileName, void* clientOnRestoreContext, OMFileSignature signature, const OMAccessMode mode, OMStoredObject* store, const OMClassFactory* factory, OMDictionary* dictionary, const OMLoadMode loadMode)**

Constructor. Create an **OMFile** object representing an existing named external file.

OMFile**(const wchar_t* fileName, void* clientOnRestoreContext, OMFileSignature signature, const OMAccessMode mode, OMStoredObject* store, const OMClassFactory* factory, OMDictionary* dictionary, OMRootStorable* root)**

Constructor. Create an **OMFile** object representing a new named external file.

OMFile**(OMRawStorage* rawStorage, void* clientOnRestoreContext, const OMAccessMode mode, const OMClassFactory* factory, OMDictionary* dictionary, const OMLoadMode loadMode)**

Constructor. Create an **OMFile** object representing an existing external file on the given OMRawStorage**.**

Developer Notes

Will superceed OMFile::OMFile (for existing files) above.

OMFile**(OMRawStorage* rawStorage, void* clientOnRestoreContext, OMFileSignature signature, const OMAccessMode mode, const OMClassFactory* factory, OMDictionary* dictionary, OMRootStorable* root, const OMByteOrder byteOrder)**

Constructor. Create an **OMFile** object representing a new external file on the given OMRawStorage.

Developer Notes

Will superceed OMFile::OMFile (for new files) above.

OMFile::accessMode

OMFile::OMAccessMode OMFile::accessMode(void) const

The access mode of this OMFile.

Defined in: OMFile.cpp

Return Value

The access mode of this OMFile.

True if this OMFile is readable, false otherwise.

True if this OMFile is writable, false otherwise.

Class Members

bool OMFile:: isReadable(void) const
Is it possible to read from this OMFile ?

Class Members

bool OMFile:: isWritable(void) const
Is it possible to write to this OMFile ?

Back to OMFile

OMFile::byteOrder

OMByteOrder OMFile::byteOrder(void) const

The byte order of this OMFile.

Defined in: OMFile.cpp

Return Value

The byte order of this OMFile.

Back to OMFile

OMFile::clientRoot

OMStorable* OMFile::clientRoot(void)

Retrieve the client root OMStorable from this OMFile.

Defined in: OMFile.cpp

Return Value

The root OMStorable.

Back to OMFile

OMFile::close

void OMFile::close(void)

Close this OMFile, any unsaved changes are discarded.

Defined in: OMFile.cpp

preconditions

isOpen()

postconditions

!isOpen()
isClosed()

Back to OMFile

OMFile::compatibleRawStorage

bool OMFile::compatibleRawStorage(const OMRawStorage* *rawStorage*, **const OMAccessMode** *accessMode*, **const OMFileSignature&** *signature***)**

Is the given OMRawStorage compatible with the given file access mode and signature ? Can a file of the encoding specified by *signature* be created successfully on *rawStorage* and then accessed successfully in the mode specified by *accessMode* ?

Defined in: OMFile.cpp

Return Value

TBS

Parameters

rawStorage
The OMRawStorage on which the file is to be created.
accessMode
TBS
signature
TBS
Back to OMFile

OMFile::encoding

OMFile::OMFileEncoding OMFile::encoding(void)

The encoding of this OMFile.

Defined in: OMFile.cpp

Return Value

The encoding of the OMFile.

Back to OMFile

OMFile::fileName

const wchar_t* OMFile::fileName(void)

The name of this OMFile.

Defined in: OMFile.cpp

Return Value

The name of this OMFile.

Back to OMFile

OMFile::findPropertyPath

OMProperty* OMFile::findPropertyPath(const wchar_t* *propertyPathName***) const**

Find the property instance in this OMFile named by *propertyPathName*.

Defined in: OMFile.cpp

Return Value

The property instance.

Parameters

propertyPathName
The pathname to the desired property.

Back to OMFile

OMFile::isClosed

bool OMFile::isClosed(void)

Is this OMFile closed ? Note that **isClosed()** is not the same as !**isOpen()** since before open() is called, **isClosed()** is false. That is, **isClosed()** means "was once open and is now closed".

Defined in: OMFile.cpp

Back to OMFile

OMFile::isOMFile

bool OMFile::isOMFile(void) const

Is this file recognized by the Object Manager ?

Defined in: OMFile.cpp

Return Value

True if this file is recognized by the Object Manager, false otherwise.

Back to OMFile

OMFile::isOpen

bool OMFile::isOpen(void)

Is this OMFile open ?

Defined in: OMFile.cpp

Back to OMFile

OMFile::isRecognized

bool OMFile::isRecognized(const wchar_t* *fileName*, **OMFileSignature&** *signature*, **OMFileEncoding&** *encoding***)**

Is the file named *fileName* a recognized file ? If so, the result is true, and the signature is returned in *signature* and the encoding in *encoding*.

Defined in: OMFile.cpp

Return Value

True if the file is recognized, false otherwise.

Parameters

fileName
: The name of the file to check.

signature
: If recognized, the file signature.

encoding
: If recognized, the file encoding.

Back to OMFile

OMFile::isRecognized

bool OMFile::isRecognized(OMRawStorage* *rawStorage*, **OMFileSignature&** *signature*, **OMFileEncoding&** *encoding***)**

Does *rawStorage* contain a recognized file ? If so, the result is true, and the signature is returned in *signature* and the encoding in *encoding*.

Defined in: OMFile.cpp

Return Value

True if the OMRawStorage contains a recognized file, false otherwise.

Parameters

rawStorage
: The OMRawStorage to check.

signature
If recognized, the file signature.
encoding
If recognized, the file encoding.
Back to OMFile

OMFile::isRecognized

bool OMFile::isRecognized(const OMFileSignature& *signature*)

Is *signature* recognized ? If so, the result is true, and the encoding is returned in *encoding*.

Defined in: OMFile.cpp

Return Value

True if the signature is recognized, false otherwise.

Parameters

signature
If recognized, the encoding.
Back to OMFile

OMFile::loadMode

OMFile::OMLoadMode OMFile::loadMode(void) const

The loading mode (eager or lazy) of this OMFile.

Defined in: OMFile.cpp

Return Value

The loading mode (eager or lazy) of this OMFile.

Back to OMFile

OMFile::objectDirectory

OMObjectDirectory* OMFile::objectDirectory(void)

Retrieve the OMObjectDirectory from this OMFile.

Defined in: OMFile.cpp

Return Value

The OMObjectDirectory associated with this file.

Back to OMFile

OMFile::OMFile

OMFile::OMFile(void)

Constructor. Create an OMFile object representing an existing external file on the given OMRawStorage.

Defined in: OMFile.cpp

Back to OMFile

OMFile::OMFile

OMFile::OMFile(const wchar_t* *fileName*, **void*** *clientOnRestoreContext*, **OMFileSignature** *signature*, **const OMAccessMode** *mode*, **OMStoredObject*** *store***)**

Constructor. Create an OMFile object representing an existing named external file.

Defined in: OMFile.cpp

Parameters

fileName
: The name of this OMFile.

clientOnRestoreContext
: The access mode of this OMFile.

signature
: The OMStoredObject containing the root OMStorable object.

mode
: The OMClassFactory to use to restore objects from this OMFile.

store
: The OMLoadMode for this OMFile.

Back to OMFile

OMFile::OMFile

OMFile::OMFile(const wchar_t* *fileName*, **void*** *clientOnRestoreContext*, **OMFileSignature** *signature*, **const OMAccessMode** *mode*, **OMStoredObject*** *store*, **const OMClassFactory*** *factory***)**

Constructor. Create an OMFile object representing a new named external file.

Defined in: OMFile.cpp

Parameters

fileName
The name of this OMFile.
clientOnRestoreContext
The signature of this OMFile.
signature
The access mode of this OMFile.
mode
The OMStoredObject in which to store the root OMStorable object.
store
The OMClassFactory to use to restore objects from this OMFile.
factory
The root OMStorable object to save in this file.
Back to OMFile

OMFile::OMFile

OMFile::OMFile(void)

Constructor. Create an OMFile object representing a new external file on the given OMRawStorage.

Defined in: OMFile.cpp

Back to OMFile

OMFile::open

void OMFile::open(void)

Open this OMFile.

Defined in: OMFile.cpp

preconditions

!isOpen()
!isClosed()

postconditions

isOpen()

Back to OMFile

OMFile::openExistingModify

OMFile* OMFile::openExistingModify(const wchar_t* *fileName*, **const OMClassFactory*** *factory*, **void*** *clientOnRestoreContext***)**

Open an existing OMFile for modify access, the OMFile is named *fileName*, use the OMClassFactory *factory* to create the objects. The file must already exist.

Defined in: OMFile.cpp

Return Value

The newly opened OMFile.

Parameters

fileName
: The name of the file to open.

factory
: The factory to use for creating objects.

clientOnRestoreContext
: Specifies the use of lazy or eager loading.

Back to OMFile

OMFile::openExistingRead

OMFile* OMFile::openExistingRead(const wchar_t* *fileName*, **const OMClassFactory*** *factory*, **void*** *clientOnRestoreContext***)**

Open an existing OMFile for read-only access, the OMFile is named *fileName*, use the OMClassFactory *factory* to create the objects. The file must already exist.

Defined in: OMFile.cpp

Return Value

The newly opened OMFile.

Parameters

fileName
: The name of the file to open.

factory
: The factory to use for creating objects.

clientOnRestoreContext
Specifies the use of lazy or eager loading.

Back to OMFile

OMFile::openNewModify

OMFile* OMFile::openNewModify(const wchar_t* *fileName*, **const OMClassFactory*** *factory*, **void*** *clientOnRestoreContext*, **const OMByteOrder** *byteOrder***)**

Open a new OMFile for modify access, the OMFile is named *fileName*, use the OMClassFactory *factory* to create the objects. The file must not already exist. The byte ordering on the newly created file is given by *byteOrder*. The client root OMStorable in the newly created file is given by *clientRoot*.

Defined in: OMFile.cpp

Return Value

The newly created OMFile.

Parameters

fileName
The name of the file to create.
factory
The factory to use for creating objects.
clientOnRestoreContext
The byte order to use for the newly created file.
byteOrder
The client root OMStorable in the newly created file.

Back to OMFile

OMFile::rawStorage

OMRawStorage* OMFile::rawStorage(void)

The raw storage on which this OMFile is stored.

Defined in: OMFile.cpp

Return Value

The raw storage on which the raw bytes of this OMFile reside.

Back to OMFile

OMFile::referencedProperties

OMPropertyTable* OMFile::referencedProperties(void)

Retrieve the OMPropertyTable from this OMFile.

Defined in: OMFile.cpp

Return Value

The table of referenced properties.

Back to OMFile

OMFile::restore

OMStorable* OMFile::restore(void)

Restore the client root OMStorable object from this OMFile.

Defined in: OMFile.cpp

Return Value

The newly restored root OMStorable.

preconditions

isOpen()

Back to OMFile

OMFile::revert

void OMFile::revert(void)

Discard all changes made to this OMFile since the last **save** or **open**.

Defined in: OMFile.cpp

Back to OMFile

OMFile::saveAsFile

void OMFile::saveAsFile(OMFile* *destFile*) const

Save the entire contents of this OMFile as well as any unsaved changes in the new file *destFile*. *destFile* must be open, writeable and not yet contain any objects. **saveAsFile** may be called for files opened in modify mode and for files opened in read-only and transient modes.

Defined in: OMFile.cpp

Parameters

destFile
The destination file.

Back to OMFile

OMFile::saveFile

void OMFile::saveFile(void* *clientOnSaveContext*)

Save all changes made to the contents of this OMFile. It is not possible to save read-only or transient files.

Defined in: OMFile.cpp

Parameters

clientOnSaveContext
Client context for callbacks.

preconditions

isOpen()

Back to OMFile

OMFile::signature

OMFileSignature OMFile::signature(void)

The signature of this OMFile.

Defined in: OMFile.cpp

Return Value

The signature of this OMFile.

Back to OMFile

OMFile::validSignature

bool OMFile::validSignature(const OMFileSignature& *signature***)**

Is *signature* a valid signature for an OMFile ?

Defined in: OMFile.cpp

Return Value

True if *signature* is a valid signature for an OMFile, false otherwise.

Parameters

signature
The signature to check.

Back to OMFile

OMFile::~OMFile

OMFile::~OMFile(void)

Destructor.

Defined in: OMFile.cpp

Back to OMFile

OMFixedSizeProperty class

OMFixedSizeProperty **class OMFixedSizeProperty: public OMSimpleProperty**

Fixed size simple (data) persistent properties supported by the Object Manager.

Defined in: OMFixedSizeProperty.h

Class Template Arguments

PropertyType
The type of the property. This can be any type with well defined copy and assignment semantics.

Author

Tim Bingham - tjb - (Avid Technology, Inc.)

Class Members

Public members.

OMFixedSizeProperty(const OMPropertyId propertyId, const wchar_t* name)
Constructor.

virtual ~OMFixedSizeProperty(void)
Destructor.

void getValue**(PropertyType& value) const**
Get the value of this **OMFixedSizeProperty.**

void setValue**(const PropertyType& value)**
Set the value of this **OMFixedSizeProperty.**

OMFixedSizeProperty&ltPropertyType>& operator=**(const PropertyType& value)**
Assignment operator.

operator PropertyType**() const**
Type conversion. Convert an **OMFixedSizeProperty** into a *PropertyType*.

PropertyType* operator&**(void)**
"Address of" operator.

const PropertyType& reference**(void) const**
Convert this **OMFixedSizeProperty** into a const reference to a *PropertyType*.

virtual void restore**(size_t externalSize)**
Restore this **OMFixedSizeProperty**, the external (persisted) size of the **OMFixedSizeProperty** is *externalSize*.

OMFixedSizeProperty::getValue

template <class *PropertyType***>**
void OMFixedSizeProperty<*PropertyType***>::getValue(PropertyType&** *value***) const**

Get the value of this OMFixedSizeProperty.

Defined in: OMFixedSizePropertyT.h

Parameters

value
A value of *PropertyType* by reference.

Class Template Arguments

PropertyType
The type of the property. This can be any type with well defined copy and assignment semantics.

Back to OMFixedSizeProperty

OMFixedSizeProperty::operator PropertyType

template <class *PropertyType***>**
OMFixedSizeProperty<*PropertyType***>::operator PropertyType(void) const**

Type conversion. Convert an OMFixedSizeProperty into a *PropertyType*.

Defined in: OMFixedSizePropertyT.h

Return Value

The result of the conversion as a value of type *PropertyType*.

Class Template Arguments

PropertyType
: The type of the property. This can be any type with well defined copy and assignment semantics.

Back to OMFixedSizeProperty

OMFixedSizeProperty::operator&

template <class *PropertyType***>**
PropertyType* OMFixedSizeProperty<*PropertyType***>::operator&(void)**

"Address of" operator.

Defined in: OMFixedSizePropertyT.h

Return Value

Pointer to a *PropertyType*

Class Template Arguments

PropertyType
: The type of the property. This can be any type with well defined copy and assignment semantics.

Back to OMFixedSizeProperty

OMFixedSizeProperty::operator=

template <class *PropertyType***>**
OMFixedSizeProperty&ltPropertyType>& OMFixedSizeProperty<*PropertyType***>::operator=(const PropertyType&** *value***)**

Assignment operator.

Defined in: OMFixedSizePropertyT.h

Return Value

A value of OMFixedSizeProperty by reference.

Parameters

value
: A value of *PropertyType* by reference.

Class Template Arguments

PropertyType
: The type of the property. This can be any type with well defined copy and assignment semantics.

Back to OMFixedSizeProperty

OMFixedSizeProperty::reference

template <class *PropertyType***>**
const PropertyType& OMFixedSizeProperty<*PropertyType***>::reference(void) const**

Convert this OMFixedSizeProperty into a const reference to a *PropertyType*.

Defined in: OMFixedSizePropertyT.h

Return Value

Const reference to a *PropertyType*.

Class Template Arguments

PropertyType
: The type of the property. This can be any type with well defined copy and assignment semantics.

Back to OMFixedSizeProperty

OMFixedSizeProperty::restore

template <class *PropertyType***>**
void OMFixedSizeProperty<*PropertyType***>::restore(size_t** *externalSize***)**

Restore this OMFixedSizeProperty, the external (persisted) size of the OMFixedSizeProperty is *externalSize*.

Defined in: OMFixedSizePropertyT.h

Parameters

externalSize

The external (persisted) size of the OMFixedSizeProperty.

Class Template Arguments

PropertyType
The type of the property. This can be any type with well defined copy and assignment semantics.

Back to OMFixedSizeProperty

OMFixedSizeProperty::setValue

template <class *PropertyType***>**
void OMFixedSizeProperty<*PropertyType***>::setValue(const PropertyType&** *value***)**

Set the value of this OMFixedSizeProperty.

Defined in: OMFixedSizePropertyT.h

Parameters

value
A value of *PropertyType* by reference.

Class Template Arguments

PropertyType
The type of the property. This can be any type with well defined copy and assignment semantics.

Back to OMFixedSizeProperty

OMIdentitySet class

OMIdentitySet **class OMIdentitySet: public OMContainer**

Sets of unique elements. Duplicate elements are not allowed.

Defined in: OMIdentitySet.h

Class Template Arguments

Element
The type of the unique elements. This type must support operator =, operator !=, operator == and operator

Author

Tim Bingham - tjb - (Avid Technology, Inc.)

Class Members

Public members.

OMIdentitySet()

Constructor.

virtual ~OMIdentitySet(void)

Destructor.

virtual void insert(const Element& element)

Insert *element* into this **OMIdentitySet.**

preconditions

!contains(element)

bool ensurePresent(const Element& element)

If it is not already present, insert *element* into this **OMIdentitySet** and return true, otherwise return false.

virtual bool contains(const Element& element) const

Does this **OMIdentitySet** contain *element* ?

size_t count(void) const

The number of elements in this **OMIdentitySet.**

virtual void remove(const Element& element)

Remove the *element* from this **OMIdentitySet.**

preconditions

contains(element)

virtual void clear(void)

Remove all elements from this **OMIdentitySet.**

bool ensureAbsent(const Element& element)

If it is present, remove *element* from this **OMIdentitySet** and return true, otherwise return false.

Class Members

Private members.

OMKLVStoredObject class

OMKLVStoredObject **class OMKLVStoredObject**

In-memory representation of an object persisted in a SMPTE (Society of Motion Picture and Television Engineers) Key Length Value (KLV) binary file.

Defined in: OMKLVStoredObject.h

Author

Tim Bingham - tjb - (Avid Technology, Inc.)

Class Members

Static members.

static OMKLVStoredObject* openRead**(OMRawStorage* rawStorage)**

Open the root **OMKLVStoredObject** in the raw storage *rawStorage* for reading only.

static OMKLVStoredObject* openModify**(OMRawStorage* rawStorage)**

Open the root **OMKLVStoredObject** in the raw storage *rawStorage* for modification.

static OMKLVStoredObject* createWrite**(OMRawStorage* rawStorage, const OMByteOrder byteOrder)**

Create a new root **OMKLVStoredObject** in the raw storage *rawStorage*. The byte order of the newly created root is given by *byteOrder*.

static OMKLVStoredObject* createModify**(OMRawStorage* rawStorage, const OMByteOrder byteOrder)**

Create a new root **OMKLVStoredObject** in the raw storage *rawStorage*. The byte order of the newly created root is given by *byteOrder*.

static bool isRecognized**(const wchar_t* fileName, OMFileSignature& signature)**

Is the file named *fileName* a recognized file ? If so, the result is true, and the signature is returned in *signature*.

static bool isRecognized**(OMRawStorage* rawStorage, OMFileSignature& signature)**

Does *rawStorage* contain a recognized file ? If so, the result is true, and the signature is returned in *signature*.

static bool isRecognized**(const OMFileSignature& signature)**

Is *signature* recognized ?

Class Members

Public members.

virtual ~OMKLVStoredObject**(void)**

Destructor.

virtual OMStoredObject* create**(const wchar_t* name)**

Create a new **OMKLVStoredObject**, named *name*, contained by this **OMKLVStoredObject.**

Developer Notes

The name argument to this member function doesn't make sense for all derived instances of OMStoredObject.

virtual OMStoredObject* open**(const wchar_t* name)**

Open an exsiting **OMKLVStoredObject**, named *name*, contained by this **OMKLVStoredObject.**

Developer Notes

The name argument to this member function doesn't make sense for all derived instances of OMStoredObject.

virtual void close**(void)**

Close this **OMKLVStoredObject.**

virtual OMByteOrder byteOrder**(void) const**

The byte order of this **OMKLVStoredObject.**

Developer Notes

This member function doesn't make sense for all derived instances of OMStoredObject.

virtual void save**(const OMStoredObjectIdentification& id)**

Save the **OMStoredObjectIdentification** *id* in this **OMKLVStoredObject.**

virtual void save**(const OMPropertySet& properties)**

Save the OMPropertySet *properties* in this **OMKLVStoredObject.**

virtual void save**(const OMSimpleProperty& property)**

Save the OMSimpleProperty *property* in this **OMKLVStoredObject.**

virtual void save**(const OMStrongReference& singleton)**

Save the OMStrongReference *singleton* in this **OMKLVStoredObject.**

virtual void save**(const OMStrongReferenceVector& vector)**

Save the OMStrongReferenceVector *vector* in this **OMKLVStoredObject.**

virtual void save**(const OMStrongReferenceSet& set)**

Save the OMStrongReferenceSet *set* in this **OMKLVStoredObject.**

virtual void save**(const OMWeakReference& singleton)**

Save the OMWeakReference *singleton* in this **OMKLVStoredObject.**

virtual void save**(const OMWeakReferenceVector& vector)**

Save the OMWeakReferenceVector *vector* in this **OMKLVStoredObject.**

virtual void save**(const OMWeakReferenceSet& set)**

Save the OMWeakReferenceSet *set* in this **OMKLVStoredObject.**

virtual void save**(const OMPropertyTable* table)**

Save the OMPropertyTable *table* in this **OMKLVStoredObject.**

Developer Notes

Does this member function make sense for all derived instances of OMStoredObject ?

virtual void save**(const OMDataStream& stream)**

Save the OMDataStream *stream* in this **OMKLVStoredObject.**

virtual void restore**(OMStoredObjectIdentification& id)**

Restore the **OMStoredObjectIdentification** of this **OMKLVStoredObject** into *id*.

virtual void restore**(OMPropertySet& properties)**

Restore the OMPropertySet *properties* into this **OMKLVStoredObject.**

virtual void restore**(OMSimpleProperty& property, size_t externalSize)**

Restore the OMSimpleProperty *property* into this **OMKLVStoredObject.**

Developer Notes

The externalSize argument to this member function doesn't make sense for all derived instances of OMStoredObject.

virtual void restore**(OMStrongReference& singleton, size_t externalSize)**

Restore the OMStrongReference *singleton* into this **OMKLVStoredObject.**

virtual void restore**(OMStrongReferenceVector& vector, size_t externalSize)**

Restore the OMStrongReferenceVector *vector* into this **OMKLVStoredObject.**

virtual void restore**(OMStrongReferenceSet& set, size_t externalSize)**

Restore the OMStrongReferenceSet *set* into this **OMKLVStoredObject.**

virtual void restore**(OMWeakReference& singleton, size_t externalSize)**

Restore the OMWeakReference *singleton* into this **OMKLVStoredObject.**

virtual void restore**(OMWeakReferenceVector& vector, size_t externalSize)**

Restore the OMWeakReferenceVector *vector* into this **OMKLVStoredObject.**

virtual void restore**(OMWeakReferenceSet& set, size_t externalSize)**

Restore the OMWeakReferenceSet *set* into this **OMKLVStoredObject.**

virtual void restore**(OMPropertyTable*& table)**

Restore the OMPropertyTable in this **OMKLVStoredObject.**

Developer Notes

Does this member function make sense for all derived instances of OMStoredObject ?

virtual void restore**(OMDataStream& stream, size_t externalSize)**

Restore the OMDataStream *stream* into this **OMKLVStoredObject.**

virtual OMStoredStream* openStoredStream**(const OMDataStream& property)**

Open the OMStoredStream representing the property *stream* contained within this **OMKLVStoredObject.**

virtual OMStoredStream* createStoredStream**(const OMDataStream& property)**

Create an OMStoredStream representing the property *stream* contained within this **OMKLVStoredObject.**

Class Members

Private members.

OMKLVStoredObject**(OMRawStorage* s, OMByteOrder byteOrder)**

Constructor.

OMKLVStoredObject::byteOrder

OMByteOrder OMKLVStoredObject::byteOrder(void) const

The byte order of this OMKLVStoredObject.

Defined in: OMKLVStoredObject.cpp

Return Value

The byte order.

Back to OMKLVStoredObject

OMKLVStoredObject::close

void OMKLVStoredObject::close(void)

Close this OMKLVStoredObject.

Defined in: OMKLVStoredObject.cpp

Back to OMKLVStoredObject

OMKLVStoredObject::create

OMStoredObject* OMKLVStoredObject::create(const wchar_t* { *TRACE***)**

Create a new OMKLVStoredObject, named *name*, contained by this OMKLVStoredObject.

Defined in: OMKLVStoredObject.cpp

Return Value

A new OMKLVStoredObject contained by this OMKLVStoredObject. name */)

Parameters

TRACE
The name to be used for the new OMKLVStoredObject.

Back to OMKLVStoredObject

OMKLVStoredObject::createModify

OMKLVStoredObject* OMKLVStoredObject::createModify(OMRawStorage* *rawStorage*, **const OMByteOrder** *byteOrder***)**

Create a new root OMKLVStoredObject in the raw storage *rawStorage*. The byte order of the newly created root is given by *byteOrder*.

Defined in: OMKLVStoredObject.cpp

Return Value

An OMKLVStoredObject representing the root object.

Parameters

rawStorage
The raw storage in which to create the file.

byteOrder
The desired byte ordering for the new file.

Back to OMKLVStoredObject

OMKLVStoredObject::createStoredStream

OMStoredStream* OMKLVStoredObject::createStoredStream(const OMDataStream& { *TRACE***)**

Create an OMStoredStream representing the property *stream* contained within this OMKLVStoredObject.

Defined in: OMKLVStoredObject.cpp

Return Value

The newly created OMStoredStream.

Parameters

TRACE
The OMDataStream to be created.
Back to OMKLVStoredObject

OMKLVStoredObject::createWrite

OMKLVStoredObject* OMKLVStoredObject::createWrite(OMRawStorage* *rawStorage*, **const OMByteOrder** *byteOrder***)**

Create a new root OMKLVStoredObject in the raw storage *rawStorage*. The byte order of the newly created root is given by *byteOrder*.

Defined in: OMKLVStoredObject.cpp

Return Value

An OMKLVStoredObject representing the root object.

Parameters

rawStorage
The raw storage in which to create the file.
byteOrder
The desired byte ordering for the new file.
Back to OMKLVStoredObject

OMKLVStoredObject::isRecognized

bool OMKLVStoredObject::isRecognized(const wchar_t* *ANAME*, *fileName***)**

Is the file named *fileName* a recognized file ? If so, the result is true, and the signature is returned in *signature*.

Defined in: OMKLVStoredObject.cpp

Return Value

True if the file is recognized, false otherwise.

Parameters

ANAME
The name of the file to check.
fileName
If recognized, the file signature.
Back to OMKLVStoredObject

OMKLVStoredObject::isRecognized

bool OMKLVStoredObject::isRecognized(OMRawStorage* *ANAME*, *rawStorage***)**

Does *rawStorage* contain a recognized file ? If so, the result is true, and the signature is returned in *signature*.

Defined in: OMKLVStoredObject.cpp

Return Value

True if the OMRawStorage contains a recognized file, false otherwise.

Parameters

ANAME
: The OMRawStorage to check.

rawStorage
: If recognized, the file signature.

Back to OMKLVStoredObject

OMKLVStoredObject::isRecognized

bool OMKLVStoredObject::isRecognized(const OMFileSignature& { *TRACE***)**

Is *signature* recognized ? If so, the result is true.

Defined in: OMKLVStoredObject.cpp

Return Value

True if the signature is recognized, false otherwise. signature */)

Parameters

TRACE
: The signature to check.

Back to OMKLVStoredObject

OMKLVStoredObject::OMKLVStoredObject

OMKLVStoredObject::OMKLVStoredObject(OMRawStorage* *s***, OMByteOrder** *byteOrder***)**

Constructor.

Defined in: OMKLVStoredObject.cpp

Parameters

s
The OMRawStorage on which this OMKLVStoredObject resides.
byteOrder
TBS
Back to OMKLVStoredObject

OMKLVStoredObject::open

OMStoredObject* OMKLVStoredObject::open(const wchar_t* { *TRACE***)**

Open an exsiting OMKLVStoredObject, named *name*, contained by this OMKLVStoredObject.

Defined in: OMKLVStoredObject.cpp

Return Value

The existing OMKLVStoredObject contained by this OMKLVStoredObject. name */)

Parameters

TRACE
The name of the existing OMKLVStoredObject.
Back to OMKLVStoredObject

OMKLVStoredObject::openModify

OMKLVStoredObject* OMKLVStoredObject::openModify(OMRawStorage* *ANAME***)**

Open the root OMKLVStoredObject in the raw storage

Defined in: OMKLVStoredObject.cpp

Return Value

An OMKLVStoredObject representing the root object. *rawStorage* for modification.

Parameters

ANAME
The raw storage in which to open the file.
Back to OMKLVStoredObject

OMKLVStoredObject::openRead

OMKLVStoredObject* OMKLVStoredObject::openRead(OMRawStorage* *ANAME***)**

Open the root OMKLVStoredObject in the raw storage *rawStorage* for reading only.

Defined in: OMKLVStoredObject.cpp

Return Value

An OMKLVStoredObject representing the root object.

Parameters

ANAME
The raw storage in which to open the file.

Back to OMKLVStoredObject

OMKLVStoredObject::openStoredStream

OMStoredStream* OMKLVStoredObject::openStoredStream(const OMDataStream& { *TRACE***)**

Open the OMStoredStream representing the property *stream* contained within this OMKLVStoredObject.

Defined in: OMKLVStoredObject.cpp

Return Value

The newly created OMStoredStream.

Parameters

TRACE
The OMDataStream to be opened.

Back to OMKLVStoredObject

OMKLVStoredObject::restore

void OMKLVStoredObject::restore(OMStrongReferenceSet& size_t { *TRACE***, OMKLVStoredObject::** *restore***)**

Restore the OMStrongReferenceSet *set* into this OMKLVStoredObject.

Defined in: OMKLVStoredObject.cpp

Parameters

TRACE
The newly restored OMStrongReferenceSet.

restore
The external size. set */,

Back to OMKLVStoredObject

OMKLVStoredObject::restore

void OMKLVStoredObject::restore(OMWeakReference& size_t { *TRACE*, OMKLVStoredObject:: *restore*)

Restore the OMWeakReference *singleton* into this OMKLVStoredObject.

Defined in: OMKLVStoredObject.cpp

Parameters

TRACE
The newly restored OMWeakReference.
restore
The external size. singleton */,
Back to OMKLVStoredObject

OMKLVStoredObject::restore

void OMKLVStoredObject::restore(OMSimpleProperty& size_t { *TRACE*, OMKLVStoredObject:: *restore*)

Restore the OMSimpleProperty *property* into this OMKLVStoredObject.

Defined in: OMKLVStoredObject.cpp

Parameters

TRACE
The newly restored OMSimpleProperty
restore
The external size. property */,
Back to OMKLVStoredObject

OMKLVStoredObject::restore

void OMKLVStoredObject::restore(OMStrongReference& size_t { *TRACE*, OMKLVStoredObject:: *restore*)

Restore the OMStrongReference *singleton* into this OMKLVStoredObject.

Defined in: OMKLVStoredObject.cpp

Parameters

TRACE

The newly restored OMStrongReference.

restore

The external size. singleton */,

Back to OMKLVStoredObject

OMKLVStoredObject::restore

void OMKLVStoredObject::restore(OMWeakReferenceVector& size_t { *TRACE*, OMKLVStoredObject:: *restore***)**

Restore the OMWeakReferenceVector *vector* into this OMKLVStoredObject.

Defined in: OMKLVStoredObject.cpp

Parameters

TRACE

The newly restored OMWeakReferenceVector.

restore

The external size. vector */,

Back to OMKLVStoredObject

OMKLVStoredObject::restore

void OMKLVStoredObject::restore(*TRACE***)**

Restore the OMPropertySet *properties* into this OMKLVStoredObject.

Defined in: OMKLVStoredObject.cpp

Parameters

TRACE

The newly restored OMPropertySet. properties */)

Back to OMKLVStoredObject

OMKLVStoredObject::restore

void OMKLVStoredObject::restore(*TRACE***)**

Restore the **OMStoredObjectIdentification** of this OMKLVStoredObject into *id*.

Defined in: OMKLVStoredObject.cpp

Parameters

TRACE
The newly restored **OMStoredObjectIdentification.** id */)
Back to OMKLVStoredObject

OMKLVStoredObject::restore

void OMKLVStoredObject::restore(OMWeakReferenceSet& size_t { *TRACE*, OMKLVStoredObject:: *restore*)

Restore the OMWeakReferenceSet *set* into this OMKLVStoredObject.

Defined in: OMKLVStoredObject.cpp

Parameters

TRACE
The newly restored OMWeakReferenceSet.
restore
The external size. set */,
Back to OMKLVStoredObject

OMKLVStoredObject::restore

void OMKLVStoredObject::restore(*TRACE*)

Restore the OMPropertyTable in this OMKLVStoredObject.

Defined in: OMKLVStoredObject.cpp

Parameters

TRACE
The newly restored OMPropertyTable. table */)
Back to OMKLVStoredObject

OMKLVStoredObject::restore

void OMKLVStoredObject::restore(OMStrongReferenceVector& size_t { *TRACE*, OMKLVStoredObject:: *restore*)

Restore the OMStrongReferenceVector *vector* into this OMKLVStoredObject.

Defined in: OMKLVStoredObject.cpp

Parameters

TRACE
The newly restored OMStrongReferenceVector.
restore
The external size. vector */,
Back to OMKLVStoredObject

OMKLVStoredObject::restore

void OMKLVStoredObject::restore(OMDataStream& size_t { *TRACE*, OMKLVStoredObject:: *restore*)

Restore the OMDataStream *stream* into this OMKLVStoredObject.

Defined in: OMKLVStoredObject.cpp

Parameters

TRACE
The newly restored OMDataStream.
restore
The external size. stream */,
Back to OMKLVStoredObject

OMKLVStoredObject::save

void OMKLVStoredObject::save(*TRACE*)

Save the OMStrongReferenceVector *vector* in this OMKLVStoredObject.

Defined in: OMKLVStoredObject.cpp

Parameters

TRACE
The OMStrongReferenceVector to save. vector */)
Back to OMKLVStoredObject

OMKLVStoredObject::save

void OMKLVStoredObject::save(*TRACE*)

Save the OMStrongReference *singleton* in this OMKLVStoredObject.

Defined in: OMKLVStoredObject.cpp

Parameters

TRACE
The OMStrongReference to save. singleton */)
Back to OMKLVStoredObject

OMKLVStoredObject::save

void OMKLVStoredObject::save(*TRACE***)**

Save the **OMStoredObjectIdentification** *id* in this OMKLVStoredObject.

Defined in: OMKLVStoredObject.cpp

Parameters

TRACE
The **OMStoredObjectIdentification** to save. id */)
Back to OMKLVStoredObject

OMKLVStoredObject::save

void OMKLVStoredObject::save(*TRACE***)**

Save the OMPropertySet *properties* in this OMKLVStoredObject.

Defined in: OMKLVStoredObject.cpp

Parameters

TRACE
The OMPropertySet to save. properties */)
Back to OMKLVStoredObject

OMKLVStoredObject::save

void OMKLVStoredObject::save(*TRACE***)**

Save the OMSimpleProperty *property* in this OMKLVStoredObject.

Defined in: OMKLVStoredObject.cpp

Parameters

TRACE
The OMSimpleProperty to save. property */)
Back to OMKLVStoredObject

OMKLVStoredObject::save

void OMKLVStoredObject::save(*TRACE***)**

Save the OMPropertyTable *table* in this OMKLVStoredObject.

Defined in: OMKLVStoredObject.cpp

Parameters

TRACE
The OMPropertyTable to save. table */)

Back to OMKLVStoredObject

OMKLVStoredObject::save

void OMKLVStoredObject::save(*TRACE***)**

Save the OMDataStream *stream* in this OMKLVStoredObject.

Defined in: OMKLVStoredObject.cpp

Parameters

TRACE
The OMDataStream to save. stream */)

Back to OMKLVStoredObject

OMKLVStoredObject::save

void OMKLVStoredObject::save(*TRACE***)**

Save the OMWeakReferenceSet *set* in this OMKLVStoredObject.

Defined in: OMKLVStoredObject.cpp

Parameters

TRACE
The OMWeakReferenceSet to save. set */)

Back to OMKLVStoredObject

OMKLVStoredObject::save

void OMKLVStoredObject::save(*TRACE***)**

Save the OMStrongReferenceSet *set* in this OMKLVStoredObject.

Defined in: OMKLVStoredObject.cpp

Parameters

TRACE
The OMStrongReference to save. set */)

Back to OMKLVStoredObject

OMKLVStoredObject::save

void OMKLVStoredObject::save(*TRACE***)**

Save the OMWeakReference *singleton* in this OMKLVStoredObject.

Defined in: OMKLVStoredObject.cpp

Parameters

TRACE
The OMWeakReference to save. singleton */)

Back to OMKLVStoredObject

OMKLVStoredObject::save

void OMKLVStoredObject::save(*TRACE***)**

Save the OMWeakReferenceVector *vector* in this OMKLVStoredObject.

Defined in: OMKLVStoredObject.cpp

Parameters

TRACE
The OMWeakReferenceVector to save. vector */)

Back to OMKLVStoredObject

OMKLVStoredObject::~OMKLVStoredObject

OMKLVStoredObject::~OMKLVStoredObject(void)

Destructor.

Defined in: OMKLVStoredObject.cpp

Back to OMKLVStoredObject

OMKLVStoredStream class

OMKLVStoredStream **class OMKLVStoredStream: public OMStoredStream**

Implementation of OMStoredStream for SMPTE (Society of Motion Picture and Television Engineers) Key Length Value (KLV) binary files.

Defined in: OMKLVStoredStream.h

Author

Tim Bingham - tjb - (Avid Technology, Inc.)

Class Members

Public members.

OMKLVStoredStream(OMRawStorage* store)
: Constructor.

~OMKLVStoredStream(void)
: Destructor.

virtual void read(void* data, size_t size) const
: Read *size* bytes from this OMXMLStoredStream into the buffer at address *data*.

virtual void read(OMByte* data, const OMUInt32 bytes, OMUInt32& bytesRead) const
: Attempt to read *bytes* bytes from this OMXMLStoredStream into the buffer at address *data*. The actual number of bytes read is returned in *bytesRead*.

virtual void write(void* data, size_t size)
: Write *size* bytes from the buffer at address *data* to this OMXMLStoredStream.

virtual void write(const OMByte* data, const OMUInt32 bytes, OMUInt32& bytesWritten)
: Attempt to write *bytes* bytes from the buffer at address *data* to this OMXMLStoredStream. The actual number of bytes written is returned in *bytesWritten*.

virtual OMUInt64 size(void) const
: The size of this OMXMLStoredStream in bytes.

virtual void setSize(const OMUInt64 newSize)
: Set the size of this OMXMLStoredStream to *bytes*.

virtual OMUInt64 position(void) const
: The current position for **read()** and **write()**, as an offset in bytes from the begining of this OMXMLStoredStream.

virtual void setPosition(const OMUInt64 offset)
: Set the current position for **read()** and **write()**, as an offset in bytes from the begining of this OMXMLStoredStream.

virtual void close(void)
: Close this OMXMLStoredStream.

Class Members

Private members.

OMMappedFileRawStorage class

OMMappedFileRawStorage **class OMMappedFileRawStorage: public OMRawStorage**

Class supporting access to the raw bytes of memory mapped disk files supported by the Object Manager.

This is an Object Manager built-in implementation of the OMRawStorage interface.

Defined in: OMMappedFileRawStorage.h

Author

Tim Bingham - tjb - (Avid Technology, Inc.)

Class Members

Static members.

static OMMappedFileRawStorage* openExistingRead**(const wchar_t* fileName)**

Create an **OMMappedFileRawStorage** object by opening an existing file for read-only access, the file is named *fileName*. The file must already exist.

static OMMappedFileRawStorage* openExistingModify**(const wchar_t* fileName)**

Create an **OMMappedFileRawStorage** object by opening an existing file for modify access, the file is named *fileName*. The file must already exist.

static OMMappedFileRawStorage* openNewModify**(const wchar_t* fileName)**

Create an **OMMappedFileRawStorage** object by creating a new file for modify access, the file is named *fileName*. The file must not already exist.

Class Members

Public members.

virtual ~OMMappedFileRawStorage**(void)**

Destructor.

virtual bool isReadable**(void) const**

Is it possible to read from this **OMMappedFileRawStorage** ?

virtual void read**(OMByte* bytes, OMUInt32 byteCount, OMUInt32& bytesRead) const**

Attempt to read the number of bytes given by *byteCount* from the current position in this **OMMappedFileRawStorage** into the buffer at address *bytes*. The actual number of bytes read is returned in *bytesRead*. Reading from positions greater than **size** causes *bytesRead* to be less than *byteCount*. Reading bytes that have never been written returns undefined data in *bytes*.

virtual void readAt**(OMUInt64 position, OMByte* bytes, OMUInt32 byteCount, OMUInt32& bytesRead) const**

Attempt to read the number of bytes given by *byteCount* from offset *position* in this **OMMappedFileRawStorage** into the buffer at address *bytes*. The actual number of bytes read is returned in *bytesRead*. Reading from positions greater than **size** causes *bytesRead* to be less than *byteCount*. Reading bytes that have never been written returns undefined data in *bytes*.

preconditions

isReadable() && **isPositionable()**

virtual bool isWritable**(void) const**

Is it possible to write to this **OMMappedFileRawStorage** ?

virtual void write**(const OMByte* bytes, OMUInt32 byteCount, OMUInt32& bytesWritten)**

Attempt to write the number of bytes given by *byteCount* to the current position in this **OMMappedFileRawStorage** from the buffer at address *bytes*. The actual number of bytes written is returned in *bytesWritten*. Writing to positions greater than **size** causes this **OMMappedFileRawStorage** to be extended, however such extension can fail, causing *bytesWritten* to be less than *byteCount*.

virtual void writeAt**(OMUInt64 position, const OMByte* bytes, OMUInt32 byteCount, OMUInt32& bytesWritten)**

Attempt to write the number of bytes given by *byteCount* to offset *position* in this **OMMappedFileRawStorage** from the buffer at address *bytes*. The actual number of bytes written is returned in *bytesWritten*. Writing to positions greater than **size** causes this **OMMappedFileRawStorage** to be extended, however such extension can fail, causing *bytesWritten* to be less than *byteCount*.

preconditions

isWritable() && **isPositionable()**

Developer Notes

How is failure to extend indicated ?

virtual bool isExtendible**(void) const**

May this **OMMappedFileRawStorage** be changed in size ?

virtual OMUInt64 extent**(void) const**

The current extent of this **OMMappedFileRawStorage** in bytes. The **extent()** is the allocated size, while the **size()** is the valid size. precondition - isPositionable()

virtual void extend**(OMUInt64 newSize)**

Set the size of this **OMMappedFileRawStorage** to *newSize* bytes. If *newSize* is greater than **size** then this **OMMappedFileRawStorage** is extended. If *newSize* is less than **size** then this **OMMappedFileRawStorage** is truncated. Truncation may also result in the current position for **read()** and **write()** being set to **size**. precondition - isExtendible()

virtual OMUInt64 size**(void) const**

The current size of this **OMMappedFileRawStorage** in bytes. The **size()** is the valid size, while the **extent()** is the allocated size. precondition - isPositionable()

virtual bool isPositionable**(void) const**

May the current position, for **read()** and **write()**, of this **OMMappedFileRawStorage** be changed ?

virtual OMUInt64 position**(void) const**

The current position for **read()** and **write()**, as an offset in bytes from the beginning of this **OMMappedFileRawStorage**. precondition - isPositionable()

virtual void setPosition**(OMUInt64 newPosition) const**

Set the current position for **read()** and **write()**, as an offset in bytes from the beginning of this **OMMappedFileRawStorage**. precondition - isPositionable()

virtual void synchronize**(void)**

Synchronize this **OMMappedFileRawStorage** with its external representation.

Class Members

Private members.
none OMMappedFileRawStorage**accessMode**

Constructor. TBS */ OMFile::OMAccessMode accessMode);

OMMappedFileRawStorage::extend

void OMMappedFileRawStorage::extend(OMUInt64 { *TRACE***)**

Set the size of this OMMappedFileRawStorage to *newSize* bytes. If *newSize* is greater than **size** then this OMMappedFileRawStorage is extended. If *newSize* is less than **size** then this OMMappedFileRawStorage is truncated. Truncation may also result in the current position for **read()** and **write()** being set to **size**. precondition - isExtendible()

Defined in: OMMappedFileRawStorage.cpp

Parameters

TRACE

The new size of this OMMappedFileRawStorage in bytes.

Developer Notes

There is no ISO/ANSI way of truncating a file in place. newSize */)

Back to OMMappedFileRawStorage

OMMappedFileRawStorage::extent

OMUInt64 OMMappedFileRawStorage::extent(void) const

The current extent of this OMMappedFileRawStorage in bytes. precondition - isPositionable()

Defined in: OMMappedFileRawStorage.cpp

Return Value

The current extent of this OMMappedFileRawStorage in bytes.

Back to OMMappedFileRawStorage

OMMappedFileRawStorage::isExtendible

bool OMMappedFileRawStorage::isExtendible(void) const

May this OMMappedFileRawStorage be changed in size ?

Defined in: OMMappedFileRawStorage.cpp

Return Value

Always **true** .

Back to OMMappedFileRawStorage

OMMappedFileRawStorage::isPositionable

bool OMMappedFileRawStorage::isPositionable(void) const

May the current position, for **read()** and **write()**, of this OMMappedFileRawStorage be changed ?

Defined in: OMMappedFileRawStorage.cpp

Return Value

Always **true** .

Back to OMMappedFileRawStorage

OMMappedFileRawStorage::isReadable

bool OMMappedFileRawStorage::isReadable(void) const

Is it possible to read from this OMMappedFileRawStorage ?

Defined in: OMMappedFileRawStorage.cpp

Return Value

True if this OMMappedFileRawStorage is readable, false otherwise.

Back to OMMappedFileRawStorage

OMMappedFileRawStorage::isWritable

bool OMMappedFileRawStorage::isWritable(void) const

Is it possible to write to this OMMappedFileRawStorage ?

Defined in: OMMappedFileRawStorage.cpp

Return Value

True if this OMMappedFileRawStorage is writable, false otherwise.

Back to OMMappedFileRawStorage

OMMappedFileRawStorage::OMMappedFileRawStorage

OMMappedFileRawStorage::OMMappedFileRawStorage(OMFile::OMAccessMode { *TRACE*)

Constructor.

Defined in: OMMappedFileRawStorage.cpp

Parameters

TRACE
The access mode.

Back to OMMappedFileRawStorage

OMMappedFileRawStorage::openExistingModify

OMMappedFileRawStorage* OMMappedFileRawStorage::openExistingModify(const wchar_t* { *TRACE*)

Create an OMMappedFileRawStorage object by opening an existing file for modify access, the file is named *fileName*. The file must already exist.

Defined in: OMMappedFileRawStorage.cpp

Return Value

The newly created OMMappedFileRawStorage object.

Parameters

TRACE
The file name.

Back to OMMappedFileRawStorage

OMMappedFileRawStorage::openExistingRead

OMMappedFileRawStorage* OMMappedFileRawStorage::openExistingRead(const wchar_t* { *TRACE*)

Create an OMMappedFileRawStorage object by opening an existing file for read-only access, the file is named *fileName*. The file must already exist.

Defined in: OMMappedFileRawStorage.cpp

Return Value

The newly created OMMappedFileRawStorage object.

Parameters

TRACE
The file name.

Back to OMMappedFileRawStorage

OMMappedFileRawStorage::openNewModify

OMMappedFileRawStorage* OMMappedFileRawStorage::openNewModify(const wchar_t* { *TRACE*)

Create an OMMappedFileRawStorage object by creating a new file for modify access, the file is named *fileName*. The file must not already exist.

Defined in: OMMappedFileRawStorage.cpp

Return Value

The newly created OMMappedFileRawStorage object.

Parameters

TRACE
The file name.

Back to OMMappedFileRawStorage

OMMappedFileRawStorage::position

OMUInt64 OMMappedFileRawStorage::position(void) const

The current position for **read()** and **write()**, as an offset in bytes from the beginning of this OMMappedFileRawStorage. precondition - isPositionable()

Defined in: OMMappedFileRawStorage.cpp

Return Value

The current position for **read()** and **write()**.

Back to OMMappedFileRawStorage

OMMappedFileRawStorage::read

void OMMappedFileRawStorage::read(OMByte* *bytes*, **OMUInt32** *byteCount*, **OMUInt32&** *bytesRead*) **const bytes */,**

Attempt to read the number of bytes given by *byteCount* from the current position in this OMMappedFileRawStorage into the buffer at address *bytes*. The actual number of bytes read is returned in *bytesRead*. Reading from positions greater than **size** causes *bytesRead* to be less than *byteCount*. Reading bytes that have never been written returns undefined data in *bytes*.

Defined in: OMMappedFileRawStorage.cpp

Parameters

bytes
: The buffer into which the bytes are to be read.

byteCount
: The number of bytes to read.

bytesRead
: The number of bytes actually read.

Back to OMMappedFileRawStorage

OMMappedFileRawStorage::readAt

void OMMappedFileRawStorage::readAt(OMUInt64 *position*, **OMByte*** *bytes*, **OMUInt32** *byteCount*, **OMUInt32&** *bytesRead*) **const position */,**

Attempt to read the number of bytes given by *byteCount* from offset *position* in this OMMappedFileRawStorage into the buffer at address *bytes*. The actual number of bytes read is returned in *bytesRead*. Reading from positions greater than **size** causes *bytesRead* to be less than *byteCount*. Reading bytes that have never been written returns undefined data in *bytes*.

Defined in: OMMappedFileRawStorage.cpp

Parameters

position
: The position from which the bytes are to be read.

bytes
: The buffer into which the bytes are to be read.

byteCount
: The number of bytes to read.

bytesRead
: The number of bytes actually read.

Back to OMMappedFileRawStorage

OMMappedFileRawStorage::setPosition

void OMMappedFileRawStorage::setPosition(OMUInt64 { *TRACE*) const

Set the current position for **read()** and **write()**, as an offset in bytes from the beginning of this OMMappedFileRawStorage. precondition - isPositionable()

Defined in: OMMappedFileRawStorage.cpp

Parameters

TRACE
The new position.

Developer Notes

fseek takes a long int for offset this may not be sufficient for 64-bit offsets. newPosition */) const

Back to OMMappedFileRawStorage

OMMappedFileRawStorage::size

OMUInt64 OMMappedFileRawStorage::size(void) const

The current size of this OMMappedFileRawStorage in bytes. precondition - isPositionable()

Defined in: OMMappedFileRawStorage.cpp

Return Value

The current size of this OMMappedFileRawStorage in bytes.

Back to OMMappedFileRawStorage

OMMappedFileRawStorage::synchronize

void OMMappedFileRawStorage::synchronize(void)

Synchronize this OMMappedFileRawStorage with its external representation.

Defined in: OMMappedFileRawStorage.cpp

Back to OMMappedFileRawStorage

OMMappedFileRawStorage::write

void OMMappedFileRawStorage::write(const OMByte* *bytes*, **OMUInt32** *byteCount*, **OMUInt32&** *bytesWritten***)**

Attempt to write the number of bytes given by *byteCount* to the current position in this OMMappedFileRawStorage from the buffer at address *bytes*. The actual number of bytes written is returned in *bytesWritten*. Writing to positions greater than **size** causes this OMMappedFileRawStorage to be extended, however such extension can fail, causing *bytesWritten* to be less than *byteCount*.

Defined in: OMMappedFileRawStorage.cpp

Parameters

bytes
The buffer from which the bytes are to be written.
byteCount
The number of bytes to write.
bytesWritten
The actual number of bytes written. bytes */,
Back to OMMappedFileRawStorage

OMMappedFileRawStorage::writeAt

void OMMappedFileRawStorage::writeAt(OMUInt64 *position*, **OMByte*** *bytes*, **OMUInt32** *byteCount*, **OMUInt32&** *bytesWritten***)**

Attempt to write the number of bytes given by *byteCount* to offset *position* in this OMMappedFileRawStorage from the buffer at address *bytes*. The actual number of bytes written is returned in *bytesWritten*. Writing to positions greater than **size** causes this OMMappedFileRawStorage to be extended, however such extension can fail, causing *bytesWritten* to be less than *byteCount*.

Defined in: OMMappedFileRawStorage.cpp

Parameters

position
The position to which the bytes are to be written.
bytes
The buffer from which the bytes are to be written.
byteCount
The number of bytes to write.
bytesWritten
The actual number of bytes written. position */,
Back to OMMappedFileRawStorage

OMMappedFileRawStorage::~OMMappedFileRawStorage

OMMappedFileRawStorage::~OMMappedFileRawStorage(void)

Destructor.

Defined in: OMMappedFileRawStorage.cpp

Back to OMMappedFileRawStorage

OMMemoryRawStorage class

OMMemoryRawStorage **class OMMemoryRawStorage: public OMRawStorage**

Class supporting access to the raw bytes of memory files supported by the Object Manager.

This is an Object Manager built-in implementation of the OMRawStorage interface.

Defined in: OMMemoryRawStorage.h

Author

Tim Bingham - tjb - (Avid Technology, Inc.)

Class Members

Static members.
static OMMemoryRawStorage* openNewModify**(void)**
Create an **OMMemoryRawStorage** object for modify access.

Class Members

Public members.
virtual ~OMMemoryRawStorage**(void)**
Destructor.
virtual bool isReadable**(void) const**
Is it possible to read from this **OMMemoryRawStorage** ?
virtual void read**(OMByte* bytes, OMUInt32 byteCount, OMUInt32& bytesRead) const**
Attempt to read the number of bytes given by *byteCount* from the current position in this **OMMemoryRawStorage** into the buffer at address *bytes*. The actual number of bytes read is returned in *bytesRead*. Reading from positions greater than **size** causes *bytesRead* to be less than *byteCount*. Reading bytes that have never been written returns undefined data in *bytes*.
virtual void readAt**(OMUInt64 position, OMByte* bytes, OMUInt32 byteCount, OMUInt32& bytesRead) const**
Attempt to read the number of bytes given by *byteCount* from offset *position* in this **OMMemoryRawStorage** into the buffer at address *bytes*. The actual number of bytes read is returned in *bytesRead*. Reading from positions greater than **size** causes *bytesRead* to be less than *byteCount*. Reading bytes that have never been written returns undefined data in *bytes*.

preconditions

isReadable() && **isPositionable()**

virtual bool isWritable**(void) const**

Is it possible to write to this **OMMemoryRawStorage** ?

virtual void write**(const OMByte* bytes, OMUInt32 byteCount, OMUInt32& bytesWritten)**

Attempt to write the number of bytes given by *byteCount* to the current position in this **OMMemoryRawStorage** from the buffer at address *bytes*. The actual number of bytes written is returned in *bytesWritten*. Writing to positions greater than **size** causes this **OMMemoryRawStorage** to be extended, however such extension can fail, causing *bytesWritten* to be less than *byteCount*.

virtual void writeAt**(OMUInt64 position, const OMByte* bytes, OMUInt32 byteCount, OMUInt32& bytesWritten)**

Attempt to write the number of bytes given by *byteCount* to offset *position* in this **OMMemoryRawStorage** from the buffer at address *bytes*. The actual number of bytes written is returned in *bytesWritten*. Writing to positions greater than **size** causes this **OMMemoryRawStorage** to be extended, however such extension can fail, causing *bytesWritten* to be less than *byteCount*.

preconditions

isWritable() && **isPositionable()**

Developer Notes

How is failure to extend indicated ?

virtual bool isExtendible**(void) const**

May this **OMMemoryRawStorage** be changed in size ?

virtual OMUInt64 extent**(void) const**

The current extent of this **OMMemoryRawStorage** in bytes. The **extent()** is the allocated size, while the **size()** is the valid size. precondition - isPositionable()

virtual void extend**(OMUInt64 newSize)**

Set the size of this **OMMemoryRawStorage** to *newSize* bytes. If *newSize* is greater than **size** then this **OMMemoryRawStorage** is extended. If *newSize* is less than **size** then this **OMMemoryRawStorage** is truncated. Truncation may also result in the current position for **read()** and **write()** being set to **size**. precondition - isExtendible()

virtual OMUInt64 size**(void) const**

The current size of this **OMMemoryRawStorage** in bytes. The **size()** is the valid size, while the **extent()** is the allocated size. precondition - isPositionable()

virtual bool isPositionable**(void) const**

May the current position, for **read()** and **write()**, of this **OMMemoryRawStorage** be changed ?

virtual OMUInt64 position**(void) const**

The current position for **read()** and **write()**, as an offset in bytes from the beginning of this **OMMemoryRawStorage**. precondition - isPositionable()

virtual void setPosition**(OMUInt64 newPosition) const**

Set the current position for **read()** and **write()**, as an offset in bytes from the beginning of this **OMMemoryRawStorage**. precondition - isPositionable()

virtual void synchronize**(void)**

Synchronize this **OMMemoryRawStorage** with its external representation.

Class Members

Private members.

OMMemoryRawStorage**(void)**

Constructor.

virtual void write**(size_t page, size_t offset, size_t byteCount, const OMByte* source)**

Write a page or partial page.

virtual void read**(size_t page, size_t offset, size_t byteCount, OMByte* destination) const**

Read a page or partial page.

OMMemoryRawStorage::extend

void OMMemoryRawStorage::extend(OMUInt64 *newSize***)**

Set the size of this OMMemoryRawStorage to *newSize* bytes. If *newSize* is greater than **size** then this OMMemoryRawStorage is extended. If *newSize* is less than **size** then this OMMemoryRawStorage is truncated. Truncation may also result in the current position for **read()** and **write()** being set to **size**. precondition - isExtendible()

Defined in: OMMemoryRawStorage.cpp

Parameters

newSize

The new size of this OMMemoryRawStorage in bytes.

Developer Notes

There is no ANSI way of truncating a file in place.

Back to OMMemoryRawStorage

OMMemoryRawStorage::extent

OMUInt64 OMMemoryRawStorage::extent(void) const

The current extent of this OMMemoryRawStorage in bytes. precondition - isPositionable()

Defined in: OMMemoryRawStorage.cpp

Return Value

The current extent of this OMMemoryRawStorage in bytes.

Back to OMMemoryRawStorage

OMMemoryRawStorage::isExtendible

bool OMMemoryRawStorage::isExtendible(void) const

May this OMMemoryRawStorage be changed in size ?

Defined in: OMMemoryRawStorage.cpp

Return Value

Always **true** .

Back to OMMemoryRawStorage

OMMemoryRawStorage::isPositionable

bool OMMemoryRawStorage::isPositionable(void) const

May the current position, for **read()** and **write()**, of this OMMemoryRawStorage be changed ?

Defined in: OMMemoryRawStorage.cpp

Return Value

Always **true** .

Back to OMMemoryRawStorage

OMMemoryRawStorage::isReadable

bool OMMemoryRawStorage::isReadable(void) const

Is it possible to read from this OMMemoryRawStorage ?

Defined in: OMMemoryRawStorage.cpp

Return Value

True if this OMMemoryRawStorage is readable, false otherwise.

Back to OMMemoryRawStorage

OMMemoryRawStorage::isWritable

bool OMMemoryRawStorage::isWritable(void) const

Is it possible to write to this OMMemoryRawStorage ?

Defined in: OMMemoryRawStorage.cpp

Return Value

True if this OMMemoryRawStorage is writable, false otherwise.

Back to OMMemoryRawStorage

OMMemoryRawStorage::OMMemoryRawStorage

OMMemoryRawStorage::OMMemoryRawStorage(void)

Constructor.

Defined in: OMMemoryRawStorage.cpp

Back to OMMemoryRawStorage

OMMemoryRawStorage::openNewModify

OMMemoryRawStorage* OMMemoryRawStorage::openNewModify(void)

Create an OMMemoryRawStorage object for modify access.

Defined in: OMMemoryRawStorage.cpp

Return Value

The newly created OMMemoryRawStorage object.

Back to OMMemoryRawStorage

OMMemoryRawStorage::position

OMUInt64 OMMemoryRawStorage::position(void) const

The current position for **read()** and **write()**, as an offset in bytes from the beginning of this OMMemoryRawStorage.
precondition - isPositionable()

Defined in: OMMemoryRawStorage.cpp

Return Value

The current position for **read()** and **write()**.

Back to OMMemoryRawStorage

OMMemoryRawStorage::read

void OMMemoryRawStorage::read(size_t *page*, **size_t** *offset*, **size_t** *byteCount*, **OMByte*** *destination***)**

Read a page or partial page.

Defined in: OMMemoryRawStorage.cpp

Parameters

page
: The page to be read from.

offset
: The starting offset within the page.

byteCount
: the number of bytes to read.

destination
: The buffer into which the bytes are to be read.

Back to OMMemoryRawStorage

OMMemoryRawStorage::read

void OMMemoryRawStorage::read(OMByte* *bytes*, **OMUInt32** *byteCount*, **OMUInt32&** *bytesRead***) const**

Attempt to read the number of bytes given by *byteCount* from the current position in this OMMemoryRawStorage into the buffer at address *bytes*. The actual number of bytes read is returned in *bytesRead*. Reading from positions greater than **size** causes *bytesRead* to be less than *byteCount*. Reading bytes that have never been written returns undefined data in *bytes*.

Defined in: OMMemoryRawStorage.cpp

Parameters

bytes
: The buffer into which the bytes are to be read.

byteCount
: The number of bytes to read.

bytesRead
: The number of bytes actually read.

Developer Notes

fseek takes a long int for offset this may not be sufficient for 64-bit offsets.

Back to OMMemoryRawStorage

OMMemoryRawStorage::readAt

void OMMemoryRawStorage::readAt(OMUInt64 *position*, **OMByte*** *bytes*, **OMUInt32** *byteCount*, **OMUInt32&** *bytesRead*) **const**

Attempt to read the number of bytes given by *byteCount* from offset *position* in this OMMemoryRawStorage into the buffer at address *bytes*. The actual number of bytes read is returned in *bytesRead*. Reading from positions greater than **size** causes *bytesRead* to be less than *byteCount*. Reading bytes that have never been written returns undefined data in *bytes*.

Defined in: OMMemoryRawStorage.cpp

Parameters

position
: The position from which the bytes are to be read.

bytes
: The buffer into which the bytes are to be read.

byteCount
: The number of bytes to read.

bytesRead
: The number of bytes actually read.

Back to OMMemoryRawStorage

OMMemoryRawStorage::setPosition

void OMMemoryRawStorage::setPosition(OMUInt64 *newPosition*) **const**

Set the current position for **read()** and **write()**, as an offset in bytes from the beginning of this OMMemoryRawStorage. precondition - isPositionable()

Defined in: OMMemoryRawStorage.cpp

Parameters

newPosition
: The new position.

Developer Notes

fseek takes a long int for offset this may not be sufficient for 64-bit offsets.

Back to OMMemoryRawStorage

OMMemoryRawStorage::size

OMUInt64 OMMemoryRawStorage::size(void) const

The current size of this OMMemoryRawStorage in bytes. precondition - isPositionable()

Defined in: OMMemoryRawStorage.cpp

Return Value

The current size of this OMMemoryRawStorage in bytes.

Back to OMMemoryRawStorage

OMMemoryRawStorage::synchronize

void OMMemoryRawStorage::synchronize(void)

Synchronize this OMMemoryRawStorage with its external representation.

Defined in: OMMemoryRawStorage.cpp

Back to OMMemoryRawStorage

OMMemoryRawStorage::write

void OMMemoryRawStorage::write(const OMByte* *bytes*, **OMUInt32** *byteCount*, **OMUInt32&** *bytesWritten***)**

Attempt to write the number of bytes given by *byteCount* to the current position in this OMMemoryRawStorage from the buffer at address *bytes*. The actual number of bytes written is returned in *bytesWritten*. Writing to positions greater than **size** causes this OMMemoryRawStorage to be extended, however such extension can fail, causing *bytesWritten* to be less than *byteCount*.

Defined in: OMMemoryRawStorage.cpp

Parameters

bytes
: The buffer from which the bytes are to be written.

byteCount
: The number of bytes to write.

bytesWritten
: The actual number of bytes written.

Developer Notes

fseek takes a long int for offset this may not be sufficient for 64-bit offsets.

Back to OMMemoryRawStorage

OMMemoryRawStorage::write

void OMMemoryRawStorage::write(size_t *page*, **size_t** *offset*, **size_t** *byteCount*, **const OMByte*** *source***)**

Write a page or partial page.

Defined in: OMMemoryRawStorage.cpp

Parameters

page
: The page to be written to.

offset
: The starting offset within the page.

byteCount
: the number of bytes to write.

source
: The buffer from which the bytes are to be written.

Back to OMMemoryRawStorage

OMMemoryRawStorage::writeAt

void OMMemoryRawStorage::writeAt(OMUInt64 *position*, **const OMByte*** *bytes*, **OMUInt32** *byteCount*, **OMUInt32&** *bytesWritten***)**

Attempt to write the number of bytes given by *byteCount* to offset *position* in this OMMemoryRawStorage from the buffer at address *bytes*. The actual number of bytes written is returned in *bytesWritten*. Writing to positions greater than **size** causes this OMMemoryRawStorage to be extended, however such extension can fail, causing *bytesWritten* to be less than *byteCount*.

Defined in: OMMemoryRawStorage.cpp

Parameters

position
: The position to which the bytes are to be written.

bytes
: The buffer from which the bytes are to be written.

byteCount
: The number of bytes to write.

bytesWritten
: The actual number of bytes written.

Back to OMMemoryRawStorage

OMMemoryRawStorage::~OMMemoryRawStorage

OMMemoryRawStorage::~OMMemoryRawStorage(void)

Destructor.

Defined in: OMMemoryRawStorage.cpp

Back to OMMemoryRawStorage

OMMSSStoredObject class

OMMSSStoredObject **class OMMSSStoredObject**

In-memory representation of an object persisted in a Microsoft Structured Storage (MSS) binary file.

Defined in: OMMSSStoredObject.h

Author

Tim Bingham - tjb - (Avid Technology, Inc.)

Class Members

Static members.
static OMMSSStoredObject* openRead**(const wchar_t* fileName)**

Open the root **OMMSSStoredObject** in the disk file *fileName* for reading only.

Developer Notes

Soon to be obsolete.

static OMMSSStoredObject* openModify**(const wchar_t* fileName)**

Open the root **OMMSSStoredObject** in the disk file *fileName* for modification.

Developer Notes

Soon to be obsolete.

static OMMSSStoredObject* createModify**(const wchar_t* fileName, const OMByteOrder byteOrder)**

Create a new root **OMMSSStoredObject** in the disk file *fileName*. The byte order of the newly created root is given by *byteOrder*.

Developer Notes

Soon to be obsolete.

static OMMSSStoredObject* openRead**(OMRawStorage* rawStorage)**

Open the root **OMMSSStoredObject** in the raw storage *rawStorage* for reading only.

static OMMSSStoredObject* openModify**(OMRawStorage* rawStorage)**

Open the root **OMMSSStoredObject** in the raw storage *rawStorage* for modification.

static OMMSSStoredObject* createWrite**(OMRawStorage* rawStorage, const OMByteOrder byteOrder)**

Create a new root **OMMSSStoredObject** in the raw storage *rawStorage*. The byte order of the newly created root is given by *byteOrder*.

static OMMSSStoredObject* createModify**(OMRawStorage* rawStorage, const OMByteOrder byteOrder)**

Create a new root **OMMSSStoredObject** in the raw storage *rawStorage*. The byte order of the newly created root is given by *byteOrder*.

static bool isRecognized**(const wchar_t* fileName, OMFileSignature& signature)**

Is the file named *fileName* a recognized file ? If so, the result is true, and the signature is returned in *signature*.

static bool isRecognized**(OMRawStorage* rawStorage, OMFileSignature& signature)**

Does *rawStorage* contain a recognized file ? If so, the result is true, and the signature is returned in *signature*.

static bool isRecognized**(const OMFileSignature& signature)**

Is *signature* recognized ?

Class Members

Public members.

virtual ~OMMSSStoredObject**(void)**

Destructor.

virtual OMStoredObject* create**(const wchar_t* name)**

Create a new **OMMSSStoredObject**, named *name*, contained by this **OMMSSStoredObject.**

Developer Notes

The name argument to this member function doesn't make sense for all derived instances of OMStoredObject.

virtual OMStoredObject* open**(const wchar_t* name)**

Open an exsiting **OMMSSStoredObject**, named *name*, contained by this **OMMSSStoredObject.**

Developer Notes

The name argument to this member function doesn't make sense for all derived instances of OMStoredObject.

virtual void close**(void)**

Close this **OMMSSStoredObject.**

virtual OMByteOrder byteOrder(void) const

The byte order of this **OMMSSStoredObject.**

Developer Notes

This member function doesn't make sense for all derived instances of OMStoredObject.

virtual void save**(const OMStoredObjectIdentification& id)**

Save the **OMStoredObjectIdentification** *id* in this **OMMSSStoredObject.**

virtual void save**(const OMPropertySet& properties)**

Save the OMPropertySet *properties* in this **OMMSSStoredObject.**

virtual void save**(const OMSimpleProperty& property)**

Save the OMSimpleProperty *property* in this **OMMSSStoredObject.**

virtual void save**(const OMStrongReference& singleton)**

Save the OMStrongReference *singleton* in this **OMMSSStoredObject.**

virtual void save**(const OMStrongReferenceVector& vector)**

Save the OMStrongReferenceVector *vector* in this **OMMSSStoredObject.**

virtual void save**(const OMStrongReferenceSet& set)**

Save the OMStrongReferenceSet *set* in this **OMMSSStoredObject.**

virtual void save**(const OMWeakReference& singleton)**

Save the OMWeakReference *singleton* in this **OMMSSStoredObject.**

virtual void save**(const OMWeakReferenceVector& vector)**

Save the OMWeakReferenceVector *vector* in this **OMMSSStoredObject.**

virtual void save**(const OMWeakReferenceSet& set)**

Save the OMWeakReferenceSet *set* in this **OMMSSStoredObject.**

virtual void save**(const OMPropertyTable* table)**

Save the OMPropertyTable *table* in this **OMMSSStoredObject.**

Developer Notes

Does this member function make sense for all derived instances of OMStoredObject ?

virtual void save**(const OMDataStream& stream)**

Save the OMDataStream *stream* in this **OMMSSStoredObject.**

virtual void restore**(OMStoredObjectIdentification& id)**

Restore the **OMStoredObjectIdentification** of this **OMMSSStoredObject** into *id*.

virtual void restore**(OMPropertySet& properties)**

Restore the OMPropertySet *properties* into this **OMMSSStoredObject.**

virtual void restore**(OMSimpleProperty& property, size_t externalSize)**

Restore the OMSimpleProperty *property* into this **OMMSSStoredObject.**

Developer Notes

The externalSize argument to this member function doesn't make sense for all derived instances of OMStoredObject.

virtual void restore**(OMStrongReference& singleton, size_t externalSize)**

Restore the OMStrongReference *singleton* into this **OMMSSStoredObject.**

virtual void restore**(OMStrongReferenceVector& vector, size_t externalSize)**

Restore the OMStrongReferenceVector *vector* into this **OMMSSStoredObject.**

virtual void restore**(OMStrongReferenceSet& set, size_t externalSize)**

Restore the OMStrongReferenceSet *set* into this **OMMSSStoredObject.**

virtual void restore**(OMWeakReference& singleton, size_t externalSize)**

Restore the OMWeakReference *singleton* into this **OMMSSStoredObject.**

virtual void restore**(OMWeakReferenceVector& vector, size_t externalSize)**

Restore the OMWeakReferenceVector *vector* into this **OMMSSStoredObject.**

virtual void restore**(OMWeakReferenceSet& set, size_t externalSize)**

Restore the OMWeakReferenceSet *set* into this **OMMSSStoredObject.**

virtual void restore**(OMPropertyTable*& table)**

Restore the OMPropertyTable in this **OMMSSStoredObject.**

Developer Notes

Does this member function make sense for all derived instances of OMStoredObject ?

virtual void restore**(OMDataStream& stream, size_t externalSize)**

Restore the OMDataStream *stream* into this **OMMSSStoredObject.**

virtual OMStoredStream* openStoredStream**(const OMDataStream& property)**

Open the OMStoredStream representing the property *stream* contained within this **OMMSSStoredObject.**

virtual OMStoredStream* createStoredStream**(const OMDataStream& property)**

Create an OMStoredStream representing the property *stream* contained within this **OMMSSStoredObject.**

Class Members

Protected members.

OMMSSStoredObject**(IStorage* s)**

Constructor.

void validate**(const OMPropertySet* propertySet, const OMStoredPropertySetIndex* propertySetIndex) const**

Check that the OMPropertySet *propertySet* is consistent with the OMStoredPropertySetIndex propertySetIndex.

void save**(const OMStoredVectorIndex* vector, const wchar_t* vectorName)**

Save the OMStoredVectorIndex *vector* in this **OMMSSStoredObject**, the vector is named *vectorName*.

void save**(const OMStoredSetIndex* set, const wchar_t* setName)**

Save the OMStoredSetIndex *set* in this **OMMSSStoredObject**, the set is named *setName*.

void save**(OMPropertyId propertyId, OMStoredForm storedForm, const OMUniqueObjectIdentification& id, OMPropertyTag tag, OMPropertyId keyPropertyId)**

Save a single weak reference.

void save**(const wchar_t* collectionName, const OMUniqueObjectIdentification* index, size_t count, OMPropertyTag tag, OMPropertyId keyPropertyId)**

Save a collection (vector/set) of weak references.

void restore**(OMStoredVectorIndex*& vector, const wchar_t* vectorName)**

Restore the vector named *vectorName* into this **OMMSSStoredObject.**

void restore**(OMStoredSetIndex*& set, const wchar_t* setName)**

Restore the set named *setName* into this **OMMSSStoredObject.**

void restore**(OMPropertyId propertyId, OMStoredForm storedForm, OMUniqueObjectIdentification& id, OMPropertyTag& tag, OMPropertyId& keyPropertyId)**

Restore a single weak reference.

void restore**(const wchar_t* collectionName, OMUniqueObjectIdentification*& index, size_t &count, OMPropertyTag& tag, OMPropertyId& keyPropertyId)**

Restore a collection (vector/set) of weak references.

void write**(OMPropertyId propertyId, OMStoredForm storedForm, void* start, size_t size)**

Write a property value to this **OMMSSStoredObject**. The property value to be written occupies *size* bytes at the address *start*. The property id is *propertyId*. The property type is *type*.

void read**(OMPropertyId propertyId, OMStoredForm storedForm, void* start, size_t size)**

Read a property value from this **OMMSSStoredObject**. The property value is read into a buffer which occupies *size* bytes at the address *start*. The property id is *propertyId*. The property type is *type*.

IStream* openStream**(const wchar_t* streamName)**

Open a stream called *streamName* contained within this **OMMSSStoredObject.**

IStream* createStream**(const wchar_t* streamName)**

Create a stream called *streamName* contained within this **OMMSSStoredObject.**

void readFromStream**(IStream* stream, void* data, size_t size)**

Read *size* bytes from *stream* into the buffer at address *data*.

void readFromStream**(IStream* stream, OMByte* data, const OMUInt32 bytes, OMUInt32& bytesRead)**

Attempt to read *bytes* bytes from *stream* into the buffer at address *data*. The actual number of bytes read is returned in *bytesRead*.

void writeToStream**(IStream* stream, void* data, size_t size)**

Write *size* bytes from the buffer at address *data* to *stream*.

void writeToStream**(IStream* stream, const OMByte* data, const OMUInt32 bytes, OMUInt32& bytesWritten)**

Attempt to write *bytes* bytes from the buffer at address *data* to *stream*. The actual number of bytes written is returned in *bytesWritten*.

void readUInt8FromStream**(IStream* stream, OMUInt8& i)**

Read an OMUInt8 from *stream* into *i*.

void writeUInt8ToStream**(IStream* stream, OMUInt8& i)**

Write an OMUInt8 from *i* to *stream*.

void readUInt16FromStream**(IStream* stream, OMUInt16& i, bool reorderBytes)**

Read an OMUInt16 from *stream* into *i*. If *reorderBytes* is true then the bytes are reordered.

void writeUInt16ToStream**(IStream* stream, OMUInt16& i, bool reorderBytes)**

Write an OMUInt16 from *i* to *stream*. If *reorderBytes* is true then the bytes are reordered.

static void reorderUInt16**(OMUInt16& i)**

Reorder the OMUInt16 *i*.

void readUInt32FromStream**(IStream* stream, OMUInt32& i, bool reorderBytes)**

Read an OMUInt32 from *stream* into *i*. If *reorderBytes* is true then the bytes are reordered.

void writeUInt32ToStream**(IStream* stream, OMUInt32& i, bool reorderBytes)**

Write an OMUInt32 from *i* to *stream*. If *reorderBytes* is true then the bytes are reordered.

static void reorderUInt32**(OMUInt32& i)**

Reorder the OMUInt32 *i*.

void readUInt64FromStream**(IStream* stream, OMUInt64& i, bool reorderBytes)**

Read an OMUInt64 from *stream* into *i*. If *reorderBytes* is true then the bytes are reordered.

void writeUInt64ToStream**(IStream* stream, OMUInt64& i, bool reorderBytes)**

Write an OMUInt64 from *i* to *stream*. If *reorderBytes* is true then the bytes are reordered.

static void reorderUInt64**(OMUInt64& i)**

Reorder the OMUInt64 *i*.

void readUniqueObjectIdentificationFromStream**(IStream* stream, OMUniqueObjectIdentification& id, bool reorderBytes)**

Read a UniqueObjectIdentification from *stream* into *id*. If *reorderBytes* is true then the bytes are reordered.

void writeUniqueObjectIdentificationToStream**(IStream* stream, OMUniqueObjectIdentification& id, bool reorderBytes)**

Write a UniqueObjectIdentification from *id* to *stream*. If *reorderBytes* is true then the bytes are reordered.

void readUniqueMaterialIdentificationFromStream**(IStream* stream, OMUniqueMaterialIdentification& id, bool reorderBytes)**

Read a UniqueMaterialIdentification from *stream* into *id*. If *reorderBytes* is true then the bytes are reordered.

void writeUniqueMaterialIdentificationToStream**(IStream* stream, OMUniqueMaterialIdentification& id, bool reorderBytes)**

Write a UniqueMaterialIdentification from *id* to *stream*. If *reorderBytes* is true then the bytes are reordered.

static void reorderUniqueObjectIdentification**(OMUniqueObjectIdentification& id)**

Reorder the UniqueObjectIdentification *id*.

static void reorderUniqueMaterialIdentification**(OMUniqueMaterialIdentification& id)**

Reorder the UniqueMaterialIdentification *id*.

OMUInt64 streamSize**(IStream* stream) const**

Size of *stream* in bytes.

void streamSetSize**(IStream* stream, const OMUInt64 newSize)**

Set the size, in bytes, of *stream*

OMUInt64 streamPosition**(IStream* stream) const**

The current position for **readFromStream()** and **writeToStream()**, as an offset in bytes from the begining of the data stream.

void streamSetPosition**(IStream* stream, const OMUInt64 offset)**

Set the current position for **readFromStream()** and **writeToStream()**, as an offset in bytes from the begining of the data stream.

void closeStream**(IStream*& stream)**

Close *stream*.

void saveName**(const OMProperty& property, const wchar_t* name)**

The persisted value of *property* is its name. Write the property name and enter it into the property index.

void restoreName**(OMProperty& property, const wchar_t* name, size_t size)**

The persisted value of *property* is its name. Read (and check) the property name.

Class Members

Private members.

wchar_t* collectionIndexStreamName**(const wchar_t* collectionName)**

The stream name for the index of a collection named *collectionName*.

static void writeSignature**(OMRawStorage* rawStorage, const OMFileSignature& signature)**

Write the signature to the given raw storage.

static void writeSignature**(const wchar_t* fileName, const OMFileSignature& signature)**

Write the signature to the given file.

Developer Notes

Soon to be obsolete.

OMMSSStoredObject::close

void OMMSSStoredObject::close(void)

Close this OMMSSStoredObject.

Defined in: OMMSSStoredObject.cpp

Back to OMMSSStoredObject

OMMSSStoredObject::closeStream

void OMMSSStoredObject::closeStream(IStream*& *stream***)**

Close *stream*.

Defined in: OMMSSStoredObject.cpp

Parameters

stream
The stream to close.
Back to OMMSSStoredObject

OMMSSStoredObject::collectionIndexStreamName

wchar_t* OMMSSStoredObject::collectionIndexStreamName(const wchar_t* *collectionName***)**

The stream name for the index of a collection named *collectionName*.

Defined in: OMMSSStoredObject.cpp

Return Value

The stream name for the collection index.

Parameters

collectionName
The collection name.
Back to OMMSSStoredObject

OMMSSStoredObject::create

OMStoredObject* OMMSSStoredObject::create(const wchar_t* *name***)**

Create a new OMMSSStoredObject, named *name*, contained by this OMMSSStoredObject.

Defined in: OMMSSStoredObject.cpp

Return Value

A new OMMSSStoredObject contained by this OMMSSStoredObject.

Parameters

name
The name to be used for the new OMMSSStoredObject.
Back to OMMSSStoredObject

OMMSSStoredObject::createModify

OMMSSStoredObject* OMMSSStoredObject::createModify(const wchar_t* *fileName***, const OMByteOrder** *byteOrder***)**

Create a new root OMMSSStoredObject in the disk file *fileName*. The byte order of the newly created root is given by *byteOrder*.

Defined in: OMMSSStoredObject.cpp

Return Value

An OMMSSStoredObject representing the root object in the disk file.

Parameters

fileName
The name of the file to create. The file must not exist.
byteOrder
The desired byte ordering for the new file.
Back to OMMSSStoredObject

OMMSSStoredObject::createModify

OMMSSStoredObject* OMMSSStoredObject::createModify(OMRawStorage* *rawStorage*, **const OMByteOrder** *byteOrder***)**

Create a new root OMMSSStoredObject in the raw storage *rawStorage*. The byte order of the newly created root is given by *byteOrder*.

Defined in: OMMSSStoredObject.cpp

Return Value

An OMMSSStoredObject representing the root object.

Parameters

rawStorage
The raw storage in which to create the file.
byteOrder
The desired byte ordering for the new file.
Back to OMMSSStoredObject

OMMSSStoredObject::createStoredStream

OMStoredStream* OMMSSStoredObject::createStoredStream(const OMDataStream& *property***)**

Create an OMStoredStream representing the property *stream* contained within this OMMSSStoredObject.

Defined in: OMMSSStoredObject.cpp

Return Value

TBS

Parameters

property
TBS

Back to OMMSSStoredObject

OMMSSStoredObject::createStream

IStream* OMMSSStoredObject::createStream(const wchar_t* *streamName***)**

Create a stream called *streamName* contained within this OMMSSStoredObject.

Defined in: OMMSSStoredObject.cpp

Return Value

An open stream.

Parameters

streamName
The name of the stream to create.

Back to OMMSSStoredObject

OMMSSStoredObject::createWrite

OMMSSStoredObject* OMMSSStoredObject::createWrite(OMRawStorage* *rawStorage***, const OMByteOrder** *byteOrder***)**

Create a new root OMMSSStoredObject in the raw storage *rawStorage*. The byte order of the newly created root is given by *byteOrder*.

Defined in: OMMSSStoredObject.cpp

Return Value

An OMMSSStoredObject representing the root object.

Parameters

rawStorage
The raw storage in which to create the file.

byteOrder
The desired byte ordering for the new file.
Back to OMMSSStoredObject

OMMSSStoredObject::isRecognized

bool OMMSSStoredObject::isRecognized(const wchar_t* *fileName*, **OMFileSignature&** *signature***)**

Is the file named *fileName* a recognized file ? If so, the result is true, and the signature is returned in *signature*.

Defined in: OMMSSStoredObject.cpp

Return Value

True if the file is recognized, false otherwise.

Parameters

fileName
The name of the file to check.
signature
If recognized, the file signature.
Back to OMMSSStoredObject

OMMSSStoredObject::isRecognized

bool OMMSSStoredObject::isRecognized(OMRawStorage* *rawStorage*, **OMFileSignature&** *signature***)**

Does *rawStorage* contain a recognized file ? If so, the result is true, and the signature is returned in *signature*.

Defined in: OMMSSStoredObject.cpp

Return Value

True if the OMRawStorage contains a recognized file, false otherwise.

Parameters

rawStorage
The OMRawStorage to check.
signature
If recognized, the file signature.
Back to OMMSSStoredObject

OMMSSStoredObject::isRecognized

bool OMMSSStoredObject::isRecognized(const OMFileSignature& *signature***)**

Is *signature* recognized ? If so, the result is true.

Defined in: OMMSSStoredObject.cpp

Return Value

True if the signature is recognized, false otherwise.

Parameters

signature
The signature to check.

Back to OMMSSStoredObject

OMMSSStoredObject::OMMSSStoredObject

OMMSSStoredObject::OMMSSStoredObject(IStorage* *s***)**

Constructor.

Defined in: OMMSSStoredObject.cpp

Parameters

s
The IStorage for the persistent representation of this OMMSSStoredObject.

Back to OMMSSStoredObject

OMMSSStoredObject::open

OMStoredObject* OMMSSStoredObject::open(const wchar_t* *name***)**

Open an exsiting OMMSSStoredObject, named *name*, contained by this OMMSSStoredObject.

Defined in: OMMSSStoredObject.cpp

Return Value

The existing OMMSSStoredObject contained by this OMMSSStoredObject.

Parameters

name
The name of the existing OMMSSStoredObject.

Back to OMMSSStoredObject

OMMSSStoredObject::openModify

OMMSSStoredObject* OMMSSStoredObject::openModify(const wchar_t* *fileName***)**

Open the root OMMSSStoredObject in the disk file *fileName* for modification.

Defined in: OMMSSStoredObject.cpp

Return Value

An OMMSSStoredObject representing the root object in the disk file.

Parameters

fileName
The name of the file to open. The file must already exist.

Back to OMMSSStoredObject

OMMSSStoredObject::openModify

OMMSSStoredObject* OMMSSStoredObject::openModify(OMRawStorage* *rawStorage***)**

Open the root OMMSSStoredObject in the raw storage *rawStorage* for modification.

Defined in: OMMSSStoredObject.cpp

Return Value

An OMMSSStoredObject representing the root object.

Parameters

rawStorage
The raw storage in which to open the file.

Back to OMMSSStoredObject

OMMSSStoredObject::openRead

OMMSSStoredObject* OMMSSStoredObject::openRead(OMRawStorage* *rawStorage***)**

Open the root OMMSSStoredObject in the raw storage *rawStorage* for reading only.

Defined in: OMMSSStoredObject.cpp

Return Value

An OMMSSStoredObject representing the root object.

Parameters

rawStorage
The raw storage in which to open the file.

Back to OMMSSStoredObject

OMMSSStoredObject::openRead

OMMSSStoredObject* OMMSSStoredObject::openRead(const wchar_t* *fileName***)**

Open the root OMMSSStoredObject in the disk file *fileName* for reading only.

Defined in: OMMSSStoredObject.cpp

Return Value

An OMMSSStoredObject representing the root object in the disk file.

Parameters

fileName
The name of the file to open. The file must already exist.

Back to OMMSSStoredObject

OMMSSStoredObject::openStoredStream

OMStoredStream* OMMSSStoredObject::openStoredStream(const OMDataStream& *property***)**

Open the OMStoredStream representing the property *property* contained within this OMMSSStoredObject.

Defined in: OMMSSStoredObject.cpp

Return Value

TBS

Parameters

property
TBS

Back to OMMSSStoredObject

OMMSSStoredObject::openStream

IStream* OMMSSStoredObject::openStream(const wchar_t* *streamName*)

Open a stream called *streamName* contained within this OMMSSStoredObject.

Defined in: OMMSSStoredObject.cpp

Return Value

An open stream.

Parameters

streamName
The name of the stream to open.

Back to OMMSSStoredObject

OMMSSStoredObject::read

void OMMSSStoredObject::read(OMPropertyId *propertyId*, int *type*, void* *start*, size_t *size*)

Read a property value from this OMMSSStoredObject. The property value is read into a buffer which occupies *size* bytes at the address *start*. The property id is *propertyId*. The property type is *type*.

Defined in: OMMSSStoredObject.cpp

Parameters

propertyId
The property id.

type
The property type.

start
The start address of the buffer to hold the property value.

size
The size of the buffer in bytes.

Back to OMMSSStoredObject

OMMSSStoredObject::readFromStream

void OMMSSStoredObject::readFromStream(IStream* *stream*, void* *data*, size_t *size*)

Read *size* bytes from *stream* into the buffer at address *data*.

Defined in: OMMSSStoredObject.cpp

Parameters

stream
: The stream from which to read.

data
: The buffer into which the bytes are read.

size
: The number of bytes to read.

Back to OMMSSStoredObject

OMMSSStoredObject::readFromStream

void OMMSSStoredObject::readFromStream(IStream* *stream*, OMByte* *data*, const OMUInt32 *bytes*, OMUInt32& *bytesRead*)

Attempt to read *bytes* bytes from *stream* into the buffer at address *data*. The actual number of bytes read is returned in *bytesRead*.

Defined in: OMMSSStoredObject.cpp

Parameters

stream
: The stream from which to read.

data
: The buffer into which the bytes are read.

bytes
: The number of bytes to write

bytesRead
: The actual number of bytes read.

Back to OMMSSStoredObject

OMMSSStoredObject::readUInt16FromStream

void OMMSSStoredObject::readUInt16FromStream(IStream* *stream*, OMUInt16& *i*, bool *reorderBytes*)

Read an OMUInt16 from *stream* into *i*. If *reorderBytes* is true then the bytes are reordered.

Defined in: OMMSSStoredObject.cpp

Parameters

stream
: The stream from which to read.

i
: The resulting OMUInt16.

reorderBytes
: If true then reorder the bytes.

Back to OMMSSStoredObject

OMMSSStoredObject::readUInt32FromStream

void OMMSSStoredObject::readUInt32FromStream(IStream* *stream*, **OMUInt32&** *i*, **bool** *reorderBytes***)**

Read an OMUInt32 from *stream* into *i*. If *reorderBytes* is true then the bytes are reordered.

Defined in: OMMSSStoredObject.cpp

Parameters

stream
: The stream from which to read.

i
: The resulting OMUInt32.

reorderBytes
: If true then reorder the bytes.

Back to OMMSSStoredObject

OMMSSStoredObject::readUInt64FromStream

void OMMSSStoredObject::readUInt64FromStream(IStream* *stream*, **OMUInt64&** *i*, **bool** *reorderBytes***)**

Read an OMUInt64 from *stream* into *i*. If *reorderBytes* is true then the bytes are reordered.

Defined in: OMMSSStoredObject.cpp

Parameters

stream
: The stream from which to read.

i
: The resulting OMUInt64.

reorderBytes
: If true then reorder the bytes.

Back to OMMSSStoredObject

OMMSSStoredObject::readUInt8FromStream

void OMMSSStoredObject::readUInt8FromStream(IStream* *stream*, **OMUInt8&** *i***)**

Read an OMUInt8 from *stream* into *i*.

Defined in: OMMSSStoredObject.cpp

Parameters

stream
The stream from which to read.
i
The resulting OMUInt8.
Back to OMMSSStoredObject

OMMSSStoredObject::readUniqueMaterialIdentificationFromStream

void OMMSSStoredObject::readUniqueMaterialIdentificationFromStream(IStream* *stream*, **OMUniqueMaterialIdentification&** *id*, **bool** *reorderBytes***)**

Read a UniqueMaterialIdentification from *stream* into *id*. If *reorderBytes* is true then the bytes are reordered.

Defined in: OMMSSStoredObject.cpp

Parameters

stream
The stream from which to read.
id
The resulting OMUniqueMaterialIdentification.
reorderBytes
If true then reorder the bytes.
Back to OMMSSStoredObject

OMMSSStoredObject::readUniqueObjectIdentificationFromStream

void OMMSSStoredObject::readUniqueObjectIdentificationFromStream(IStream* *stream*, **OMUniqueObjectIdentification&** *id*, **bool** *reorderBytes***)**

Read a UniqueObjectIdentification from *stream* into *id*. If *reorderBytes* is true then the bytes are reordered.

Defined in: OMMSSStoredObject.cpp

Parameters

stream
The stream from which to read.
id
The resulting OMUniqueObjectIdentification.
reorderBytes
If true then reorder the bytes.
Back to OMMSSStoredObject

OMMSSStoredObject::reorderUInt16

void OMMSSStoredObject::reorderUInt16(OMUInt16& *i*)

Reorder the OMUInt16 *i*.

Defined in: OMMSSStoredObject.cpp

Parameters

i
The OMUInt16 to reorder.

Back to OMMSSStoredObject

OMMSSStoredObject::reorderUInt32

void OMMSSStoredObject::reorderUInt32(OMUInt32& *i*)

Reorder the OMUInt32 *i*.

Defined in: OMMSSStoredObject.cpp

Parameters

i
The OMUInt32 to reorder.

Back to OMMSSStoredObject

OMMSSStoredObject::reorderUInt64

void OMMSSStoredObject::reorderUInt64(OMUInt64& *i*)

Reorder the OMUInt64 *i*.

Defined in: OMMSSStoredObject.cpp

Parameters

i
The OMUInt64 to reorder.

Back to OMMSSStoredObject

OMMSSStoredObject::reorderUniqueMaterialIdentification

void OMMSSStoredObject::reorderUniqueMaterialIdentification(OMUniqueMaterialIdentification& *id*)

Reorder the OMUniqueMaterialIdentification *id*.

Defined in: OMMSSStoredObject.cpp

Parameters

id
The OMUniqueMaterialIdentification to reorder.

Back to OMMSSStoredObject

OMMSSStoredObject::reorderUniqueObjectIdentification

void OMMSSStoredObject::reorderUniqueObjectIdentification(OMUniqueObjectIdentification& *id*)

Reorder the OMUniqueObjectIdentification *id*.

Defined in: OMMSSStoredObject.cpp

Parameters

id
The OMUniqueObjectIdentification to reorder.

Back to OMMSSStoredObject

OMMSSStoredObject::restore

void OMMSSStoredObject::restore(OMWeakReferenceSet& *set*, size_t *externalSize*)

Restore the OMWeakReferenceSet *set* into this OMMSSStoredObject.

Defined in: OMMSSStoredObject.cpp

Parameters

set
TBS
externalSize
TBS

Back to OMMSSStoredObject

OMMSSStoredObject::restore

void OMMSSStoredObject::restore(const wchar_t* *collectionName*, OMUniqueObjectIdentification*& *index*, size_t & *count*, OMPropertyTag& *tag*, OMPropertyId& *keyPropertyId*)

Restore a collection (vector/set) of weak references.

Defined in: OMMSSStoredObject.cpp

Parameters

collectionName
The name of the collection.
index
The unique identifications of the targets.
count
Count of targets.
tag
A tag identifying the collection in which each of the targets reside.
keyPropertyId
The id of the property whose value is the unique identifier of objects in the target set.
Back to OMMSSStoredObject

OMMSSStoredObject::restore

void OMMSSStoredObject::restore(OMStrongReferenceSet& *set*, **size_t** *externalSize***)**

Restore the OMStrongReferenceSet *set* into this OMMSSStoredObject.

Defined in: OMMSSStoredObject.cpp

Parameters

set
TBS
externalSize
TBS
Back to OMMSSStoredObject

OMMSSStoredObject::restore

void OMMSSStoredObject::restore(OMDataStream& *stream*, **size_t** *ANAME***)**

Restore the OMDataStream *stream* into this OMMSSStoredObject.

Defined in: OMMSSStoredObject.cpp

Parameters

stream
TBS
ANAME
TBS

Back to OMMSSStoredObject

OMMSSStoredObject::restore

void OMMSSStoredObject::restore(OMWeakReferenceVector& *vector*, size_t *externalSize*)

Restore the OMWeakReferenceVector *vector* into this OMMSSStoredObject.

Defined in: OMMSSStoredObject.cpp

Parameters

vector
: TBS

externalSize
: TBS

Back to OMMSSStoredObject

OMMSSStoredObject::restore

void OMMSSStoredObject::restore(OMPropertyTable*& *table*)

Restore the OMPropertyTable in this OMMSSStoredObject.

Defined in: OMMSSStoredObject.cpp

Parameters

table
: A pointer to the newly restored OMPropertyTable by reference.

Back to OMMSSStoredObject

OMMSSStoredObject::restore

void OMMSSStoredObject::restore(OMStoredVectorIndex*& *vector*)

Restore the vector named *vectorName* into this OMMSSStoredObject.

Defined in: OMMSSStoredObject.cpp

Return Value

The newly restored OMStoredVectorIndex.

Parameters

vector
The name of the vector.
Back to OMMSSStoredObject

OMMSSStoredObject::restore

void OMMSSStoredObject::restore(OMPropertySet& *properties***)**

Restore the OMPropertySet *properties* into this OMMSSStoredObject.

Defined in: OMMSSStoredObject.cpp

Parameters

properties
The OMPropertySet to restore.
Back to OMMSSStoredObject

OMMSSStoredObject::restore

void OMMSSStoredObject::restore(OMWeakReference& *singleton***, size_t** *ANAME***)**

Restore the OMWeakReference *singleton* into this OMMSSStoredObject.

Defined in: OMMSSStoredObject.cpp

Parameters

singleton
TBS
ANAME
TBS
Back to OMMSSStoredObject

OMMSSStoredObject::restore

void OMMSSStoredObject::restore(OMSimpleProperty& *property***, size_t** *externalSize***)**

Restore the OMSimpleProperty *property* into this OMMSSStoredObject.

Defined in: OMMSSStoredObject.cpp

Parameters

property

TBS
externalSize
TBS
Back to OMMSSStoredObject

OMMSSStoredObject::restore

void OMMSSStoredObject::restore(OMStrongReference& *singleton*, **size_t** *externalSize***)**

Restore the OMStrongReference *singleton* into this OMMSSStoredObject.

Defined in: OMMSSStoredObject.cpp

Parameters

singleton
TBS
externalSize
TBS
Back to OMMSSStoredObject

OMMSSStoredObject::restore

void OMMSSStoredObject::restore(OMStrongReferenceVector& *vector*, **size_t** *externalSize***)**

Restore the OMStrongReferenceVector *vector* into this OMMSSStoredObject.

Defined in: OMMSSStoredObject.cpp

Parameters

vector
TBS
externalSize
TBS
Back to OMMSSStoredObject

OMMSSStoredObject::restore

void OMMSSStoredObject::restore(OMPropertyId *propertyId*, **OMStoredForm** *storedForm*, **OMUniqueObjectIdentification&** *id*, **OMPropertyTag&** *tag*, **OMPropertyId&** *keyPropertyId***)**

Restore a single weak reference.

Defined in: OMMSSStoredObject.cpp

Parameters

propertyId
The property id.
storedForm
The property type.
id
The unique identification of the target.
tag
A tag identifying the collection in which the target resides.
keyPropertyId
The id of the property whose value is the unique identifier of objects in the target set.
Back to OMMSSStoredObject

OMMSSStoredObject::restore

void OMMSSStoredObject::restore(OMStoredObjectIdentification& *id*)

Restore the **OMStoredObjectIdentification** of this OMMSSStoredObject into *id*.

Defined in: OMMSSStoredObject.cpp

Parameters

id
The **OMStoredObjectIdentification** of this OMMSSStoredObject.
Back to OMMSSStoredObject

OMMSSStoredObject::restore

void OMMSSStoredObject::restore(OMStoredSetIndex*& *set*)

Restore the set named *setName* into this OMMSSStoredObject.

Defined in: OMMSSStoredObject.cpp

Return Value

The newly restored OMStoredSetIndex.

Parameters

set
The name of the set.
Back to OMMSSStoredObject

OMMSSStoredObject::restoreName

void OMMSSStoredObject::restoreName(OMProperty& *property*, **const wchar_t*** *ANAME*, *name***)**

The persisted value of *property* is its name. Read (and check) the property name.

Defined in: OMMSSStoredObject.cpp

Parameters

property
: The property.

ANAME
: The expected property name.

name
: The (expected) size of the property name.

Back to OMMSSStoredObject

OMMSSStoredObject::save

void OMMSSStoredObject::save(const OMStrongReference& *singleton***)**

Save the OMStrongReference *singleton* in this OMMSSStoredObject.

Defined in: OMMSSStoredObject.cpp

Parameters

singleton
: TBS

Back to OMMSSStoredObject

OMMSSStoredObject::save

void OMMSSStoredObject::save(const OMStoredVectorIndex* *vector*, **const wchar_t*** *vectorName***)**

Save the OMStoredVectorIndex *vector* in this OMMSSStoredObject, the vector is named *vectorName*.

Defined in: OMMSSStoredObject.cpp

Parameters

vector
: The OMStoredVectorIndex to save.

vectorName
: The name of the vector.

Back to OMMSSStoredObject

OMMSSStoredObject::save

void OMMSSStoredObject::save(const wchar_t* *collectionName*, **const OMUniqueObjectIdentification*** *index*, **size_t** *count*, **OMPropertyTag** *tag*, **OMPropertyId** *keyPropertyId***)**

Save a collection (vector/set) of weak references.

Defined in: OMMSSStoredObject.cpp

Parameters

collectionName
: The name of the collection.

index
: The unique identifications of the targets.

count
: Count of targets.

tag
: A tag identifying the collection in which each of the targets reside.

keyPropertyId
: The id of the property whose value is the unique identifier of objects in the target set.

Back to OMMSSStoredObject

OMMSSStoredObject::save

void OMMSSStoredObject::save(const OMWeakReferenceVector& *vector***)**

Save the OMWeakReferenceVector *vector* in this OMMSSStoredObject.

Defined in: OMMSSStoredObject.cpp

Parameters

vector
: TBS

Back to OMMSSStoredObject

OMMSSStoredObject::save

void OMMSSStoredObject::save(const OMStrongReferenceSet& *set***)**

Save the OMStrongReferenceSet *set* in this OMMSSStoredObject.

Defined in: OMMSSStoredObject.cpp

Parameters

set
The set of strong references to save.

Back to OMMSSStoredObject

OMMSSStoredObject::save

void OMMSSStoredObject::save(const OMWeakReferenceSet& *set***)**

Save the OMWeakReferenceSet *set* in this OMMSSStoredObject.

Defined in: OMMSSStoredObject.cpp

Parameters

set
TBS

Back to OMMSSStoredObject

OMMSSStoredObject::save

void OMMSSStoredObject::save(OMPropertyId *propertyId***, OMStoredForm** *storedForm***, const OMUniqueObjectIdentification&** *id***, OMPropertyTag** *tag***, OMPropertyId** *keyPropertyId***)**

Save a single weak reference.

Defined in: OMMSSStoredObject.cpp

Parameters

propertyId
The property id.
storedForm
The property type.
id
The unique identification of the target.
tag
A tag identifying the collection in which the target resides.
keyPropertyId
The id of the property whose value is the unique identifier of objects in the target set.

Back to OMMSSStoredObject

OMMSSStoredObject::save

void OMMSSStoredObject::save(const OMWeakReference& *singleton***)**

Save the OMWeakReference *singleton* in this OMMSSStoredObject.

Defined in: OMMSSStoredObject.cpp

Parameters

singleton
TBS

Back to OMMSSStoredObject

OMMSSStoredObject::save

void OMMSSStoredObject::save(const OMStrongReferenceVector& *vector***)**

Save the OMStrongReferenceVector *vector* in this OMMSSStoredObject.

Defined in: OMMSSStoredObject.cpp

Parameters

vector
The vector of strong references to save.

Back to OMMSSStoredObject

OMMSSStoredObject::save

void OMMSSStoredObject::save(const OMPropertyTable* *table***)**

Save the OMPropertyTable *table* in this OMMSSStoredObject.

Defined in: OMMSSStoredObject.cpp

Parameters

table
The table to save.

Back to OMMSSStoredObject

OMMSSStoredObject::save

void OMMSSStoredObject::save(const OMPropertySet& *properties***)**

Save the OMPropertySet *properties* in this OMMSSStoredObject.

Defined in: OMMSSStoredObject.cpp

Parameters

properties
The OMPropertySet to save.

Back to OMMSSStoredObject

OMMSSStoredObject::save

void OMMSSStoredObject::save(const OMStoredObjectIdentification& *id***)**

Save the **OMStoredObjectIdentification** *id* in this OMMSSStoredObject.

Defined in: OMMSSStoredObject.cpp

Parameters

id
The **OMStoredObjectIdentification** of this OMMSSStoredObject**.**

Back to OMMSSStoredObject

OMMSSStoredObject::save

void OMMSSStoredObject::save(const OMStoredSetIndex* *set***, const wchar_t*** *setName***)**

Save the OMStoredSetIndex *set* in this OMMSSStoredObject, the set is named *setName*.

Defined in: OMMSSStoredObject.cpp

Parameters

set
The OMStoredSetIndex to save.

setName
The name of the set.

Back to OMMSSStoredObject

OMMSSStoredObject::save

void OMMSSStoredObject::save(const OMDataStream& *stream***)**

Save the OMDataStream *stream* in this OMMSSStoredObject.

Defined in: OMMSSStoredObject.cpp

Parameters

stream
The OMDataStream to save.

Back to OMMSSStoredObject

OMMSSStoredObject::save

void OMMSSStoredObject::save(const OMSimpleProperty& *property***)**

Save the OMSimpleProperty *property* in this OMMSSStoredObject.

Defined in: OMMSSStoredObject.cpp

Parameters

property
TBS

Back to OMMSSStoredObject

OMMSSStoredObject::saveName

void OMMSSStoredObject::saveName(const OMProperty& *property***, const wchar_t*** *name***) const**

The persisted value of *property* is its name. Write the property name and enter it into the property index.

Defined in: OMMSSStoredObject.cpp

Parameters

property
The property.

name
TBS

Back to OMMSSStoredObject

OMMSSStoredObject::streamPosition

OMUInt64 OMMSSStoredObject::streamPosition(void) const

The current position for **readFromStream()** and **writeToStream()**, as an offset in bytes from the begining of the data stream.

Defined in: OMMSSStoredObject.cpp

Return Value

The current position for **readFromStream()** and **writeToStream()**, as an offset in bytes from the begining of the data stream.

Back to OMMSSStoredObject

OMMSSStoredObject::streamSetPosition

void OMMSSStoredObject::streamSetPosition(IStream* *stream*) const

Set the current position for **readFromStream()** and **writeToStream()**, as an offset in bytes from the begining of the data stream.

Defined in: OMMSSStoredObject.cpp

Parameters

stream
: The position to use for subsequent calls to readFromStream() and writeToStream() on this stream. The position is specified as an offset in bytes from the begining of the data stream.

Back to OMMSSStoredObject

OMMSSStoredObject::streamSetSize

void OMMSSStoredObject::streamSetSize(IStream* *stream*, const OMUInt64 *newSize*)

Set the size, in bytes, of *stream*

Defined in: OMMSSStoredObject.cpp

Parameters

stream
: An open stream.

newSize
: The new size for the stream.

Back to OMMSSStoredObject

OMMSSStoredObject::streamSize

OMUInt64 OMMSSStoredObject::streamSize(IStream* *stream*) const

Size of *stream* in bytes.

Defined in: OMMSSStoredObject.cpp

Return Value

The size of *stream* in bytes

Parameters

stream
An open stream.

Back to OMMSSStoredObject

OMMSSStoredObject::validate

void OMMSSStoredObject::validate(const OMPropertySet* *propertySet*, **const OMStoredPropertySetIndex*** *propertySetIndex***)**

Check that the OMPropertySet *propertySet* is consistent with the OMStoredPropertySetIndex propertySetIndex.

Defined in: OMMSSStoredObject.cpp

Parameters

propertySet
The OMPropertySet to validate.

propertySetIndex
The OMStoredPropertySetIndex to validate.

Back to OMMSSStoredObject

OMMSSStoredObject::write

void OMMSSStoredObject::write(OMPropertyId *propertyId*, **OMStoredForm** *storedForm*, **void*** *start*, **size_t** *size***)**

Write a property value to this OMMSSStoredObject. The property value to be written occupies *size* bytes at the address *start*. The property id is *propertyId*. The property type is *type*.

Defined in: OMMSSStoredObject.cpp

Parameters

propertyId
The property id.

storedForm
The property type.

start
The start address of the property value.

size
The size of the property value in bytes.
Back to OMMSSStoredObject

OMMSSStoredObject::writeSignature

void OMMSSStoredObject::writeSignature(const wchar_t* *fileName*, **const OMFileSignature&** *signature***)**

Write the signature to the given file.

Defined in: OMMSSStoredObject.cpp

Parameters

fileName
The file name.
signature
The signature.
Back to OMMSSStoredObject

OMMSSStoredObject::writeSignature

void OMMSSStoredObject::writeSignature(OMRawStorage* *rawStorage*, **const OMFileSignature&** *signature***)**

Write the signature to the given raw storage.

Defined in: OMMSSStoredObject.cpp

Parameters

rawStorage
The raw storage.
signature
The signature.
Back to OMMSSStoredObject

OMMSSStoredObject::writeToStream

void OMMSSStoredObject::writeToStream(IStream* *stream*, **void*** *data*, **size_t** *size***)**

Write *size* bytes from the buffer at address *data* to *stream*.

Defined in: OMMSSStoredObject.cpp

Parameters

stream
The stream on which to write.
data
The buffer to write.
size
The number of bytes to write.
Back to OMMSSStoredObject

OMMSSStoredObject::writeToStream

void OMMSSStoredObject::writeToStream(IStream* *stream*, const OMByte* *data*, const OMUInt32 *bytes*, OMUInt32& *bytesWritten*)

Attempt to write *bytes* bytes from the buffer at address *data* to *stream*. The actual number of bytes written is returned in *bytesWritten*.

Defined in: OMMSSStoredObject.cpp

Parameters

stream
The stream on which to write.
data
The buffer to write.
bytes
The number of bytes to write
bytesWritten
The actual number of bytes written.
Back to OMMSSStoredObject

OMMSSStoredObject::writeUInt16ToStream

void OMMSSStoredObject::writeUInt16ToStream(IStream* *stream*, OMUInt16& *i*, bool *reorderBytes*)

Write an OMUInt16 from *i* to *stream*. If *reorderBytes* is true then the bytes are reordered.

Defined in: OMMSSStoredObject.cpp

Parameters

stream
The stream to write to.
i
The OMUInt16 to write.
reorderBytes
If true then reorder the bytes.

Back to OMMSSStoredObject

OMMSSStoredObject::writeUInt32ToStream

void OMMSSStoredObject::writeUInt32ToStream(IStream* *stream*, **OMUInt32&** *i*, **bool** *reorderBytes***)**

Write an OMUInt32 from *i* to *stream*. If *reorderBytes* is true then the bytes are reordered.

Defined in: OMMSSStoredObject.cpp

Parameters

stream
: The stream to write to.

i
: The OMUInt32 to write.

reorderBytes
: If true then reorder the bytes.

Back to OMMSSStoredObject

OMMSSStoredObject::writeUInt64ToStream

void OMMSSStoredObject::writeUInt64ToStream(IStream* *stream*, **OMUInt64&** *i*, **bool** *reorderBytes***)**

Write an OMUInt64 from *i* to *stream*. If *reorderBytes* is true then the bytes are reordered.

Defined in: OMMSSStoredObject.cpp

Parameters

stream
: The stream to write to.

i
: The OMUInt64 to write.

reorderBytes
: If true then reorder the bytes.

Back to OMMSSStoredObject

OMMSSStoredObject::writeUInt8ToStream

void OMMSSStoredObject::writeUInt8ToStream(IStream* *stream*, **OMUInt8&** *i***)**

Write an OMUInt8 from *i* to *stream*.

Defined in: OMMSSStoredObject.cpp

Parameters

stream
The stream to write to.
i
The OMUInt8 to write.
Back to OMMSSStoredObject

OMMSSStoredObject::writeUniqueMaterialIdentificationToStream

void OMMSSStoredObject::writeUniqueMaterialIdentificationToStream(IStream* *stream*, **OMUniqueMaterialIdentification&** *id*, **bool** *reorderBytes***)**

Write a UniqueMaterialIdentification from *id* to *stream*. If *reorderBytes* is true then the bytes are reordered.

Defined in: OMMSSStoredObject.cpp

Parameters

stream
The stream to write to.
id
The OMUniqueMaterialIdentification to write.
reorderBytes
If true then reorder the bytes.
Back to OMMSSStoredObject

OMMSSStoredObject::writeUniqueObjectIdentificationToStream

void OMMSSStoredObject::writeUniqueObjectIdentificationToStream(IStream* *stream*, **OMUniqueObjectIdentification&** *id*, **bool** *reorderBytes***)**

Write a UniqueObjectIdentification from *id* to *stream*. If *reorderBytes* is true then the bytes are reordered.

Defined in: OMMSSStoredObject.cpp

Parameters

stream
The stream to write to.
id
The OMUniqueObjectIdentification to write.
reorderBytes
If true then reorder the bytes.
Back to OMMSSStoredObject

OMMSSStoredObject::~OMMSSStoredObject

OMMSSStoredObject::~OMMSSStoredObject(void)

Destructor.

Defined in: OMMSSStoredObject.cpp

Back to OMMSSStoredObject

OMMSSStoredStream class

OMMSSStoredStream **class OMMSSStoredStream: public OMStoredStream**

Implementation of OMStoredStream for Microsoft Structured Storage.

Defined in: OMMSSStoredStream.h

Author

Tim Bingham - tjb - (Avid Technology, Inc.)

Class Members

Public members.

OMMSSStoredStream(IStream* stream)
: Constructor.

~OMMSSStoredStream(void)
: Destructor.

virtual void read(void* data, size_t size) const
: Read *size* bytes from this OMStoredStream into the buffer at address *data*.

virtual void read(OMByte* data, const OMUInt32 bytes, OMUInt32& bytesRead) const
: Attempt to read *bytes* bytes from this OMStoredStream into the buffer at address *data*. The actual number of bytes read is returned in *bytesRead*.

virtual void write(void* data, size_t size)
: Write *size* bytes from the buffer at address *data* to this OMStoredStream.

virtual void write(const OMByte* data, const OMUInt32 bytes, OMUInt32& bytesWritten)
: Attempt to write *bytes* bytes from the buffer at address *data* to this OMStoredStream. The actual number of bytes written is returned in *bytesWritten*.

virtual OMUInt64 size(void) const
: The size of this OMStoredStream in bytes.

virtual void setSize(const OMUInt64 newSize)
: Set the size of this OMStoredStream to *bytes*.

virtual OMUInt64 position(void) const
: The current position for **read()** and **write()**, as an offset in bytes from the begining of this OMStoredStream.

virtual void setPosition(const OMUInt64 offset)
: Set the current position for **read()** and **write()**, as an offset in bytes from the begining of this OMStoredStream.

virtual void close(void)

Close this OMStoredStream.

Class Members

Private members.

OMObject class

OMObject **class OMObject**

Abstract base class for all objects known to the Object Manager.

Defined in: OMObject.h

Author

Tim Bingham - tjb - (Avid Technology, Inc.)

Class Members

Public members.
virtual ~OMObject(void)
Destructor.

OMObjectDirectory class

OMObjectDirectory **class OMObjectDirectory**

Debug only data structure for tracking objects by name.

Defined in: OMObjectDirectory.h

Author

Tim Bingham - tjb - (Avid Technology, Inc.)

OMObjectReference class

OMObjectReference **class OMObjectReference**

Persistent references to persistent objects.

Defined in: OMObjectReference.h

Author

Tim Bingham - tjb - (Avid Technology, Inc.)

Class Members

Public members.

OMObjectReference**(void)**
: Constructor.

OMObjectReference**(OMProperty* property)**
: Constructor.

OMObjectReference**(const OMObjectReference&)**
: Copy constructor.

virtual ~OMObjectReference**(void)**
: Destructor.

virtual bool isVoid**(void) const**
: Is this **OMObjectReference** void ?

OMObjectReference& operator=**(const OMObjectReference& rhs)**
: Assignment. This operator provides value semantics for OMContainer. This operator does not provide assignment of object references.

bool operator==**(const OMObjectReference& rhs) const**
: Equality. This operator provides value semantics for OMContainer. This operator does not provide equality of object references.

virtual void save(void) const
: Save this **OMObjectReference.**

virtual void close(void)
: Close this **OMObjectReference.**

virtual void detach(void)
: Detach this **OMObjectReference.**

virtual void restore(void)
: Restore this **OMObjectReference.**

virtual OMStorable* getValue(void) const
: Get the value of this **OMObjectReference.** The value is a pointer to the referenced OMStorable.

virtual OMStorable* pointer**(void) const**
: The value of this **OMObjectReference** as a pointer. This function provides low-level access. If the object exits but has not yet been loaded then the value returned is 0.

Class Members

Protected members.

OMProperty* _property
: The containing property.

OMStorable* _pointer
: A pointer to the actual object.

OMObjectReference::isVoid

bool OMObjectReference::isVoid(void) const

Is this OMObjectReference void ?

Defined in: OMObjectReference.cpp

Return Value

True if this OMObjectReference is void, false otherwise.

Back to OMObjectReference

OMObjectReference::OMObjectReference

OMObjectReference::OMObjectReference(void)

Constructor.

Defined in: OMObjectReference.cpp

Back to OMObjectReference

OMObjectReference::OMObjectReference

OMObjectReference::OMObjectReference(const OMObjectReference& *rhs***)**

Copy constructor.

Defined in: OMObjectReference.cpp

Parameters

rhs
The OMObjectReference to copy.

Back to OMObjectReference

OMObjectReference::OMObjectReference

OMObjectReference::OMObjectReference(OMProperty* *property***)**

Constructor.

Defined in: OMObjectReference.cpp

Parameters

property
The OMProperty that contains this OMObjectReference.

Back to OMObjectReference

OMObjectReference::operator=

OMObjectReference& OMObjectReference::operator=(const OMObjectReference& *rhs*)

Assignment. This operator provides value semantics for OMContainer. This operator does not provide assignment of object references.

Defined in: OMObjectReference.cpp

Return Value

The OMObjectReference resulting from the assignment.

Parameters

rhs
The OMObjectReference to be assigned.

Back to OMObjectReference

OMObjectReference::operator==

bool OMObjectReference::operator==(const OMObjectReference& *rhs*) const

Equality. This operator provides value semantics for OMContainer. This operator does not provide equality of object references.

Defined in: OMObjectReference.cpp

Return Value

True if the values are the same, false otherwise.

Parameters

rhs
The OMObjectReference to be compared.

Back to OMObjectReference

OMObjectReference::pointer

OMStorable* OMObjectReference::pointer(void) const

The value of this OMObjectReference as a pointer. This function provides low-level access. If the object exits but has not yet been loaded then the value returned is 0.

Defined in: OMObjectReference.cpp

Return Value

A pointer to the referenced object, if any, otherwise 0.

Back to OMObjectReference

OMObjectReference::~OMObjectReference

OMObjectReference::~OMObjectReference(void)

Destructor.

Defined in: OMObjectReference.cpp

Back to OMObjectReference

OMObjectSet class

OMObjectSet **class OMObjectSet: public OMReferenceContainer**

Abstract base class for persistent object reference set properties supported by the Object Manager.

Defined in: OMObjectSet.h

Author

Tim Bingham - tjb - (Avid Technology, Inc.)

Class Members

Public members.
virtual OMObject* remove(void* identification)
Remove the OMObject identified by *identification* from this **OMObjectSet.**
virtual bool contains(void* identification) const
Does this **OMObjectSet** contain an OMObject identified by *identification* ?
virtual bool findObject(void* identification, OMObject*& object) const
Find the OMObject in this **OMObjectSet** identified by *identification*. If the object is found it is returned in *object* and the result is **true** . If the object is not found the result is **false** .

OMObjectVector class

OMObjectVector **class OMObjectVector: public OMReferenceContainer**

Abstract base class for elastic sequential collections of objects supported by the Object Manager. Objects are accessible by index. The order of objects is determined externally. Duplicate objects are allowed.

Defined in: OMObjectVector.h

Author

Tim Bingham - tjb - (Avid Technology, Inc.)

Class Members

Public members.
virtual OMObject* setObjectAt(const OMObject* object, const size_t index)
Set the value of this **OMObjectVector** at position *index* to *object*.
virtual OMObject* getObjectAt(const size_t index) const
The value of this **OMObjectVector** at position *index*.
virtual void appendObject(const OMObject* object)
Append the given *OMObject object* to this **OMObjectVector**.
virtual void prependObject(const OMObject* object)
Prepend the given *OMObject object* to this **OMObjectVector**.
virtual OMObject* removeObjectAt(const size_t index)
Remove the object from this **OMObjectVector** at position *index*. Existing objects in this **OMObjectVector** at *index* + 1 and higher are shifted down one index position.
virtual void insertObjectAt(const OMObject* object, const size_t index)
Insert *object* into this **OMObjectVector** at position *index*. Existing objects at *index* and higher are shifted up one index position.

OMOStream class

OMOStream **class OMOStream**

Simple, platform independent, text output stream for diagnostic (debug only) use by the Object Manager.

Defined in: OMOStream.h

Author

Tim Bingham - tjb - (Avid Technology, Inc.)

Class Members

Public members.
OMOStream& operator<<(const char* string)
Insert a character string.
OMOStream& operator<<(OMUInt32 i)
Insert an OMUInt32 in decimal.
OMOStream& operator<<(void* p)
Insert a pointer in hex.
OMOStream& endLine(void)
Insert a new line.
OMOStream& operator<<(OMOStream& (*manipulator)(OMOStream&))
Insert (call) a manipulator.

Class Members

Protected members.
OMOStream& put(const char* string)
Put a character string.

OMOStream& put(OMUInt32 i)
Put an OMUInt32 in decimal.
OMOStream& put(void* p)
Put a pointer in hex.
OMOStream& putLine(void)
Put a new line.

OMOStream::endLine

OMOStream& OMOStream::endLine(void)

Put a new line.

Defined in: OMOStream.cpp

Return Value

The modified OMOStream

Back to OMOStream

OMOStream::operator<<

OMOStream& OMOStream::operator<<(void* *p***)**

Insert a pointer in hex.

Defined in: OMOStream.cpp

Return Value

The modified OMOStream

Parameters

p
The pointer to insert.
Back to OMOStream

OMOStream::operator<<

OMOStream& OMOStream::operator<<(const char* *string***)**

Insert a character string.

Defined in: OMOStream.cpp

Return Value

The modified OMOStream

Parameters

string
The string to insert.

Back to OMOStream

OMOStream::operator<<

OMOStream& OMOStream::operator<<(OMOStream& (*manipulator)(OMOStream&)))

Insert (call) a manipulator.

Defined in: OMOStream.cpp

Return Value

The modified OMOStream

Parameters

The manipulator to insert (call).

Back to OMOStream

OMOStream::operator<<

OMOStream& OMOStream::operator<<(OMUInt32 *i*)

Insert an OMUInt32 in decimal.

Defined in: OMOStream.cpp

Return Value

The modified OMOStream

Parameters

i
The OMUInt32 to insert.

Back to OMOStream

OMOStream::put

OMOStream& OMOStream::put(const char* *string***)**

Put a character string.

Defined in: OMOStream.cpp

Return Value

The modified OMOStream

Parameters

string
The character string to be written.

Back to OMOStream

OMOStream::put

OMOStream& OMOStream::put(OMUInt32 *i***)**

Put an OMUInt32 in decimal.

Defined in: OMOStream.cpp

Return Value

The modified OMOStream

Parameters

i
The OMUInt32 to write.

Back to OMOStream

OMOStream::put

OMOStream& OMOStream::put(void* *p***)**

Put a pointer in hex.

Defined in: OMOStream.cpp

Return Value

The modified OMOStream

Parameters

p
The pointer to write.
Back to OMOStream

OMOStream::putLine

OMOStream& OMOStream::putLine(void)

Put a new line.

Defined in: OMOStream.cpp

Return Value

The modified OMOStream

Back to OMOStream

OMProperty class

OMProperty **class OMProperty**

Abstract base class for persistent properties supported by the Object Manager.

Defined in: OMProperty.h

Author

Tim Bingham - tjb - (Avid Technology, Inc.)

Class Members

Public members.
OMProperty**(const OMPropertyId propertyId, const OMStoredForm storedForm, const wchar_t* name)**
Constructor.
void initialize**(const OMPropertyDefinition* definition)**
Temporary pseudo-constructor for clients which provide a property definition.
virtual ~OMProperty**(void)**
Destructor.
virtual void save(void) const
Save this **OMProperty.**
virtual void close**(void)**
Close this **OMProperty.**
virtual void detach**(void)**
Detach this **OMProperty.**
virtual void restore(size_t externalSize)
Restore this **OMProperty**, the external (persisted) size of the **OMProperty** is *externalSize*.
const OMPropertyDefinition* definition**(void) const**

The OMPropertyDefinition defining this **OMProperty.**

const wchar_t* name(void) const

The name of this **OMProperty.**

OMPropertyId propertyId(void) const

The property id of this **OMProperty.**

const OMPropertySet* propertySet(void) const

The OMPropertySet containing this **OMProperty.**

void setPropertySet(const OMPropertySet* propertySet)

Inform this **OMProperty** that it is a member of the OMPropertySet *propertySet*.

OMProperty* address(void)

The address of this **OMProperty** object.

virtual bool isVoid(void) const

Is this **OMProperty** void ?

bool isOptional(void) const

Is this an optional property ?

bool isPresent(void) const

Is this optional property present ?

virtual void removeProperty(void)

Remove this optional **OMProperty.**

virtual size_t bitsSize(void) const

The size of the raw bits of this **OMProperty.** The size is given in bytes.

virtual void getBits(OMByte* bits, size_t size) const

Get the raw bits of this **OMProperty.** The raw bits are copied to the buffer at address *bits* which is *size* bytes in size.

virtual void setBits(const OMByte* bits, size_t size)

Set the raw bits of this **OMProperty.** The raw bits are copied from the buffer at address *bits* which is *size* bytes in size.

virtual OMStorable* storable(void) const

The value of this **OMProperty** as an OMStorable. If this **OMProperty** does not represent an OMStorable then the value returned is 0.

OMStoredForm storedForm(void) const

The stored form of this **OMProperty.**

const OMType* type(void) const

The type of this **OMProperty.**

Class Members

Protected members.

void setPresent(void)

Set the bit that indicates that this optional **OMProperty** is present.

void clearPresent(void)

Clear the bit that indicates that this optional **OMProperty** is present.

OMStorable* container(void) const

The OMStorable that contains this **OMProperty.**

OMStoredObject* store(void) const

The OMStoredObject that contains the persisted representation of this **OMProperty.**

OMFile* file(void) const

The OMFile that contains the persisted representation of this **OMProperty.**

OMProperty::address

OMProperty* OMProperty::address(void)

The address of this OMProperty object. This function is defined so that descendants may override "operator &" to obtain the address of the contained property value. This function can then be used to obtain the address of this OMProperty.

Defined in: OMProperty.cpp

Return Value

The address of this OMProperty.

Back to OMProperty

OMProperty::clearPresent

void OMProperty::clearPresent(void)

Clear the bit that indicates that this optional OMProperty is present.

Defined in: OMProperty.cpp

Back to OMProperty

OMProperty::close

void OMProperty::close(void)

Close this OMProperty.

Defined in: OMProperty.cpp

Back to OMProperty

OMProperty::container

OMStorable* OMProperty::container(void) const

The OMStorable that contains this OMProperty.

Defined in: OMProperty.cpp

Return Value

The containing OMStorable.

Back to OMProperty

OMProperty::definition

const OMPropertyDefinition* OMProperty::definition(void) const

The OMPropertyDefinition defining this OMProperty.

Defined in: OMProperty.cpp

Return Value

The defining OMPropertyDefinition.

Back to OMProperty

OMProperty::detach

void OMProperty::detach(void)

Detach this OMProperty.

Defined in: OMProperty.cpp

Back to OMProperty

OMProperty::file

OMFile* OMProperty::file(void) const

The OMFile that contains the persisted representation of this OMProperty.

Defined in: OMProperty.cpp

Return Value

The containing OMFile.

Back to OMProperty

OMProperty::initialize

void OMProperty::initialize(const OMPropertyDefinition* *definition***)**

Temporary pseudo-constructor for clients which provide a property definition.

Defined in: OMProperty.cpp

Parameters

definition
The definition of this OMProperty.

Back to OMProperty

OMProperty::isOptional

bool OMProperty::isOptional(void) const

Is this an optional property ?

Defined in: OMProperty.cpp

Return Value

True if this property is optional, false otherwise.

Back to OMProperty

OMProperty::isPresent

bool OMProperty::isPresent(void) const

Is this optional property present ?

Defined in: OMProperty.cpp

Return Value

True if this property is present, false otherwise.

Back to OMProperty

OMProperty::isVoid

bool OMProperty::isVoid(void) const

Is this OMProperty void ?

Defined in: OMProperty.cpp

Return Value

True if this OMProperty is void, false otherwise.

Back to OMProperty

OMProperty::name

const wchar_t* OMProperty::name(void) const

The name of this OMProperty.

Defined in: OMProperty.cpp

Return Value

The property name.

Back to OMProperty

OMProperty::OMProperty

OMProperty::OMProperty(const OMPropertyId *propertyId*, **const OMStoredForm** *storedForm*, **const wchar_t*** *name***)**

Constructor.

Defined in: OMProperty.cpp

Parameters

propertyId
: The property id.

storedForm
: The stored form of this OMProperty.

name
: The name of this OMProperty.

Back to OMProperty

OMProperty::propertyId

OMPropertyId OMProperty::propertyId(void) const

The property id of this OMProperty.

Defined in: OMProperty.cpp

Return Value

The property id.

Back to OMProperty

OMProperty::propertySet

const OMPropertySet* OMProperty::propertySet(void) const

The OMPropertySet containing this OMProperty.

Defined in: OMProperty.cpp

Return Value

The containing OMPropertySet.

Back to OMProperty

OMProperty::removeProperty

void OMProperty::removeProperty(void)

Remove this optional OMProperty.

Defined in: OMProperty.cpp

Back to OMProperty

OMProperty::setPresent

void OMProperty::setPresent(void)

Set the bit that indicates that this optional OMProperty is present.

Defined in: OMProperty.cpp

Back to OMProperty

OMProperty::setPropertySet

void OMProperty::setPropertySet(const OMPropertySet* *propertySet***)**

Inform this OMProperty that it is a member of the OMPropertySet *propertySet*.

Defined in: OMProperty.cpp

Parameters

propertySet
The OMPropertySet of which this OMProperty is a member.

Back to OMProperty

OMProperty::storable

OMStorable* OMProperty::storable(void) const

The value of this OMProperty as an OMStorable. If this OMProperty does not represent an OMStorable then the value returned is 0.

Defined in: OMProperty.cpp

Return Value

Always 0.

Back to OMProperty

OMProperty::store

OMStoredObject* OMProperty::store(void) const

The OMStoredObject that contains the persisted representation of this OMProperty.

Defined in: OMProperty.cpp

Return Value

The OMStoredObject.

Back to OMProperty

OMProperty::type

const OMType* OMProperty::type(void) const

The type of this OMProperty.

Defined in: OMProperty.cpp

Return Value

The type.

Back to OMProperty

OMProperty::~OMProperty

OMProperty::~OMProperty(void)

Destructor.

Defined in: OMProperty.cpp

Back to OMProperty

OMPropertyDefinition class

OMPropertyDefinition **class OMPropertyDefinition**

Abstract base class used to define persistent properties supported by the Object Manager.

Defined in: OMPropertyDefinition.h

Author

Tim Bingham - tjb - (Avid Technology, Inc.)

Class Members

Public members.

virtual ~OMPropertyDefinition(void)
: Destructor.

virtual const OMType* type(void) const
: The type of the OMProperty defined by this **OMPropertyDefinition.**

virtual const wchar_t* name(void) const
: The name of the OMProperty defined by this **OMPropertyDefinition.**

virtual OMPropertyId localIdentification(void) const
: The locally unique identification of the OMProperty defined by this **OMPropertyDefinition.**

virtual bool isOptional(void) const

Is the OMProperty defined by this **OMPropertyDefinition** optional?

OMPropertySet class

OMPropertySet **class OMPropertySet**

Container class for OMProperty objects.

Defined in: OMPropertySet.h

Author

Tim Bingham - tjb - (Avid Technology, Inc.)

Class Members

Public members.

OMPropertySet(void)

Constructor.

~OMPropertySet(void)

Destructor.

OMProperty* get(const OMPropertyId propertyId) const

Get the OMProperty associated with the property id *propertyId*.

OMProperty* get(const wchar_t* propertyName) const

Get the OMProperty named *propertyName*.

void put(OMProperty* property)

Insert the OMProperty *property* into this **OMPropertySet.**

bool isPresent(const OMPropertyId propertyId) const

Is an OMProperty with property id *propertyId* present in this **OMPropertySet** ?

bool isPresent(const wchar_t* propertyName) const

Is an OMProperty with name *propertyName* present in this **OMPropertySet** ?

bool isAllowed(const OMPropertyId propertyId) const

Is an OMProperty with property id *propertyId* allowed in this **OMPropertySet** ?

bool isRequired(const OMPropertyId propertyId) const

Is an OMProperty with property id *propertyId* a required member of this **OMPropertySet** ?

size_t count(void) const

The number of OMProperty objects in this **OMPropertySet.**

void setContainer(const OMStorable* container)

This **OMPropertySet** is contained by the given OMStorable object *container*. The OMProperty objects in this **OMPropertySet** are the properties of the given OMStorable object *container*.

OMStorable* container(void) const

The OMStorable object that contains this **OMPropertySet.**

OMPropertySet::container

OMStorable* OMPropertySet::container(void) const

The OMStorable object that contains this OMPropertySet.

Defined in: OMPropertySet.cpp

Return Value

The OMStorable object that contains this OMPropertySet.

Back to OMPropertySet

OMPropertySet::count

size_t OMPropertySet::count(void) const

The number of OMProperty objects in this OMPropertySet.

Defined in: OMPropertySet.cpp

Return Value

The number of OMProperty objects in this OMPropertySet.

Back to OMPropertySet

OMPropertySet::get

OMProperty* OMPropertySet::get(const OMPropertyId *propertyId***) const**

Get the OMProperty associated with the property id *propertyId*.

Defined in: OMPropertySet.cpp

Return Value

The OMProperty object with property id *propertyId*.

Parameters

propertyId
Property id.

Back to OMPropertySet

OMPropertySet::get

OMProperty* OMPropertySet::get(const wchar_t* *propertyName***) const**

Get the OMProperty named *propertyName*.

Defined in: OMPropertySet.cpp

Return Value

The OMProperty with name *propertyName*.

Parameters

propertyName
Property name.

Back to OMPropertySet

OMPropertySet::isAllowed

bool OMPropertySet::isAllowed(const OMPropertyId *propertyId***) const**

Is an OMProperty with property id *propertyId* allowed in this OMPropertySet ?

Defined in: OMPropertySet.cpp

Return Value

true if an OMProperty with property id *propertyId* is allowed **false** otherwise.

Parameters

propertyId
Property id.

Back to OMPropertySet

OMPropertySet::isPresent

bool OMPropertySet::isPresent(const OMPropertyId *propertyId***) const**

Is an OMProperty with property id *propertyId* present in this OMPropertySet ?

Defined in: OMPropertySet.cpp

Return Value

true if an OMProperty with property id *propertyId* is present **false** otherwise.

Parameters

propertyId

Property id.

Back to OMPropertySet

OMPropertySet::isPresent

bool OMPropertySet::isPresent(const wchar_t* *propertyName***) const**

Is an OMProperty with name *propertyName* present in this OMPropertySet ?

Defined in: OMPropertySet.cpp

Return Value

true if an OMProperty with name *propertyName* is present **false** otherwise.

Parameters

propertyName
Property name.

Back to OMPropertySet

OMPropertySet::isRequired

bool OMPropertySet::isRequired(const OMPropertyId *propertyId***) const**

Is an OMProperty with property id *propertyId* a required member of this OMPropertySet ?

Defined in: OMPropertySet.cpp

Return Value

true if an OMProperty with property id *propertyId* is required **false** otherwise.

Parameters

propertyId
Property id.

Back to OMPropertySet

OMPropertySet::put

void OMPropertySet::put(OMProperty* *property***)**

Insert the OMProperty *property* into this OMPropertySet.

Defined in: OMPropertySet.cpp

Parameters

property
OMProperty to insert.

Back to OMPropertySet

OMPropertySet::setContainer

void OMPropertySet::setContainer(const OMStorable* *container***)**

This OMPropertySet is contained by the given OMStorable object *container*. The OMProperty objects in this OMPropertySet are the properties of the given OMStorable object *container*.

Defined in: OMPropertySet.cpp

Parameters

container
The OMStorable object that contains this OMPropertySet.

Back to OMPropertySet

OMPropertySetIterator class

OMPropertySetIterator **class OMPropertySetIterator**

Iterators over OMPropertySets.

Defined in: OMPropertySetIterator.h

Author

Tim Bingham - tjb - (Avid Technology, Inc.)

Class Members

Public members.

OMPropertySetIterator**(const OMPropertySet& set, OMIteratorPosition initialPosition)**
Create an **OMPropertySetIterator** over the given OMPropertySet *set* and initialize it to the given *initialPosition*. If *initialPosition* is specified as OMBefore then this **OMPropertySetIterator** is made ready to traverse the associated OMPropertySet in the forward direction (increasing *OMPropertyIds*). If *initialPosition* is specified as OMAfter then this **OMPropertySetIterator** is made ready to traverse the associated OMPropertySet in the reverse direction (decreasing *OMPropertyIds*).

virtual ~OMPropertySetIterator**(void)**
Destroy this **OMPropertySetIterator**.

virtual void reset**(OMIteratorPosition initialPosition)**

Reset this **OMPropertySetIterator** to the given *initialPosition*. If *initialPosition* is specified as OMBefore then this **OMPropertySetIterator** is made ready to traverse the associated OMPropertySet in the forward direction (increasing *OMPropertyId*s). If *initialPosition* is specified as OMAfter then this **OMPropertySetIterator** is made ready to traverse the associated OMPropertySet in the reverse direction (decreasing *OMPropertyId*s).

virtual bool before**(void) const**

Is this **OMPropertySetIterator** positioned before the first OMProperty ?

virtual bool after**(void) const**

Is this **OMPropertySetIterator** positioned after the last OMProperty ?

virtual bool valid**(void) const**

Is this **OMPropertySetIterator** validly positioned on an OMProperty ?

virtual size_t count**(void) const**

The number of OMPropertys in the associated OMPropertySet.

virtual bool operator++**()**

Advance this **OMPropertySetIterator** to the next OMProperty, if any. If the end of the associated OMPropertySet is not reached then the result is **true** , **valid** becomes **true** and **after** becomes **false** . If the end of the associated OMPropertySet is reached then the result is **false** , **valid** becomes **false** and **after** becomes **true** .

virtual bool operator--**()**

Retreat this **OMPropertySetIterator** to the previous OMProperty, if any. If the beginning of the associated OMPropertySet is not reached then the result is **true** , **valid** becomes **true** and **before** becomes **false** . If the beginning of the associated OMPropertySet is reached then the result is **false** , **valid** becomes **false** and **before** becomes **true** .

virtual OMProperty* property**(void) const**

Return the OMProperty in the associated OMPropertySet at the position currently designated by this **OMPropertySetIterator**.

OMPropertyId propertyId**(void) const**

Return the *OMPropertyId* of the OMProperty in the associated OMPropertySet at the position currently designated by this **OMPropertySetIterator**.

OMPropertySetIterator::after

bool OMPropertySetIterator::after(void) const

Is this OMPropertySetIterator positioned after the last OMProperty ?

Defined in: OMPropertySetIterator.cpp

Return Value

true if this OMPropertySetIterator is positioned after the last OMProperty, **false** otherwise.

Back to OMPropertySetIterator

OMPropertySetIterator::before

bool OMPropertySetIterator::before(void) const

Is this OMPropertySetIterator positioned before the first OMProperty ?

Defined in: OMPropertySetIterator.cpp

Return Value

true if this OMPropertySetIterator is positioned before the first OMProperty, **false** otherwise.

Back to OMPropertySetIterator

OMPropertySetIterator::count

size_t OMPropertySetIterator::count(void) const

The number of OMPropertys in the associated OMPropertySet.

Defined in: OMPropertySetIterator.cpp

Return Value

The number of OMPropertys

Back to OMPropertySetIterator

OMPropertySetIterator::OMPropertySetIterator

OMPropertySetIterator::OMPropertySetIterator(const OMPropertySet& *set*, **OMIteratorPosition** *initialPosition***)**

Create an OMPropertySetIterator over the given OMPropertySet *set* and initialize it to the given *initialPosition*. If *initialPosition* is specified as OMBefore then this OMPropertySetIterator is made ready to traverse the associated OMPropertySet in the forward direction (increasing *OMPropertyId*s). If *initialPosition* is specified as OMAfter then this OMPropertySetIterator is made ready to traverse the associated OMPropertySet in the reverse direction (decreasing *OMPropertyId*s).

Defined in: OMPropertySetIterator.cpp

Parameters

set
: The OMPropertySet over which to iterate.

initialPosition
: The initial position for this OMPropertySetIterator.

Back to OMPropertySetIterator

OMPropertySetIterator::operator++

bool OMPropertySetIterator::operator++(void)

Advance this OMPropertySetIterator to the next OMProperty, if any. If the end of the associated OMPropertySet is not reached then the result is **true** , **valid** becomes **true** and **after** becomes **false** . If the end of the associated OMPropertySet is reached then the result is **false** , **valid** becomes **false** and **after** becomes **true** .

Defined in: OMPropertySetIterator.cpp

Return Value

false if this OMPropertySetIterator has passed the last OMProperty, **true** otherwise.

Back to OMPropertySetIterator

OMPropertySetIterator::operator--

bool OMPropertySetIterator::operator--(void)

Retreat this OMPropertySetIterator to the previous OMProperty, if any. If the beginning of the associated OMPropertySet is not reached then the result is **true** , **valid** becomes **true** and **before** becomes **false** . If the beginning of the associated OMPropertySet is reached then the result is **false** , **valid** becomes **false** and **before** becomes **true** .

Defined in: OMPropertySetIterator.cpp

Return Value

false if this OMPropertySetIterator has passed the first OMProperty, **true** otherwise.

Back to OMPropertySetIterator

OMPropertySetIterator::property

OMProperty* OMPropertySetIterator::property(void) const

Return the OMProperty in the associated OMPropertySet at the position currently designated by this OMPropertySetIterator.

Defined in: OMPropertySetIterator.cpp

Return Value

The OMProperty at the current position.

Back to OMPropertySetIterator

OMPropertySetIterator::propertyId

OMPropertyId OMPropertySetIterator::propertyId(void) const

Return the *OMPropertyId* of the OMProperty in the associated OMPropertySet at the position currently designated by this OMPropertySetIterator.

Defined in: OMPropertySetIterator.cpp

Return Value

The *OMPropertyId* at the current position.

Back to OMPropertySetIterator

OMPropertySetIterator::reset

void OMPropertySetIterator::reset(OMIteratorPosition *initialPosition***)**

Reset this OMPropertySetIterator to the given *initialPosition*. If *initialPosition* is specified as OMBefore then this OMPropertySetIterator is made ready to traverse the associated OMPropertySet in the forward direction (increasing *OMPropertyId*s). If *initialPosition* is specified as OMAfter then this OMPropertySetIterator is made ready to traverse the associated OMPropertySet in the reverse direction (decreasing *OMPropertyId*s).

Defined in: OMPropertySetIterator.cpp

Parameters

initialPosition
The position to which this OMPropertySetIterator should be reset.

Back to OMPropertySetIterator

OMPropertySetIterator::valid

bool OMPropertySetIterator::valid(void) const

Is this OMPropertySetIterator validly positioned on an OMProperty ?

Defined in: OMPropertySetIterator.cpp

Return Value

true if this OMPropertySetIterator is validly positioned on an OMProperty, **false** otherwise.

Back to OMPropertySetIterator

OMPropertySetIterator::~OMPropertySetIterator

OMPropertySetIterator::~OMPropertySetIterator(void)

Destroy this OMPropertySetIterator.

Defined in: OMPropertySetIterator.cpp

Back to OMPropertySetIterator

OMPropertyTable class

OMPropertyTable **class OMPropertyTable**

Persistent tables of property instance names.

Defined in: OMPropertyTable.h

Author

Tim Bingham - tjb - (Avid Technology, Inc.)

Class Members

Public members.

OMPropertyTable**(void)**
: Constructor.

~OMPropertyTable**(void)**
: Destructor.

OMPropertyTag insert**(const OMPropertyId* propertyPath)**
: If *propertyPath* is not already present then insert it (by copying) into the table and return its tag, otherwise just return its tag. Tags are allocated sequentially.

const OMPropertyId* valueAt**(OMPropertyTag tag) const**
: The property path corresponding to *tag* in the table.

size_t count**(void) const**
: The count of entries in the table.

bool isValid**(OMPropertyTag tag) const**
: Is *tag* valid ?

OMPropertyTable::count

size_t OMPropertyTable::count(void) const

The count of entries in the table.

Defined in: OMPropertyTable.cpp

Return Value

The count of entries.

Back to OMPropertyTable

OMPropertyTable::insert

OMPropertyTag OMPropertyTable::insert(const OMPropertyId* *propertyPath***)**

If *propertyPath* is not already present then insert it (by copying) into the table and return its tag, otherwise just return its tag. Tags are allocated sequentially.

Defined in: OMPropertyTable.cpp

Return Value

The assigned index.

Parameters

propertyPath
The property path to insert.

Back to OMPropertyTable

OMPropertyTable::isValid

bool OMPropertyTable::isValid(OMPropertyTag *tag***) const**

Is *tag* valid ?

Defined in: OMPropertyTable.cpp

Return Value

True if the tag is valid, false otherwise.

Parameters

tag
The tag to check.

Back to OMPropertyTable

OMPropertyTable::OMPropertyTable

OMPropertyTable::OMPropertyTable(void)

Constructor.

Defined in: OMPropertyTable.cpp

Back to OMPropertyTable

OMPropertyTable::valueAt

const OMPropertyId* OMPropertyTable::valueAt(OMPropertyTag *tag*) const

The property path corresponding to *tag* in the table.

Defined in: OMPropertyTable.cpp

Return Value

The property path.

Parameters

tag
The index.

Back to OMPropertyTable

OMPropertyTable::~OMPropertyTable

OMPropertyTable::~OMPropertyTable(void)

Destructor.

Defined in: OMPropertyTable.cpp

Back to OMPropertyTable

OMRawStorage class

OMRawStorage **class OMRawStorage**

Abstract base class supporting access to the raw bytes of files supported by the Object Manager.

Object Manager clients use this interface, after a file has been saved and closed, to access the raw bytes of a file, or, before a file has been opened, to supply the raw bytes to be used for that file.

Object Manager clients implement this interface to allow files to be stored in locations not known to the Object Manager.

For example, Object Manager clients may wish to store files in a proprietary file system, this could be done by providing an implementation of this interface for that file system.

Additionally a number of built-in implementations of this interface are provided -

OMDiskRawStorage - an implementation of **OMRawStorage** for disk files. Uses ANSI file functions only.

OMMemoryRawStorage - an implementation of **OMRawStorage** that stores the file in memory.

OMMappedFileRawStorage - an implementation of **OMRawStorage** for files mapped into memory.

Defined in: OMRawStorage.h

Author

Tim Bingham - tjb - (Avid Technology, Inc.)

Class Members

Public members.

virtual ~OMRawStorage(void)

Destructor.

virtual bool isReadable(void) const

Is it possible to read from this **OMRawStorage** ?

virtual void read(OMByte* bytes, OMUInt32 byteCount, OMUInt32& bytesRead) const

Attempt to read the number of bytes given by *byteCount* from the current position in this **OMRawStorage** into the buffer at address *bytes*. The actual number of bytes read is returned in *bytesRead*. Reading from positions greater than **size** causes *bytesRead* to be less than *byteCount*. Reading bytes that have never been written returns undefined data in *bytes*.

preconditions

isReadable()

virtual void readAt(OMUInt64 position, OMByte* bytes, OMUInt32 byteCount, OMUInt32& bytesRead) const

Attempt to read the number of bytes given by *byteCount* from offset *position* in this **OMRawStorage** into the buffer at address *bytes*. The actual number of bytes read is returned in *bytesRead*. Reading from positions greater than **size** causes *bytesRead* to be less than *byteCount*. Reading bytes that have never been written returns undefined data in *bytes*.

preconditions

isReadable() && **isPositionable()**

virtual bool isWritable(void) const

Is it possible to write to this **OMRawStorage** ?

virtual void write(const OMByte* bytes, OMUInt32 byteCount, OMUInt32& bytesWritten)

Attempt to write the number of bytes given by *byteCount* to the current position in this **OMRawStorage** from the buffer at address *bytes*. The actual number of bytes written is returned in *bytesWritten*. Writing to positions greater than **size** causes this **OMRawStorage** to be extended, however such extension can fail, causing *bytesWritten* to be less than *byteCount*.

preconditions

isWritable()

Developer Notes

How is failure to extend indicated ?

virtual void writeAt(OMUInt64 position, const OMByte* bytes, OMUInt32 byteCount, OMUInt32& bytesWritten)

Attempt to write the number of bytes given by *byteCount* to offset *position* in this **OMRawStorage** from the buffer at address *bytes*. The actual number of bytes written is returned in *bytesWritten*. Writing to positions greater than **size** causes this **OMRawStorage** to be extended, however such extension can fail, causing *bytesWritten* to be less than *byteCount*.

preconditions

isWritable() && **isPositionable()**

Developer Notes

How is failure to extend indicated ?

virtual bool isExtendible(void) const

May this **OMRawStorage** be changed in size ? An implementation of **OMRawStorage** for disk files would most probably return true. An implemetation for network streams would return false. An implementation for fixed size contiguous memory files (avoiding copying) would return false.

virtual OMUInt64 extent(void) const

The current extent of this **OMRawStorage** in bytes. The **extent()** is the allocated size, while the **size()** is the valid size.

preconditions

isPositionable()

virtual void extend(OMUInt64 newSize)

Set the size of this **OMRawStorage** to *newSize* bytes. If *newSize* is greater than **size** then this **OMRawStorage** is extended. If *newSize* is less than **size** then this **OMRawStorage** is truncated. Truncation may also result in the current position for **read()** and **write()** being set to **size**.

preconditions

isExtendible()

Developer Notes

How is failure to extend indicated ?

virtual OMUInt64 size(void) const

The current size of this **OMRawStorage** in bytes. The **size()** is the valid size, while the **extent()** is the allocated size.

preconditions

isPositionable()

virtual bool isPositionable(void) const

May the current position, for **read()** and **write()**, of this **OMRawStorage** be changed ? An implementation of **OMRawStorage** for disk files would most probably return true. An implemetation for network streams would return false. An implementation for memory files would return true.

virtual void synchronize(void)

Synchronize this **OMRawStorage** with its external representation. An implementation of **OMRawStorage** for disk files would most probably implement this virtual function as a flush. This virtual function would probably be implemented as a noop in implemetations for network streams and for memory files.

OMRawStorageLockBytes class

OMRawStorageLockBytes **class OMRawStorageLockBytes: public ILockBytes**

An implementation of the Microsoft Structured Storage interface ILockBytes in terms of OMRawStorage. This class is an adapter from the ILockBytes interface to the exported OMRawStorage interface. Object Manager clients may use the OMRawStorage interface to access or to control the storage of the raw bytes of a file.

Defined in: OMRawStorageLockBytes.h

Author

Tim Bingham - tjb - (Avid Technology, Inc.)

Class Members

Public members.

OMRawStorageLockBytes**(OMRawStorage* rawStorage)**

Constructor.

virtual ~OMRawStorageLockBytes**(void)**

Destructor.

virtual HRESULT STDMETHODCALLTYPE ReadAt**(ULARGE_INTEGER ulOffset, void *pv, ULONG cb, ULONG *pcbRead)**

Read bytes (see Microsoft documentation for details).

virtual HRESULT STDMETHODCALLTYPE WriteAt**(ULARGE_INTEGER ulOffset, const void *pv, ULONG cb, ULONG *pcbWritten)**

Write bytes (see Microsoft documentation for details).

virtual HRESULT STDMETHODCALLTYPE Flush**(void)**

Flush any buffered bytes (see Microsoft documentation for details).

virtual HRESULT STDMETHODCALLTYPE SetSize**(ULARGE_INTEGER cb)**

Set the size, either grow or shrink (see Microsoft documentation for details).

virtual HRESULT STDMETHODCALLTYPE LockRegion**(ULARGE_INTEGER libOffset, ULARGE_INTEGER cb, DWORD dwLockType)**

See Microsoft documentation for details.

virtual HRESULT STDMETHODCALLTYPE UnlockRegion**(ULARGE_INTEGER libOffset, ULARGE_INTEGER cb, DWORD dwLockType)**

See Microsoft documentation for details.

virtual HRESULT STDMETHODCALLTYPE Stat**(STATSTG *pstatstg, DWORD grfStatFlag)**

See Microsoft documentation for details.

Class Members

Private members.

OMRawStorageLockBytes::Flush

HRESULT STDMETHODCALLTYPE OMRawStorageLockBytes::Flush(void)

Flush any buffered bytes (see Microsoft documentation for details).

Defined in: OMRawStorageLockBytes.cpp

Back to OMRawStorageLockBytes

OMRawStorageLockBytes::LockRegion

HRESULT STDMETHODCALLTYPE OMRawStorageLockBytes::LockRegion(void)

See Microsoft documentation for details.

Defined in: OMRawStorageLockBytes.cpp

Back to OMRawStorageLockBytes

OMRawStorageLockBytes::OMRawStorageLockBytes

OMRawStorageLockBytes::OMRawStorageLockBytes(void)

Constructor.

Defined in: OMRawStorageLockBytes.cpp

Back to OMRawStorageLockBytes

OMRawStorageLockBytes::ReadAt

HRESULT STDMETHODCALLTYPE OMRawStorageLockBytes::ReadAt(void)

Read bytes (see Microsoft documentation for details).

Defined in: OMRawStorageLockBytes.cpp

Back to OMRawStorageLockBytes

OMRawStorageLockBytes::SetSize

HRESULT STDMETHODCALLTYPE OMRawStorageLockBytes::SetSize(void)

Set the size, either grow or shrink (see Microsoft documentation for details).

Defined in: OMRawStorageLockBytes.cpp

Back to OMRawStorageLockBytes

OMRawStorageLockBytes::Stat

HRESULT STDMETHODCALLTYPE OMRawStorageLockBytes::Stat(void)

See Microsoft documentation for details.

Defined in: OMRawStorageLockBytes.cpp

Back to OMRawStorageLockBytes

OMRawStorageLockBytes::UnlockRegion

HRESULT STDMETHODCALLTYPE OMRawStorageLockBytes::UnlockRegion(void)

See Microsoft documentation for details.

Defined in: OMRawStorageLockBytes.cpp

Back to OMRawStorageLockBytes

OMRawStorageLockBytes::WriteAt

HRESULT STDMETHODCALLTYPE OMRawStorageLockBytes::WriteAt(void)

Write bytes (see Microsoft documentation for details).

Defined in: OMRawStorageLockBytes.cpp

Back to OMRawStorageLockBytes

OMRawStorageLockBytes::~OMRawStorageLockBytes

OMRawStorageLockBytes::~OMRawStorageLockBytes(void)

Destructor.

Defined in: OMRawStorageLockBytes.cpp

Back to OMRawStorageLockBytes

OMRedBlackTree class

OMRedBlackTree **class OMRedBlackTree**

Red-black trees. A red-black tree is an approximately balanced binary search tree providing O(lg N) performance for the dynamic set operations. Items in the tree are uniquely identified by key and carry an associated value.

Defined in: OMRedBlackTree.h

Class Template Arguments

Key
: The type of the unique key used to identify elements. This type must support operator =, operator != and operator

Value
: The type of the value carried in an **OMRedBlackTree** item. This type must support operator =.

Author

Tim Bingham - tjb - (Avid Technology, Inc.)

Class Members

Public members.

OMRedBlackTree(void)
: Constructor.

virtual ~OMRedBlackTree(void)
: Destructor.

bool insert**(const Key k, Value v)**
: Insert the *Value v* into this **OMRedBlackTree** and associate it with *Key k*. If this the first instance of an item identified by *Key k* in this **OMRedBlackTree**, the result is true, otherwise the result is false.

bool find**(const Key k, Value& v) const**
: Find the item in this **OMRedBlackTree** identified *k*. If the item is found it is returned in *v* and the result is true. If the element is not found the result is false.

bool find**(const Key k, Value** v) const**
: Find the item in this **OMRedBlackTree** identified *k*. If the item is found it is returned in *v* and the result is true. If the element is not found the result is false.

bool contains**(const Key k) const**
: Does this **OMRdBlackTree** contain an item identified by *k*?

bool remove**(const Key k)**

Remove the item assciated with *Key* k from this **OMRedBlackTree.**

virtual void clear**(void)**

Remove all items from this **OMRedBlackTree.**

size_t count**(void) const**

The number of items in this **OMRedBlackTree. count** returns the actual number of items in the **OMRedBlackTree.**

void traverseInOrder**(void (*f)(size_t height, Key k, const Value& v)) const**

Traverse this **OMRedBlackTree** in order, the function *f* is called for each item in the tree.

void traverseInPreOrder**(void (*f)(size_t height, Key k, const Value& v)) const**

Traverse this **OMRedBlackTree** in pre-order, the function *f* is called for each item in the tree.

void traverseInPostOrder**(void (*f)(size_t height, Key k, const Value& v)) const**

Traverse this **OMRedBlackTree** in post-order, the function *f* is called for each item in the tree.

size_t height**(void) const**

The height of this **OMRedBlackTree.**

OMRedBlackTree::clear

template <class *Key*, **class** *Value***>**
void OMRedBlackTree<*Key*, *Value***>::clear(void)**

Remove all items from this OMRedBlackTree.

Defined in: OMRedBlackTreeT.h

Class Template Arguments

Key

The type of the unique key used to identify elements. This type must support operator =, operator != and operator

Value

The type of the value carried in an OMRedBlackTree item. This type must support operator =.

Back to OMRedBlackTree

OMRedBlackTree::contains

template <class *Key*, **class** *Value***>**
bool OMRedBlackTree<*Key*, *Value***>::contains(const Key** *k***) const**

Does this **OMRdBlackTree** contain an item identified by *k*?

Defined in: OMRedBlackTreeT.h

Return Value

True if *k* was found in this OMRedBlackTree, false otherwise.

Parameters

k
: The *Key* for which to search.

Class Template Arguments

Key
: The type of the unique key used to identify elements. This type must support operator =, operator != and operator

Value
: The type of the value carried in an OMRedBlackTree item. This type must support operator =.

Back to OMRedBlackTree

OMRedBlackTree::count

template <class *Key*, **class** *Value***>**
size_t OMRedBlackTree<*Key*, *Value***>::count(void) const**

The number of items in this OMRedBlackTree. **count** returns the actual number of items in the OMRedBlackTree.

Defined in: OMRedBlackTreeT.h

Return Value

The number of items in the OMRedBlackTree.

Class Template Arguments

Key
: The type of the unique key used to identify elements. This type must support operator =, operator != and operator

Value
: The type of the value carried in an OMRedBlackTree item. This type must support operator =.

Back to OMRedBlackTree

OMRedBlackTree::find

template <class *Key*, **class** *Value***>**
bool OMRedBlackTree<*Key*, *Value***>::find(const Key** *k*, **Value**** *v***) const**

Find the item in this OMRedBlackTree identified by *k*. If the item is found it is returned in *v* and the result is true. If the element is not found the result is false.

Defined in: OMRedBlackTreeT.h

Return Value

True if *k* was found in this OMRedBlackTree, false otherwise.

Parameters

k
: The *Key* for which to search.

v
: The *Value* associated with *k*, if any, by pointer.

Class Template Arguments

Key
: The type of the unique key used to identify elements. This type must support operator =, operator != and operator

Value
: The type of the value carried in an OMRedBlackTree item. This type must support operator =.

Back to OMRedBlackTree

OMRedBlackTree::find

template <class *Key*, **class** *Value***>**
bool OMRedBlackTree<*Key*, *Value***>::find(const Key** *k*, **Value&** *v***) const**

Find the item in this OMRedBlackTree identified by *k*. If the item is found it is returned in *v* and the result is true. If the element is not found the result is false.

Defined in: OMRedBlackTreeT.h

Return Value

True if *k* was found in this OMRedBlackTree, false otherwise.

Parameters

k
: The *Key* for which to search.

v
: The *Value* associated with *k*, if any, by reference.

Class Template Arguments

Key
: The type of the unique key used to identify elements. This type must support operator =, operator != and operator

Value
: The type of the value carried in an OMRedBlackTree item. This type must support operator =.

Back to OMRedBlackTree

OMRedBlackTree::height

template <class *Key*, **class** *Value***>**
size_t OMRedBlackTree<*Key*, *Value***>::height(** *void***) const**

The height of this OMRedBlackTree.

Defined in: OMRedBlackTreeT.h

Parameters

void
: The hiight of this OMRedBlackTree.

Class Template Arguments

Key
: The type of the unique key used to identify elements. This type must support operator =, operator != and operator

Value
: The type of the value carried in an OMRedBlackTree item. This type must support operator =.

Back to OMRedBlackTree

OMRedBlackTree::insert

template <class *Key*, **class** *Value***>**
bool OMRedBlackTree<*Key*, *Value***>::insert(const Key** *k***, Value** *v***)**

Insert the *Value v* into this OMRedBlackTree and associate it with *Key k*. If this the first instance of an item identified by *Key k* in this OMRedBlackTree, the result is true, otherwise the result is false.

Defined in: OMRedBlackTreeT.h

Return Value

True if this is the first instance of an item identified by *Key k* in this OMRedBlackTree, false otherwise.

Parameters

k
: The key.

v
: The value.

Class Template Arguments

Key
: The type of the unique key used to identify elements. This type must support operator =, operator != and operator

Value
: The type of the value carried in an OMRedBlackTree item. This type must support operator =.

Back to OMRedBlackTree

OMRedBlackTree::remove

template <class *Key*, **class** *Value***>**
bool OMRedBlackTree<*Key*, *Value***>::remove(const Key** *k***)**

Remove the item assciated with *Key k* from this OMRedBlackTree.

Defined in: OMRedBlackTreeT.h

Return Value

True if an item identified by *Key k* was found, false otherwise.

Parameters

k
: The *Key* of the item to remove.

Class Template Arguments

Key
: The type of the unique key used to identify elements. This type must support operator =, operator != and operator

Value
: The type of the value carried in an OMRedBlackTree item. This type must support operator =.

Back to OMRedBlackTree

OMRedBlackTree::traverseInOrder

template <class *Key*, **class** *Value***>**
void OMRedBlackTree<*Key*, *Value***>::traverseInOrder(** *void (*f)***)**

Traverse this OMRedBlackTree in order, the function *f* is called for each item.

Defined in: OMRedBlackTreeT.h

Parameters

*void (*f)*
: The function to apply to each item.

Class Template Arguments

Key
: The type of the unique key used to identify elements. This type must support operator =, operator != and operator

Value
: The type of the value carried in an OMRedBlackTree item. This type must support operator =.

Back to OMRedBlackTree

OMRedBlackTree::traverseInPostOrder

template <class *Key*, **class** *Value***>**
void OMRedBlackTree<*Key*, *Value***>::traverseInPostOrder(** *void (*f)***) const**

Traverse this OMRedBlackTree in post-order, the function *f* is called for each item.

Defined in: OMRedBlackTreeT.h

Parameters

*void (*f)*
: The function to apply to each item.

Class Template Arguments

Key
: The type of the unique key used to identify elements. This type must support operator =, operator != and operator

Value
: The type of the value carried in an OMRedBlackTree item. This type must support operator =.

Back to OMRedBlackTree

OMRedBlackTree::traverseInPreOrder

template <class *Key*, **class** *Value***>**
void OMRedBlackTree<*Key*, *Value***>::traverseInPreOrder(** *void (*f)***) const**

Traverse this OMRedBlackTree in pre-order, the function *f* is called for each item.

Defined in: OMRedBlackTreeT.h

Parameters

*void (*f)*
: The function to apply to each item.

Class Template Arguments

Key
: The type of the unique key used to identify elements. This type must support operator =, operator != and operator

Value
: The type of the value carried in an OMRedBlackTree item. This type must support operator =.

Back to OMRedBlackTree

OMRedBlackTreeIterator class

OMRedBlackTreeIterator **class OMRedBlackTreeIterator: public OMContainerIterator**

Iterators over OMRedBlackTrees.

Defined in: OMRedBlackTreeIterator.h

Class Template Arguments

Key
: The type of the unique key that identifies the contained values.

Value
: The type of the contained values.

Author

Tim Bingham - tjb - (Avid Technology, Inc.)

Class Members

Public members.

OMRedBlackTreeIterator**(const OMRedBlackTree&ltKey, Value>& redBlackTree, OMIteratorPosition initialPosition)**

Create an **OMRedBlackTreeIterator** over the given OMRedBlackTree *redBlackTree* and initialize it to the given *initialPosition*. If *initialPosition* is specified as OMBefore then this **OMRedBlackTreeIterator** is made ready to traverse the associated OMRedBlackTree in the forward direction (increasing *Key*s). If *initialPosition* is specified as OMAfter then this **OMRedBlackTreeIterator** is made ready to traverse the associated OMRedBlackTree in the reverse direction (decreasing *Key*s).

virtual ~OMRedBlackTreeIterator**(void)**

Destroy this **OMRedBlackTreeIterator.**

virtual void reset**(OMIteratorPosition initialPosition)**

Reset this **OMRedBlackTreeIterator** to the given *initialPosition*. If *initialPosition* is specified as OMBefore then this **OMRedBlackTreeIterator** is made ready to traverse the associated OMRedBlackTree in the forward direction (increasing *Key*s). If *initialPosition* is specified as OMAfter then this **OMRedBlackTreeIterator** is made ready to traverse the associated OMRedBlackTree in the reverse direction (decreasing *Key*s).

virtual bool before**(void) const**

Is this **OMRedBlackTreeIterator** positioned before the first *Value* ?

virtual bool after**(void) const**

Is this **OMRedBlackTreeIterator** positioned after the last *Value* ?

virtual size_t count**(void) const**

The number of *Value*s in the associated OMRedBlackTree.

virtual bool operator++**()**

Advance this **OMRedBlackTreeIterator** to the next *Value*, if any. If the end of the associated OMRedBlackTree is not reached then the result is **true** , **valid** becomes **true** and **after** becomes **false** . If the end of the associated OMRedBlackTree is reached then the result is **false** , **valid** becomes **false** and **after** becomes **true** .

virtual bool operator--**()**

Retreat this **OMRedBlackTreeIterator** to the previous *Value*, if any. If the beginning of the associated OMRedBlackTree is not reached then the result is **true** , **valid** becomes **true** and **before** becomes **false** . If the beginning of the associated OMRedBlackTree is reached then the result is **false** , **valid** becomes **false** and **before** becomes **true** .

virtual Value& value**(void) const**

Return the *Value* in the associated OMRedBlackTree at the position currently designated by this **OMRedBlackTreeIterator.**

virtual Value setValue**(const Key k, Value newValue)**

Set the *Value* in the associated OMRedBlackTree at the position currently designated by this **OMRedBlackTreeIterator** to *newValue*. The previous *Value* is returned. To preserve the ordering of *Key*s, the *Key* of *newValue* must be the same as that of the existing *Value*.

Key key**(void) const**

Return the *Key* of the *Value* in the associated OMRedBlackTree at the position currently designated by this **OMRedBlackTreeIterator.**

OMRedBlackTreeIterator::after

template <class *Key*, **class** *Value***>**
bool OMRedBlackTreeIterator<*Key*, *Value***>::after(void) const**

Is this OMRedBlackTreeIterator positioned after the last *Value* ?

Defined in: OMRedBlackTreeIteratorT.h

Return Value

true if this OMRedBlackTreeIterator is positioned after the last *Value*, **false** otherwise.

Class Template Arguments

Key
The type of the unique key that identifies the contained values.
Value
The type of the contained values.
Back to OMRedBlackTreeIterator

OMRedBlackTreeIterator::before

template <class *Key*, **class** *Value***>**
bool OMRedBlackTreeIterator<*Key*, *Value***>::before(void) const**

Is this OMRedBlackTreeIterator positioned before the first *Value* ?

Defined in: OMRedBlackTreeIteratorT.h

Return Value

true if this OMRedBlackTreeIterator is positioned before the first *Value*, **false** otherwise.

Class Template Arguments

Key
The type of the unique key that identifies the contained values.
Value
The type of the contained values.
Back to OMRedBlackTreeIterator

OMRedBlackTreeIterator::count

template <class *Key*, **class** *Value***>**
size_t OMRedBlackTreeIterator<*Key*, *Value***>::count(void) const**

The number of *Value*s in the associated OMRedBlackTree.

Defined in: OMRedBlackTreeIteratorT.h

Return Value

The number of *Value*s.

Class Template Arguments

Key
: The type of the unique key that identifies the contained values.

Value
: The type of the contained values.

Back to OMRedBlackTreeIterator

OMRedBlackTreeIterator::key

template <class *Key*, **class** *Value***>**
Key OMRedBlackTreeIterator<*Key*, *Value***>::key(void) const**

Return the *Key* of the *Value* in the associated OMRedBlackTree at the position currently designated by this OMRedBlackTreeIterator.

Defined in: OMRedBlackTreeIteratorT.h

Return Value

The *Key* at the current position.

Class Template Arguments

Key
: The type of the unique key that identifies the contained values.

Value
: The type of the contained values.

Back to OMRedBlackTreeIterator

OMRedBlackTreeIterator::OMRedBlackTreeIterator

template <class *Key*, **class** *Value***>**
OMRedBlackTreeIterator<*Key*, *Value***>::OMRedBlackTreeIterator(const OMRedBlackTree&ltKey, Value>&** *redBlackTree*, **OMIteratorPosition** *initialPosition***)**

Create an OMRedBlackTreeIterator over the given OMRedBlackTree *redBlackTree* and initialize it to the given *initialPosition*. If *initialPosition* is specified as OMBefore then this OMRedBlackTreeIterator is made ready to traverse the associated OMRedBlackTree in the forward direction (increasing *Key*s). If *initialPosition* is specified as OMAfter then this OMRedBlackTreeIterator is made ready to traverse the associated OMRedBlackTree in the reverse direction (decreasing *Key*s).

Defined in: OMRedBlackTreeIteratorT.h

Parameters

redBlackTree
The OMRedBlackTree over which to iterate.
initialPosition
The initial position for this OMRedBlackTreeIterator.

Class Template Arguments

Key
The type of the unique key that identifies the contained values.
Value
The type of the contained values.
Back to OMRedBlackTreeIterator

OMRedBlackTreeIterator::operator++

template <class *Key*, **class** *Value***>**
bool OMRedBlackTreeIterator<*Key*, *Value***>::operator++(void)**

Advance this OMRedBlackTreeIterator to the next *Value*, if any. If the end of the associated OMRedBlackTree is not reached then the result is **true** , **valid** becomes **true** and **after** becomes **false** . If the end of the associated OMRedBlackTree is reached then the result is **false** , **valid** becomes **false** and **after** becomes **true** .

Defined in: OMRedBlackTreeIteratorT.h

Return Value

false if this OMRedBlackTreeIterator has passed the last *Value*, **true** otherwise.

Class Template Arguments

Key
The type of the unique key that identifies the contained values.
Value
The type of the contained values.
Back to OMRedBlackTreeIterator

OMRedBlackTreeIterator::operator--

template <class *Key*, **class** *Value***>**
bool OMRedBlackTreeIterator<*Key*, *Value***>::operator--(void)**

Retreat this OMRedBlackTreeIterator to the previous *Value*, if any. If the beginning of the associated OMRedBlackTree is not reached then the result is **true** , **valid** becomes **true** and **before** becomes **false** . If the beginning of the associated OMRedBlackTree is reached then the result is **false** , **valid** becomes **false** and **before** becomes **true** .

Defined in: OMRedBlackTreeIteratorT.h

Return Value

false if this OMRedBlackTreeIterator has passed the first *Value*, **true** otherwise.

Class Template Arguments

Key
The type of the unique key that identifies the contained values.
Value
The type of the contained values.
Back to OMRedBlackTreeIterator

OMRedBlackTreeIterator::reset

template <class *Key*, **class** *Value***>**
void OMRedBlackTreeIterator<*Key*, *Value***>::reset(OMIteratorPosition** *initialPosition***)**

Reset this OMRedBlackTreeIterator to the given *initialPosition*. If *initialPosition* is specified as OMBefore then this OMRedBlackTreeIterator is made ready to traverse the associated OMRedBlackTree in the forward direction (increasing *Keys*). If *initialPosition* is specified as OMAfter then this OMRedBlackTreeIterator is made ready to traverse the associated OMRedBlackTree in the reverse direction (decreasing *Keys*).

Defined in: OMRedBlackTreeIteratorT.h

Parameters

initialPosition
The position to which this OMRedBlackTreeIterator should be reset.

Class Template Arguments

Key
The type of the unique key that identifies the contained values.
Value
The type of the contained values.
Back to OMRedBlackTreeIterator

OMRedBlackTreeIterator::setValue

template <class *Key*, **class** *Value***>**
Value OMRedBlackTreeIterator<*Key*, *Value***>::setValue(const Key** *ANAME*, *k***)**

Set the *Value* in the associated OMRedBlackTree at the position currently designated by this OMRedBlackTreeIterator to *newValue*. The previous *Value* is returned. To preserve the ordering of *Keys*, the *Key* of *newValue* must be the same as that of the existing *Value*.

Defined in: OMRedBlackTreeIteratorT.h

Return Value

The previous *Value*.

Parameters

ANAME
: The key *Key*.

k
: The new *Value*.

Class Template Arguments

Key
: The type of the unique key that identifies the contained values.

Value
: The type of the contained values.

Back to OMRedBlackTreeIterator

OMRedBlackTreeIterator::value

template <class *Key*, **class** *Value***>**
Value& OMRedBlackTreeIterator<*Key*, *Value***>::value(void) const**

Return the *Value* in the associated OMRedBlackTree at the position currently designated by this OMRedBlackTreeIterator.

Defined in: OMRedBlackTreeIteratorT.h

Return Value

The *Value* at the current position.

Class Template Arguments

Key
: The type of the unique key that identifies the contained values.

Value
: The type of the contained values.

Back to OMRedBlackTreeIterator

OMRedBlackTreeIterator::~OMRedBlackTreeIterator

template <class *Key*, **class** *Value***>**
OMRedBlackTreeIterator<*Key*, *Value***>::~OMRedBlackTreeIterator(void)**

Destroy this OMRedBlackTreeIterator.

Defined in: OMRedBlackTreeIteratorT.h

Class Template Arguments

Key
The type of the unique key that identifies the contained values.
Value
The type of the contained values.
Back to OMRedBlackTreeIterator

OMReferenceContainer class

OMReferenceContainer **class OMReferenceContainer**

Abstract base class for object reference containers supported by the Object Manager.

Defined in: OMReferenceContainer.h

Author

Tim Bingham - tjb - (Avid Technology, Inc.)

Class Members

Public members.
virtual void insertObject(const OMObject* object)
Insert *object* into this **OMReferenceContainer.**
virtual bool containsObject(const OMObject* object) const
Does this **OMReferenceContainer** contain *object* ?
virtual size_t count(void) const
The number of *OMObject*s in this **OMReferenceContainer.**
virtual void removeObject(const OMObject* object)
Remove *object* from this **OMReferenceContainer.**
virtual void removeAllObjects(void)
Remove all objects from this **OMReferenceContainer.**
virtual OMReferenceContainerIterator* createIterator(void) const
Create an OMReferenceContainerIterator over this **OMReferenceContainer.**

OMReferenceContainerIterator class

OMReferenceContainerIterator **class OMReferenceContainerIterator**

Abstract base class for iterators over Object Manager reference containers. The references may be Strong or Weak. The collections may be Vectors or Sets.

Defined in: OMReferenceContainerIter.h

Author

Tim Bingham - tjb - (Avid Technology, Inc.)

Class Members

Public members.

virtual ~OMReferenceContainerIterator(void)

Destroy this **OMReferenceContainerIterator.**

virtual OMReferenceContainerIterator* copy(void) const

Create a copy of this **OMReferenceContainerIterator.**

virtual void reset(OMIteratorPosition initialPosition = OMBefore)

Reset this **OMReferenceContainerIterator** to the given *initialPosition*. If *initialPosition* is specified as OMBefore then this **OMReferenceContainerIterator** is made ready to traverse the associated reference container in the forward direction (increasing *Key*s). If *initialPosition* is specified as OMAfter then this **OMReferenceContainerIterator** is made ready to traverse the associated reference container in the reverse direction (decreasing *Key*s).

virtual bool before(void) const

Is this **OMReferenceContainerIterator** positioned before the first *OMObject* ?

virtual bool after(void) const

Is this **OMReferenceContainerIterator** positioned after the last *OMObject* ?

virtual bool valid(void) const

Is this **OMReferenceContainerIterator** validly positioned on a *OMObject* ?

virtual size_t count(void) const

The number of *OMObject*s in the associated reference container.

virtual bool operator++()

Advance this **OMReferenceContainerIterator** to the next *OMObject*, if any. If the end of the associated reference container is not reached then the result is **true** , **valid** becomes **true** and **after** becomes **false** . If the end of the associated reference container is reached then the result is **false** , **valid** becomes **false** and **after** becomes **true** .

virtual bool operator--()

Retreat this **OMReferenceContainerIterator** to the previous *OMObject*, if any. If the beginning of the associated reference container is not reached then the result is **true** , **valid** becomes **true** and **before** becomes **false** . If the beginning of the associated reference container is reached then the result is **false** , **valid** becomes **false** and **before** becomes **true** .

virtual OMObject* currentObject(void) const

Return the *OMObject* in the associated reference container at the position currently designated by this **OMReferenceContainerIterator.**

OMReferenceProperty class

OMReferenceProperty **class OMReferenceProperty: public OMProperty**

Abstract base class for persistent reference properties supported by the Object Manager.

Defined in: OMRefProperty.h

Author

Tim Bingham - tjb - (Avid Technology, Inc.)

Class Members

Public members.
OMReferenceProperty(const OMPropertyId propertyId, const OMStoredForm storedForm, const wchar_t* name)
Constructor.
virtual ~OMReferenceProperty(void)
Destructor.
virtual size_t bitsSize**(void) const**
The size of the raw bits of this **OMReferenceProperty.** The size is given in bytes.
virtual OMObject* getObject(void) const
Get the value of this **OMReferenceProperty.**
virtual OMObject* setObject(const OMObject* object)
set the value of this **OMReferenceProperty.**

OMReferenceProperty::bitsSize

size_t OMReferenceProperty::bitsSize(void) const

The size of the raw bits of this OMReferenceProperty. The size is given in bytes.

Defined in: OMRefProperty.cpp

Return Value

The size of the raw bits of this OMReferenceProperty in bytes.

Back to OMReferenceProperty

OMReferenceSet class

OMReferenceSet **class OMReferenceSet: public OMReferenceContainer**

Sets of uniquely identified objects supported by the Object Manager. Objects are accessible by unique identifier (the key). The objects are not ordered. Duplicates objects are not allowed.

Defined in: OMReferenceSet.h

Class Template Arguments

ReferencedObject
The type of the referenced objects.
UniqueIdentification
The type of the unique key used to identify the referenced objects.

Author

Tim Bingham - tjb - (Avid Technology, Inc.)

Class Members

Public members.

OMReferenceSet**(void)**

Constructor.

virtual ~OMReferenceSet**(void)**

Destructor.

size_t count**(void) const**

The number of *ReferencedObject*s in this **OMReferenceSet.**

void insert**(const ReferencedObject* object)**

Insert *object* into this **OMReferenceSet.**

bool ensurePresent**(const ReferencedObject* object)**

If it is not already present, insert *object* into this **OMReferenceSet** and return true, otherwise return false.

void appendValue**(const ReferencedObject* object)**

Append the given *ReferencedObject object* to this **OMReferenceSet.**

ReferencedObject* remove**(const UniqueIdentification& identification)**

Remove the *ReferencedObject* identified by *identification* from this **OMReferenceSet.**

bool ensureAbsent**(const UniqueIdentification& identification)**

If it is present, remove the *ReferencedObject* identified by *identification* from this **OMReferenceSet** and return true, otherwise return false.

void removeValue**(const ReferencedObject* object)**

Remove *object* from this **OMReferenceSet.**

bool ensureAbsent**(const ReferencedObject* object)**

If it is present, remove *object* from this **OMReferenceSet** and return true, otherwise return false.

bool containsValue**(const ReferencedObject* object) const**

Does this **OMReferenceSet** contain *object* ?

virtual bool contains**(const UniqueIdentification& identification) const**

Does this **OMReferenceSet** contain a *ReferencedObject* identified by *identification*?

ReferencedObject* value**(const UniqueIdentification& identification) const**

The *ReferencedObject* in this **OMReferenceSet** identified by *identification*.

virtual bool find**(const UniqueIdentification& identification, ReferencedObject*& object) const**

Find the *ReferencedObject* in this **OMReferenceSet** identified by *identification*. If the object is found it is returned in *object* and the result is true. If the element is not found the result is false.

virtual void insertObject**(const OMObject* object)**

Insert *object* into this **OMReferenceSet.**

virtual bool containsObject**(const OMObject* object) const**

Does this **OMReferenceSet** contain *object* ?

virtual void removeObject**(const OMObject* object)**

Remove *object* from this **OMReferenceSet.**

virtual void removeAllObjects**(void)**

Remove all objects from this **OMReferenceSet.**

virtual OMReferenceContainerIterator* createIterator**(void) const**

Create an OMReferenceContainerIterator over this **OMReferenceSet.**

virtual OMObject* remove**(void* identification)**

Remove the OMObject identified by *identification* from this **OMReferenceSet.**

virtual bool contains**(void* identification) const**

Does this **OMReferenceSet** contain an OMObject identified by *identification* ?

virtual bool findObject**(void* identification, OMObject*& object) const**

Find the OMObject in this **OMReferenceSet** identified by *identification*. If the object is found it is returned in *object* and the result is **true** . If the object is not found the result is **false** .

OMReferenceSet::appendValue

template <class *UniqueIdentification*, **class** *ReferencedObject***>**
void OMReferenceSet<*UniqueIdentification*, *ReferencedObject***>::appendValue(const ReferencedObject*** *object***)**

Append the given *ReferencedObject object* to this OMReferenceSet.

Defined in: OMReferenceSetT.h

Parameters

object
A pointer to a *ReferencedObject*.

Class Template Arguments

UniqueIdentification
The type of the unique key used to identify the referenced objects.
ReferencedObject
The type of the referenced objects.
Back to OMReferenceSet

OMReferenceSet::contains

template <class *UniqueIdentification*, **class** *ReferencedObject***>**
bool OMReferenceSet<*UniqueIdentification*, *ReferencedObject***>::contains(const UniqueIdentification&** *identification***)**

Does this OMReferenceSet contain a *ReferencedObject* identified by *identification*?

Defined in: OMReferenceSetT.h

Return Value

True if the object is found, false otherwise.

Parameters

identification
The unique identification of the desired object, the search key.

Class Template Arguments

UniqueIdentification
The type of the unique key used to identify the referenced objects.
ReferencedObject
The type of the referenced objects.
Back to OMReferenceSet

OMReferenceSet::contains

template <class *UniqueIdentification*, **class** *ReferencedObject***>**
bool OMReferenceSet<*UniqueIdentification*, *ReferencedObject***>::contains(void*** *identification***)**

Does this OMReferenceSet contain an OMObject identified by *identification* ?

Defined in: OMReferenceSetT.h

Return Value

TBS

Parameters

identification
TBS

Class Template Arguments

UniqueIdentification
The type of the unique key used to identify the referenced objects.
ReferencedObject
The type of the referenced objects.
Back to OMReferenceSet

OMReferenceSet::containsObject

template <class *UniqueIdentification*, **class** *ReferencedObject***>**
bool OMReferenceSet<*UniqueIdentification*, *ReferencedObject***>::containsObject(const OMObject*** *object***)**

Does this OMReferenceSet contain *object* ?

Defined in: OMReferenceSetT.h

Return Value

TBS

Parameters

object
TBS

Class Template Arguments

UniqueIdentification
The type of the unique key used to identify the referenced objects.
ReferencedObject
The type of the referenced objects.
Back to OMReferenceSet

OMReferenceSet::containsValue

template <class *UniqueIdentification*, **class** *ReferencedObject***>**
bool OMReferenceSet<*UniqueIdentification*, *ReferencedObject***>::containsValue(const ReferencedObject*** *object***)**

Does this OMReferenceSet contain *object* ?

Defined in: OMReferenceSetT.h

Parameters

object
A pointer to a *ReferencedObject*.

Class Template Arguments

UniqueIdentification
The type of the unique key used to identify the referenced objects.
ReferencedObject
The type of the referenced objects.
Back to OMReferenceSet

OMReferenceSet::count

size_t OMReferenceSet::count(void) const

The number of *ReferencedObjects* in this OMReferenceSet.

Defined in: OMReferenceSetT.h

Back to OMReferenceSet

OMReferenceSet::createIterator

template <class *UniqueIdentification*, **class** *ReferencedObject***>**
OMReferenceContainerIterator* OMReferenceSet<*UniqueIdentification*, *ReferencedObject***>::createIterator(void)**

Create an OMReferenceContainerIterator over this OMReferenceSet.

Defined in: OMReferenceSetT.h

Return Value

TBS

Class Template Arguments

UniqueIdentification
: The type of the unique key used to identify the referenced objects.

ReferencedObject
: The type of the referenced objects.

Back to OMReferenceSet

OMReferenceSet::ensureAbsent

template <class *UniqueIdentification*, **class** *ReferencedObject***>**
bool OMReferenceSet<*UniqueIdentification*, *ReferencedObject***>::ensureAbsent(const ReferencedObject*** *object***)**

If it is present, remove *object* from this OMReferenceSet and return true, otherwise return false.

Defined in: OMReferenceSetT.h

Return Value

True if the object was removed, false if it was already absent.

Parameters

object
: The object to remove.

Class Template Arguments

UniqueIdentification
: The type of the unique key used to identify the referenced objects.

ReferencedObject
The type of the referenced objects.
Back to OMReferenceSet

OMReferenceSet::ensureAbsent

template <class *UniqueIdentification*, **class** *ReferencedObject***>**
bool OMReferenceSet<*UniqueIdentification*, *ReferencedObject***>::ensureAbsent(const UniqueIdentification&** *identification***)**

If it is present, remove the *ReferencedObject* identified by *identification* from this OMReferenceSet and return true, otherwise return false.

Defined in: OMReferenceSetT.h

Return Value

True if the object was removed, false if it was already absent.

Parameters

identification
The object to remove.

Class Template Arguments

UniqueIdentification
The type of the unique key used to identify the referenced objects.
ReferencedObject
The type of the referenced objects.
Back to OMReferenceSet

OMReferenceSet::ensurePresent

template <class *UniqueIdentification*, **class** *ReferencedObject***>**
bool OMReferenceSet<*UniqueIdentification*, *ReferencedObject***>::ensurePresent(const ReferencedObject*** *object***)**

If it is not already present, insert *object* into this OMReferenceSet and return true, otherwise return false.

Defined in: OMReferenceSetT.h

Return Value

True if the object was inserted, false if it was already present.

Parameters

object
The object to insert.

Class Template Arguments

UniqueIdentification
The type of the unique key used to identify the referenced objects.
ReferencedObject
The type of the referenced objects.
Back to OMReferenceSet

OMReferenceSet::find

template <class *UniqueIdentification*, **class** *ReferencedObject***>**
bool OMReferenceSet<*UniqueIdentification*, *ReferencedObject***>::find(const UniqueIdentification&** *identification*, **ReferencedObject*&** *object***) const**

Find the *ReferencedObject* in this OMReferenceSet identified by *identification*. If the object is found it is returned in *object* and the result is true. If the element is not found the result is false.

Defined in: OMReferenceSetT.h

Return Value

True if the object is found, false otherwise.

Parameters

identification
The unique identification of the desired object, the search key.
object
A pointer to a *ReferencedObject* by reference.

Class Template Arguments

UniqueIdentification
The type of the unique key used to identify the referenced objects.
ReferencedObject
The type of the referenced objects.
Back to OMReferenceSet

OMReferenceSet::findObject

template <class *UniqueIdentification*, **class** *ReferencedObject***>**
bool OMReferenceSet<*UniqueIdentification*, *ReferencedObject***>::findObject(void*** *identification*, **OMObject*&** *object***)**

Find the OMObject in this OMReferenceSet identified by *identification*. If the object is found it is returned in *object* and the result is **true** . If the object is not found the result is **false** .

Defined in: OMReferenceSetT.h

Return Value

TBS

Parameters

identification
: TBS

object
: TBS

Class Template Arguments

UniqueIdentification
: The type of the unique key used to identify the referenced objects.

ReferencedObject
: The type of the referenced objects.

Back to OMReferenceSet

OMReferenceSet::insert

template <class *UniqueIdentification*, **class** *ReferencedObject***>**
void OMReferenceSet<*UniqueIdentification*, *ReferencedObject***>::insert(const ReferencedObject*** *object***)**

Insert *object* into this OMReferenceSet.

Defined in: OMReferenceSetT.h

Parameters

object
: The object to insert.

Class Template Arguments

UniqueIdentification
: The type of the unique key used to identify the referenced objects.

ReferencedObject
The type of the referenced objects.
Back to OMReferenceSet

OMReferenceSet::insertObject

template <class *UniqueIdentification*, **class** *ReferencedObject***>**
void OMReferenceSet<*UniqueIdentification*, *ReferencedObject***>::insertObject(const OMObject*** *object***)**

Insert *object* into this OMReferenceSet.

Defined in: OMReferenceSetT.h

Parameters

object
TBS

Class Template Arguments

UniqueIdentification
The type of the unique key used to identify the referenced objects.
ReferencedObject
The type of the referenced objects.
Back to OMReferenceSet

OMReferenceSet::OMReferenceSet

OMReferenceSet::OMReferenceSet(void)

Constructor.

Defined in: OMReferenceSetT.h

Back to OMReferenceSet

OMReferenceSet::remove

template <class *UniqueIdentification*, **class** *ReferencedObject***>**
OMObject* OMReferenceSet<*UniqueIdentification*, *ReferencedObject***>::remove(void*** *identification***)**

Remove the OMObject identified by *identification* from this OMReferenceSet.

Defined in: OMReferenceSetT.h

Return Value

TBS

Parameters

identification
TBS

Class Template Arguments

UniqueIdentification
The type of the unique key used to identify the referenced objects.
ReferencedObject
The type of the referenced objects.
Back to OMReferenceSet

OMReferenceSet::remove

template <class *UniqueIdentification*, **class** *ReferencedObject***>**
ReferencedObject* OMReferenceSet<*UniqueIdentification*, *ReferencedObject***>::remove(const UniqueIdentification&** *identification***)**

Remove the *ReferencedObject* identified by *identification* from this OMReferenceSet.

Defined in: OMReferenceSetT.h

Return Value

A pointer to the removed *ReferencedObject*. If lazy loading is enabled and the referenced object was never loaded the value returned is 0.

Parameters

identification
The unique identification of the object to be removed, the search key.

Class Template Arguments

UniqueIdentification
The type of the unique key used to identify the referenced objects.
ReferencedObject
The type of the referenced objects.
Back to OMReferenceSet

OMReferenceSet::removeAllObjects

template <class *UniqueIdentification*, **class** *ReferencedObject***>**
void OMReferenceSet<*UniqueIdentification*, *ReferencedObject***>::removeAllObjects(void)**

Remove all objects from this OMReferenceSet.

Defined in: OMReferenceSetT.h

Class Template Arguments

UniqueIdentification
The type of the unique key used to identify the referenced objects.
ReferencedObject
The type of the referenced objects.
Back to OMReferenceSet

OMReferenceSet::removeObject

template <class *UniqueIdentification*, **class** *ReferencedObject***>**
void OMReferenceSet<*UniqueIdentification*, *ReferencedObject***>::removeObject(const OMObject*** *object***)**

Remove *object* from this OMReferenceSet.

Defined in: OMReferenceSetT.h

Parameters

object
TBS

Class Template Arguments

UniqueIdentification
The type of the unique key used to identify the referenced objects.
ReferencedObject
The type of the referenced objects.
Back to OMReferenceSet

OMReferenceSet::removeValue

template <class *UniqueIdentification*, **class** *ReferencedObject***>**
void OMReferenceSet<*UniqueIdentification*, *ReferencedObject***>::removeValue(const ReferencedObject*** *object***)**

Remove *object* from this OMReferenceSet.

Defined in: OMReferenceSetT.h

Parameters

object
A pointer to a *ReferencedObject*.

Class Template Arguments

UniqueIdentification
The type of the unique key used to identify the referenced objects.
ReferencedObject
The type of the referenced objects.
Back to OMReferenceSet

OMReferenceSet::value

template <class *UniqueIdentification*, **class** *ReferencedObject***>**
ReferencedObject* OMReferenceSet<*UniqueIdentification*, *ReferencedObject***>::value(const UniqueIdentification&** *identification***) const**

The *ReferencedObject* in this OMReferenceSet identified by *identification*.

Defined in: OMReferenceSetT.h

Return Value

A pointer to the *ReferencedObject*.

Parameters

identification
The unique identification of the desired object, the search key.

Class Template Arguments

UniqueIdentification
The type of the unique key used to identify the referenced objects.
ReferencedObject
The type of the referenced objects.
Back to OMReferenceSet

OMReferenceSet::~OMReferenceSet

OMReferenceSet::~OMReferenceSet(void)

Destructor.

Defined in: OMReferenceSetT.h

Back to OMReferenceSet

OMReferenceSetIterator class

OMReferenceSetIterator **class OMReferenceSetIterator**

Iterators over OMReferenceSets.

Defined in: OMReferenceSetIter.h

Class Template Arguments

ReferencedObject
The type of the contained objects.

Author

Tim Bingham - tjb - (Avid Technology, Inc.)

Class Members

Public members.

OMReferenceSetIterator**(const OMReferenceSet<UniqueIdentification, ReferencedObject>& set, OMIteratorPosition initialPosition = OMBefore)**
Create an **OMReferenceSetIterator** over the given OMReferenceSet *set* and initialize it to the given *initialPosition*. If *initialPosition* is specified as OMBefore then this **OMReferenceSetIterator** is made ready to traverse the associated OMReferenceSet in the forward direction (increasing *Key*s). If *initialPosition* is specified as OMAfter then this **OMReferenceSetIterator** is made ready to traverse the associated OMReferenceSet in the reverse direction (decreasing *Key*s).

virtual ~OMReferenceSetIterator**(void)**
Destroy this **OMReferenceSetIterator.**

virtual OMReferenceContainerIterator* copy**(void) const**
Create a copy of this **OMReferenceSetIterator.**

virtual void reset**(OMIteratorPosition initialPosition = OMBefore)**
Reset this **OMReferenceSetIterator** to the given *initialPosition*. If *initialPosition* is specified as OMBefore then this **OMReferenceSetIterator** is made ready to traverse the associated OMReferenceSet in the forward direction (increasing *Key*s). If *initialPosition* is specified as OMAfter then this **OMReferenceSetIterator** is made ready to traverse the associated OMReferenceSet in the reverse direction (decreasing *Key*s).

virtual bool before**(void) const**
Is this **OMReferenceSetIterator** positioned before the first *ReferencedObject* ?

virtual bool after**(void) const**
Is this **OMReferenceSetIterator** positioned after the last *ReferencedObject* ?

virtual bool valid**(void) const**

Is this **OMReferenceSetIterator** validly positioned on a *ReferencedObject* ?

virtual size_t count**(void) const**

The number of *ReferencedObject*s in the associated OMReferenceSet.

virtual bool operator++**()**

Advance this **OMReferenceSetIterator** to the next *ReferencedObject*, if any. If the end of the associated OMReferenceSet is not reached then the result is **true** , **valid** becomes **true** and **after** becomes **false** . If the end of the associated OMReferenceSet is reached then the result is **false** , **valid** becomes **false** and **after** becomes **true** .

virtual bool operator--**()**

Retreat this **OMReferenceSetIterator** to the previous *ReferencedObject*, if any. If the beginning of the associated OMReferenceSet is not reached then the result is **true** , **valid** becomes **true** and **before** becomes **false** . If the beginning of the associated OMReferenceSet is reached then the result is **false** , **valid** becomes **false** and **before** becomes **true** .

virtual ReferencedObject* value**(void) const**

Return the *ReferencedObject* in the associated OMReferenceSet at the position currently designated by this **OMReferenceSetIterator.**

virtual ReferencedObject* setValue**(const ReferencedObject* newObject)**

Set the *ReferencedObject* in the associated OMReferenceSet at the position currently designated by this **OMReferenceSetIterator** to *newObject*. The previous *ReferencedObject*, if any, is returned. To preserve the ordering of *Key*s, the *Key* of *newObject* must be the same as that of the existing *ReferencedObject*.

virtual OMObject* currentObject**(void) const**

Return the *OMObject* in the associated reference container at the position currently designated by this **OMReferenceSetIterator.**

UniqueIdentification identification**(void) const**

Return the *Key* of the *ReferencedObject* in the associated OMReferenceSet at the position currently designated by this **OMReferenceSetIterator.**

OMReferenceSetIterator**(const SetIterator& iter)**

Create an **OMReferenceSetIterator** given an underlying OMSetIterator.

OMReferenceSetIterator::after

template <class *ReferencedObject***>**
bool OMReferenceSetIterator<*ReferencedObject***>::after(void) const**

Is this OMReferenceSetIterator positioned after the last *ReferencedObject* ?

Defined in: OMReferenceSetIterT.h

Return Value

true if this OMReferenceSetIterator is positioned after the last *ReferencedObject*, **false** otherwise.

Class Template Arguments

ReferencedObject

The type of the contained objects.

Back to OMReferenceSetIterator

OMReferenceSetIterator::before

template <class *ReferencedObject***>**
bool OMReferenceSetIterator<*ReferencedObject***>::before(void) const**

Is this OMReferenceSetIterator positioned before the first *ReferencedObject* ?

Defined in: OMReferenceSetIterT.h

Return Value

true if this OMReferenceSetIterator is positioned before the first *ReferencedObject*, **false** otherwise.

Class Template Arguments

ReferencedObject
The type of the contained objects.

Back to OMReferenceSetIterator

OMReferenceSetIterator::copy

template <class *ReferencedObject***>**
OMReferenceContainerIterator* OMReferenceSetIterator<*ReferencedObject***>::copy(void) const**

Create a copy of this OMReferenceSetIterator.

Defined in: OMReferenceSetIterT.h

Return Value

The new OMReferenceSetIterator.

Class Template Arguments

ReferencedObject
The type of the contained objects.

Back to OMReferenceSetIterator

OMReferenceSetIterator::count

template <class *ReferencedObject***>**
size_t OMReferenceSetIterator<*ReferencedObject***>::count(void) const**

The number of *ReferencedObject*s in the associated OMReferenceSet.

Defined in: OMReferenceSetIterT.h

Return Value

The number of *ReferencedObject*s

Class Template Arguments

ReferencedObject
The type of the contained objects.

Back to OMReferenceSetIterator

OMReferenceSetIterator::currentObject

template <class *ReferencedObject***>**
OMObject* OMReferenceSetIterator<*ReferencedObject***>::currentObject(void) const**

Return the *OMObject* in the associated reference container at the position currently designated by this OMReferenceSetIterator.

Defined in: OMReferenceSetIterT.h

Return Value

The OMObject at the current position.

Class Template Arguments

ReferencedObject
The type of the contained objects.

Back to OMReferenceSetIterator

OMReferenceSetIterator::identification

template <class *ReferencedObject***>**
UniqueIdentification OMReferenceSetIterator<*ReferencedObject***>::identification(void) const**

Return the *Key* of the *ReferencedObject* in the associated OMReferenceSet at the position currently designated by this OMReferenceSetIterator.

Defined in: OMReferenceSetIterT.h

Return Value

The *Key* at the current position.

Class Template Arguments

ReferencedObject
The type of the contained objects.
Back to OMReferenceSetIterator

OMReferenceSetIterator::OMReferenceSetIterator

template <class *ReferencedObject***>**
OMReferenceSetIterator<*ReferencedObject***>::OMReferenceSetIterator(const SetIterator&** *iter***)**

Create an OMReferenceSetIterator given an underlying OMSetIterator.

Defined in: OMReferenceSetIterT.h

Parameters

iter
The underlying OMSetIterator.

Class Template Arguments

ReferencedObject
The type of the contained objects.
Back to OMReferenceSetIterator

OMReferenceSetIterator::OMReferenceSetIterator

template <class *ReferencedObject***>**
OMReferenceSetIterator<*ReferencedObject***>::OMReferenceSetIterator(const OMReferenceSet<** *UniqueIdentification***, ReferencedObject>&** *set***)**

Create an OMStrongReferenceSetIterator over the given OMReferenceSet *set* and initialize it to the given *initialPosition*. If *initialPosition* is specified as OMBefore then this OMStrongReferenceSetIterator is made ready to traverse the associated OMReferenceSet in the forward direction (increasing *Keys*). If *initialPosition* is specified as OMAfter then this OMStrongReferenceSetIterator is made ready to traverse the associated OMReferenceSet in the reverse direction (decreasing *Keys*).

Defined in: OMReferenceSetIterT.h

Parameters

UniqueIdentification
The OMStrongReferenceSet over which to iterate.
set
The initial position for this OMReferenceSetIterator.

Class Template Arguments

ReferencedObject
The type of the contained objects.
Back to OMReferenceSetIterator

OMReferenceSetIterator::operator++

template <class *ReferencedObject***>**
bool OMReferenceSetIterator<*ReferencedObject***>::operator++(void)**

Advance this OMReferenceSetIterator to the next *ReferencedObject*, if any. If the end of the associated OMReferenceSet is not reached then the result is **true** , **valid** becomes **true** and **after** becomes **false** . If the end of the associated OMReferenceSet is reached then the result is **false** , **valid** becomes **false** and **after** becomes **true** .

Defined in: OMReferenceSetIterT.h

Return Value

false if this OMReferenceSetIterator has passed the last *ReferencedObject*, **true** otherwise.

Class Template Arguments

ReferencedObject
The type of the contained objects.
Back to OMReferenceSetIterator

OMReferenceSetIterator::operator--

template <class *ReferencedObject***>**
bool OMReferenceSetIterator<*ReferencedObject***>::operator--(void)**

Retreat this OMReferenceSetIterator to the previous *ReferencedObject*, if any. If the beginning of the associated OMReferenceSet is not reached then the result is **true** , **valid** becomes **true** and **before** becomes **false** . If the beginning of the associated OMReferenceSet is reached then the result is **false** , **valid** becomes **false** and **before** becomes **true** .

Defined in: OMReferenceSetIterT.h

Return Value

false if this OMReferenceSetIterator has passed the first *ReferencedObject*, **true** otherwise.

Class Template Arguments

ReferencedObject
The type of the contained objects.
Back to OMReferenceSetIterator

OMReferenceSetIterator::reset

template <class *ReferencedObject***>**
void OMReferenceSetIterator<*ReferencedObject***>::reset(OMIteratorPosition** *initialPosition***)**

Reset this OMReferenceSetIterator to the given *initialPosition*. If *initialPosition* is specified as OMBefore then this OMReferenceSetIterator is made ready to traverse the associated OMReferenceSet in the forward direction (increasing *Keys*). If *initialPosition* is specified as OMAfter then this OMReferenceSetIterator is made ready to traverse the associated OMReferenceSet in the reverse direction (decreasing *Keys*).

Defined in: OMReferenceSetIterT.h

Parameters

initialPosition
The position to which this OMReferenceSetIterator should be reset.

Class Template Arguments

ReferencedObject
The type of the contained objects.

Back to OMReferenceSetIterator

OMReferenceSetIterator::setValue

template <class *ReferencedObject***>**
ReferencedObject* OMReferenceSetIterator<*ReferencedObject***>::setValue(const ReferencedObject*** *newObject***)**

Set the *ReferencedObject* in the associated OMReferenceSet at the position currently designated by this OMReferenceSetIterator to *newObject*. The previous *ReferencedObject*, if any, is returned. To preserve the ordering of *Keys*, the *Key* of *newObject* must be the same as that of the existing *ReferencedObject*.

Defined in: OMReferenceSetIterT.h

Return Value

The previous *ReferencedObject* if any, otherwise 0.

Parameters

newObject
The new *ReferencedObject*.

Class Template Arguments

ReferencedObject
The type of the contained objects.

Back to OMReferenceSetIterator

OMReferenceSetIterator::valid

template <class *ReferencedObject***>**
bool OMReferenceSetIterator<*ReferencedObject***>::valid(void) const**

Is this OMReferenceSetIterator validly positioned on a *ReferencedObject* ?

Defined in: OMReferenceSetIterT.h

Return Value

true if this OMReferenceSetIterator is positioned on a *ReferencedObject*, **false** otherwise.

Class Template Arguments

ReferencedObject
The type of the contained objects.

Back to OMReferenceSetIterator

OMReferenceSetIterator::value

template <class *ReferencedObject***>**
ReferencedObject* OMReferenceSetIterator<*ReferencedObject***>::value(void) const**

Return the *ReferencedObject* in the associated OMReferenceSet at the position currently designated by this OMReferenceSetIterator.

Defined in: OMReferenceSetIterT.h

Return Value

The *ReferencedObject* at the current position.

Class Template Arguments

ReferencedObject
The type of the contained objects.

Back to OMReferenceSetIterator

OMReferenceSetIterator::~OMReferenceSetIterator

template <class *ReferencedObject***>**
OMReferenceSetIterator<*ReferencedObject***>::~OMReferenceSetIterator(void)**

Destroy this OMReferenceSetIterator.

Defined in: OMReferenceSetIterT.h

Class Template Arguments

ReferencedObject
The type of the contained objects.

Back to OMReferenceSetIterator

OMReferenceSetProperty class

OMReferenceSetProperty **class OMReferenceSetProperty: public OMContainerProperty, public OMObjectSet**

Abstract base class for persistent object reference set properties supported by the Object Manager.

Defined in: OMRefSetProperty.h

Author

Tim Bingham - tjb - (Avid Technology, Inc.)

Class Members

Public members.

OMReferenceSetProperty**(const OMPropertyId propertyId, const OMStoredForm storedForm, const wchar_t* name)**
Constructor.

virtual ~OMReferenceSetProperty**(void)**
Destructor.

virtual OMReferenceContainer* referenceContainer**(void)**
Convert to OMReferenceContainer.

OMReferenceSetProperty::OMReferenceSetProperty

OMReferenceSetProperty::OMReferenceSetProperty(const OMPropertyId *propertyId*, **const OMStoredForm** *storedForm*, **const wchar_t*** *name***)**

Constructor.

Defined in: OMRefSetProperty.cpp

Parameters

propertyId
The property id.

storedForm

The stored form of this property.

name

The name of this property.

Back to OMReferenceSetProperty

OMReferenceSetProperty::referenceContainer

OMReferenceContainer* OMReferenceSetProperty::referenceContainer(void)

Convert to OMReferenceContainer.

Defined in: OMRefSetProperty.cpp

Return Value

The OMReferenceContainer interface implemented by this OMReferenceSetProperty

Back to OMReferenceSetProperty

OMReferenceSetProperty::~OMReferenceSetProperty

OMReferenceSetProperty::~OMReferenceSetProperty(void)

Destructor.

Defined in: OMRefSetProperty.cpp

Back to OMReferenceSetProperty

OMReferenceVector class

OMReferenceVector **class OMReferenceVector: public OMObjectVector**

Elastic sequential collections of objects supported by the Object Manager. Objects are accessible by index. The order of objects is determined externally. Duplicate objects are allowed.

Defined in: OMReferenceVector.h

Class Template Arguments

ReferencedObject

The type of the referenced object.

Author

Tim Bingham - tjb - (Avid Technology, Inc.)

Class Members

Public members.

OMReferenceVector(void)

Constructor.

virtual ~OMReferenceVector**(void)**

Destructor.

size_t count**(void) const**

The number of *ReferencedObject*s in this **OMReferenceVector.**

ReferencedObject* setValueAt**(const ReferencedObject* object, const size_t index)**

Set the value of this **OMReferenceVector** at position *index* to *object*.

ReferencedObject* clearValueAt**(const size_t index)**

Set the value of this **OMReferenceVector** at position *index* to 0.

ReferencedObject* valueAt**(const size_t index) const**

The value of this **OMReferenceVector** at position *index*.

void getValueAt**(ReferencedObject*& object, const size_t index) const**

Get the value of this **OMReferenceVector** at position *index* into *object*.

bool find**(const size_t index, ReferencedObject*& object) const**

If *index* is valid, get the value of this **OMReferenceVector** at position *index* into *object* and return true, otherwise return false.

void appendValue**(const ReferencedObject* object)**

Append the given *ReferencedObject object* to this **OMReferenceVector.**

void prependValue**(const ReferencedObject* object)**

Prepend the given *ReferencedObject object* to this **OMReferenceVector.**

void insert**(const ReferencedObject* object)**

Insert *object* into this **OMReferenceVector.**

void insertAt**(const ReferencedObject* object, const size_t index)**

Insert *object* into this **OMReferenceVector** at position *index*. Existing objects at *index* and higher are shifted up one index position.

bool containsValue**(const ReferencedObject* object) const**

Does this **OMReferenceVector** contain *object* ?

void removeValue**(const ReferencedObject* object)**

Remove *object* from this **OMReferenceVector.**

ReferencedObject* removeAt**(const size_t index)**

Remove the object from this **OMReferenceVector** at position *index*. Existing objects in this **OMReferenceVector** at *index* + 1 and higher are shifted down one index position.

ReferencedObject* removeLast**(void)**

Remove the last (index == count() - 1) object from this **OMReferenceVector.**

ReferencedObject* removeFirst**(void)**

Remove the first (index == 0) object from this **OMReferenceVector.** Existing objects in this **OMReferenceVector** are shifted down one index position.

size_t indexOfValue**(const ReferencedObject* object) const**

The index of the *ReferencedObject* object*.

size_t countOfValue**(const ReferencedObject* object) const**

The number of occurrences of *object* in this **OMReferenceVector.**

bool containsIndex**(const size_t index) const**

Does this **OMReferenceVector** contain *index* ? Is *index* valid ?

bool findIndex**(const ReferencedObject* object, size_t& index) const**

If this OMStrongReferenceProperty contains *object* then place its index in *index* and return true, otherwise return false.

void grow**(const size_t capacity)**

Increase the capacity of this **OMReferenceVector** so that it can contain at least *capacity* *ReferencedObject*s without having to be resized.

virtual void insertObject**(const OMObject* object)**

Insert *object* into this **OMReferenceVector.**

virtual bool containsObject**(const OMObject* object) const**

Does this **OMReferenceVector** contain *object* ?

virtual void removeObject**(const OMObject* object)**

Remove *object* from this **OMReferenceVector.**

virtual void removeAllObjects**(void)**

Remove all objects from this **OMReferenceVector.**

virtual OMReferenceContainerIterator* createIterator**(void) const**

Create an OMReferenceContainerIterator over this **OMReferenceVector.**

virtual OMObject* setObjectAt**(const OMObject* object, const size_t index)**

Set the value of this **OMReferenceVector** at position *index* to *object*.

virtual OMObject* getObjectAt**(const size_t index) const**

The value of this **OMReferenceVector** at position *index*.

virtual void appendObject**(const OMObject* object)**

Append the given *OMObject object* to this **OMReferenceVector.**

virtual void prependObject**(const OMObject* object)**

Prepend the given *OMObject object* to this **OMReferenceVector.**

virtual OMObject* removeObjectAt**(const size_t index)**

Remove the object from this **OMReferenceVector** at position *index*. Existing objects in this **OMReferenceVector** at *index* + 1 and higher are shifted down one index position.

virtual void insertObjectAt**(const OMObject* object, const size_t index)**

Insert *object* into this **OMReferenceVector** at position *index*. Existing objects at *index* and higher are shifted up one index position.

OMReferenceVector::

OMReferenceVector::(void)

Constructor.

Defined in: OMReferenceVectorT.h

Back to OMReferenceVector

OMReferenceVector::appendObject

template <class *ReferencedObject***>**
void OMReferenceVector<*ReferencedObject***>::appendObject(const OMObject*** *object***)**

Append the given *OMObject object* to this OMReferenceVector.

Defined in: OMReferenceVectorT.h

Parameters

object
TBS

Class Template Arguments

ReferencedObject
The type of the referenced objects.

Back to OMReferenceVector

OMReferenceVector::appendValue

template <class *ReferencedObject***>**
void OMReferenceVector<*ReferencedObject***>::appendValue(const ReferencedObject*** *object***)**

Append the given *ReferencedObject object* to this OMReferenceVector.

Defined in: OMReferenceVectorT.h

Parameters

object
A pointer to a *ReferencedObject*.

Class Template Arguments

ReferencedObject
The type of the referenced objects.

Back to OMReferenceVector

OMReferenceVector::clearValueAt

template <class *ReferencedObject***>**
ReferencedObject* OMReferenceVector<*ReferencedObject***>::clearValueAt(const size_t** *index***)**

Set the value of this OMReferenceVector at position *index* to 0.

Defined in: OMReferenceVectorT.h

Return Value

A pointer to the old *ReferencedObject*.

Parameters

index
The position to clear.

Class Template Arguments

ReferencedObject
The type of the referenced objects.

Back to OMReferenceVector

OMReferenceVector::containsIndex

template <class *ReferencedObject***>**
bool OMReferenceVector<*ReferencedObject***>::containsIndex(const size_t** *index***) const**

Does this OMReferenceVector contain *index* ? Is *index* valid ?

Defined in: OMReferenceVectorT.h

Return Value

True if the index is valid, false otherwise.

Parameters

index
The index.

Class Template Arguments

ReferencedObject
The type of the referenced objects.

Back to OMReferenceVector

OMReferenceVector::containsObject

template <class *ReferencedObject***>**
bool OMReferenceVector<*ReferencedObject***>::containsObject(const OMObject*** *object***)**

Does this OMReferenceVector contain *object* ?

Defined in: OMReferenceVectorT.h

Return Value

TBS

Parameters

object

TBS

Class Template Arguments

ReferencedObject
The type of the referenced objects.

Back to OMReferenceVector

OMReferenceVector::containsValue

template <class *ReferencedObject***>**
bool OMReferenceVector<*ReferencedObject***>::containsValue(const ReferencedObject*** *object***) const**

Does this OMReferenceVector contain *object* ?

Defined in: OMReferenceVectorT.h

Parameters

object
A pointer to a *ReferencedObject*.

Class Template Arguments

ReferencedObject
The type of the referenced objects.

Back to OMReferenceVector

OMReferenceVector::count

size_t OMReferenceVector::count(void) const

The number of *ReferencedObject*s in this OMReferenceVector.

Defined in: OMReferenceVectorT.h

Back to OMReferenceVector

OMReferenceVector::countOfValue

template <class *ReferencedObject***>**
size_t OMReferenceVector<*ReferencedObject***>::countOfValue(const ReferencedObject*** *object***) const**

The number of occurrences of *object* in this OMReferenceVector.

Defined in: OMReferenceVectorT.h

Return Value

The number of occurrences.

Parameters

object
The object to count.

Class Template Arguments

ReferencedObject
The type of the referenced objects.

Back to OMReferenceVector

OMReferenceVector::createIterator

template <class *ReferencedObject***>**
OMReferenceContainerIterator* OMReferenceVector<*ReferencedObject***>::createIterator(void)**

Create an OMReferenceContainerIterator over this OMReferenceVector.

Defined in: OMReferenceVectorT.h

Return Value

TBS

Class Template Arguments

ReferencedObject
The type of the referenced objects.

Back to OMReferenceVector

OMReferenceVector::find

template <class *ReferencedObject***>**
bool OMReferenceVector<*ReferencedObject***>::find(const size_t** *index***, ReferencedObject*&** *object***) const**

If *index* is valid, get the value of this OMReferenceVector at position *index* into *object* and return true, otherwise return false.

Defined in: OMReferenceVectorT.h

Return Value

True if *index* is valid, false otherwise.

Parameters

index
: The position from which to get the *ReferencedObject*.

object
: A pointer to a *ReferencedObject*.

Class Template Arguments

ReferencedObject
: The type of the referenced objects.

Back to OMReferenceVector

OMReferenceVector::findIndex

template <class *ReferencedObject***>**
bool OMReferenceVector<*ReferencedObject***>::findIndex(const ReferencedObject*** *object*, **size_t&** *index***) const**

If this OMReferenceVector contains *object* then place its index in *index* and return true, otherwise return false.

Defined in: OMReferenceVectorT.h

Return Value

True if the object was found, false otherwise.

Parameters

object
: The object for which to search.

index
: The index of the object.

Class Template Arguments

ReferencedObject
: The type of the referenced objects.

Back to OMReferenceVector

OMReferenceVector::getObjectAt

template <class *ReferencedObject***>**
OMObject* OMReferenceVector<*ReferencedObject***>::getObjectAt(const size_t** *index***)**

The value of this OMReferenceVector at position *index*.

Defined in: OMReferenceVectorT.h

Return Value

TBS

Parameters

index
: TBS

Class Template Arguments

ReferencedObject
: The type of the referenced objects.

Back to OMReferenceVector

OMReferenceVector::getValueAt

template <class *ReferencedObject***>**
void OMReferenceVector<*ReferencedObject***>::getValueAt(ReferencedObject*& *object*, const size_t *index*) const**

Get the value of this OMReferenceVector at position *index* into *object*.

Defined in: OMReferenceVectorT.h

Parameters

object
: A pointer to a *ReferencedObject* by reference.

index
: The position from which to get the *ReferencedObject*.

Class Template Arguments

ReferencedObject
: The type of the referenced objects.

Back to OMReferenceVector

OMReferenceVector::grow

template <class *ReferencedObject***>**
void OMReferenceVector<*ReferencedObject***>::grow(const size_t** *capacity***)**

Increase the capacity of this OMReferenceVector so that it can contain at least *capacity ReferencedObject*s without having to be resized.

Defined in: OMReferenceVectorT.h

Parameters

capacity
The desired capacity.

Class Template Arguments

ReferencedObject
The type of the referenced objects.

Back to OMReferenceVector

OMReferenceVector::indexOfValue

template <class *ReferencedObject***>**
size_t OMReferenceVector<*ReferencedObject***>::indexOfValue(const ReferencedObject*** *object***) const**

The index of the *ReferencedObject** *object*.

Defined in: OMReferenceVectorT.h

Return Value

The index.

Parameters

object
A pointer to the *ReferencedObject* to find.

Class Template Arguments

ReferencedObject
The type of the referenced objects.

Back to OMReferenceVector

OMReferenceVector::insert

template <class *ReferencedObject***>**
void OMReferenceVector<*ReferencedObject***>::insert(const ReferencedObject*** *object***)**

Insert *object* into this OMReferenceVector.

Defined in: OMReferenceVectorT.h

Parameters

object
A pointer to a *ReferencedObject*.

Class Template Arguments

ReferencedObject
The type of the referenced objects.

Back to OMReferenceVector

OMReferenceVector::insertAt

template <class *ReferencedObject***>**
void OMReferenceVector<*ReferencedObject***>::insertAt(const ReferencedObject*** *object***, const size_t** *index***)**

Insert *value* into this OMReferenceVector at position *index*. Existing values at *index* and higher are shifted up one index position.

Defined in: OMReferenceVectorT.h

Parameters

object
A pointer to a *ReferencedObject*.

index
The position at which to insert the *ReferencedObject*.

Class Template Arguments

ReferencedObject
The type of the referenced objects.

Back to OMReferenceVector

OMReferenceVector::insertObject

template <class *ReferencedObject***>**
void OMReferenceVector<*ReferencedObject***>::insertObject(const OMObject*** *object***)**

Insert *object* into this OMReferenceVector.

Defined in: OMReferenceVectorT.h

Parameters

object
TBS

Class Template Arguments

ReferencedObject
The type of the referenced objects.

Back to OMReferenceVector

OMReferenceVector::insertObjectAt

template <class *ReferencedObject***>**
void OMReferenceVector<*ReferencedObject***>::insertObjectAt(const OMObject*** *object***, const size_t** *index***)**

Insert *object* into this OMReferenceVector at position *index*. Existing objects at *index* and higher are shifted up one index position.

Defined in: OMReferenceVectorT.h

Parameters

object
TBS

index
TBS

Class Template Arguments

ReferencedObject
The type of the referenced objects.

Back to OMReferenceVector

OMReferenceVector::prependObject

template <class *ReferencedObject***>**
void OMReferenceVector<*ReferencedObject***>::prependObject(const OMObject*** *object***)**

Prepend the given *OMObject object* to this OMReferenceVector.

Defined in: OMReferenceVectorT.h

Parameters

object
TBS

Class Template Arguments

ReferencedObject
The type of the referenced objects.

Back to OMReferenceVector

OMReferenceVector::prependValue

template <class *ReferencedObject***>**
void OMReferenceVector<*ReferencedObject***>::prependValue(const ReferencedObject*** *object***)**

Prepend the given *ReferencedObject object* to this OMReferenceVector.

Defined in: OMReferenceVectorT.h

Parameters

object
A pointer to a *ReferencedObject*.

Class Template Arguments

ReferencedObject
The type of the referenced objects.

Back to OMReferenceVector

OMReferenceVector::removeAllObjects

template <class *ReferencedObject***>**
void OMReferenceVector<*ReferencedObject***>::removeAllObjects(void)**

Remove all objects from this OMReferenceVector.

Defined in: OMReferenceVectorT.h

Class Template Arguments

ReferencedObject
The type of the referenced objects.

Back to OMReferenceVector

OMReferenceVector::removeAt

template <class *ReferencedObject***>**
ReferencedObject* OMReferenceVector<*ReferencedObject***>::removeAt(const size_t** *index***)**

Remove the object from this OMReferenceVector at position *index*. Existing objects in this OMReferenceVector at *index* + 1 and higher are shifted down one index position.

Defined in: OMReferenceVectorT.h

Return Value

A pointer to the removed *ReferencedObject*.

Parameters

index
The position from which to remove the *ReferencedObject*.

Class Template Arguments

ReferencedObject
The type of the referenced objects.

Back to OMReferenceVector

OMReferenceVector::removeFirst

template <class *ReferencedObject***>**
ReferencedObject* OMReferenceVector<*ReferencedObject***>::removeFirst(void)**

Remove the first (index == 0) object from this OMReferenceVector. Existing objects in this OMReferenceVector are shifted down one index position.

Defined in: OMReferenceVectorT.h

Return Value

A pointer to the removed *ReferencedObject*.

Class Template Arguments

ReferencedObject
The type of the referenced objects.

Back to OMReferenceVector

OMReferenceVector::removeLast

template <class *ReferencedObject***>**
ReferencedObject* OMReferenceVector<*ReferencedObject***>::removeLast(void)**

Remove the last (index == count() - 1) object from this OMReferenceVector.

Defined in: OMReferenceVectorT.h

Return Value

A pointer to the removed *ReferencedObject*.

Class Template Arguments

ReferencedObject
The type of the referenced objects.

Back to OMReferenceVector

OMReferenceVector::removeObject

template <class *ReferencedObject***>**
void OMReferenceVector<*ReferencedObject***>::removeObject(const OMObject*** *object***)**

Remove *object* from this OMReferenceVector.

Defined in: OMReferenceVectorT.h

Parameters

object
TBS

Class Template Arguments

ReferencedObject
The type of the referenced objects.

Back to OMReferenceVector

OMReferenceVector::removeObjectAt

template <class *ReferencedObject***>**
OMObject* OMReferenceVector<*ReferencedObject***>::removeObjectAt(const size_t** *index***)**

Remove the object from this OMReferenceVector at position *index*. Existing objects in this OMReferenceVector at *index* + 1 and higher are shifted down one index position.

Defined in: OMReferenceVectorT.h

Return Value

TBS

Parameters

index
TBS

Class Template Arguments

ReferencedObject
The type of the referenced objects.

Back to OMReferenceVector

OMReferenceVector::removeValue

template <class *ReferencedObject***>**
void OMReferenceVector<*ReferencedObject***>::removeValue(const ReferencedObject*** *object***)**

Remove *object* from this OMReferenceVector.

Defined in: OMReferenceVectorT.h

Parameters

object
A pointer to a *ReferencedObject*.

Class Template Arguments

ReferencedObject
The type of the referenced objects.

Back to OMReferenceVector

OMReferenceVector::setObjectAt

template <class *ReferencedObject***>**
OMObject* OMReferenceVector<*ReferencedObject***>::setObjectAt(const OMObject*** *object***)**

Set the value of this OMReferenceVector at position *index* to *object*.

Defined in: OMReferenceVectorT.h

Return Value

TBS

Parameters

object
TBS

Class Template Arguments

ReferencedObject
: The type of the referenced objects.

Back to OMReferenceVector

OMReferenceVector::setValueAt

template <class *ReferencedObject***>**
ReferencedObject* OMReferenceVector<*ReferencedObject***>::setValueAt(const ReferencedObject*** *object***, const size_t** *index***)**

Set the value of this OMReferenceVector at position *index* to *object*.

Defined in: OMReferenceVectorT.h

Return Value

A pointer to the old *ReferencedObject*.

Parameters

object
: A pointer to the new *ReferencedObject*.

index
: The position at which to insert the *ReferencedObject*.

Class Template Arguments

ReferencedObject
: The type of the referenced objects.

Back to OMReferenceVector

OMReferenceVector::valueAt

template <class *ReferencedObject***>**
ReferencedObject* OMReferenceVector<*ReferencedObject***>::valueAt(const size_t** *index***) const**

The value of this OMReferenceVector at position *index*.

Defined in: OMReferenceVectorT.h

Return Value

A pointer to the *ReferencedObject*.

Parameters

index
The position from which to get the *ReferencedObject*.

Class Template Arguments

ReferencedObject
The type of the referenced objects.

Back to OMReferenceVector

OMReferenceVector::~OMReferenceVector

OMReferenceVector::~OMReferenceVector(void)

Destructor.

Defined in: OMReferenceVectorT.h

Back to OMReferenceVector

OMReferenceVectorIterator class

OMReferenceVectorIterator **class OMReferenceVectorIterator**

Iterators over OMReferenceVectors.

Defined in: OMReferenceVectorIter.h

Class Template Arguments

ReferencedObject
The type of the contained objects.

Author

Tim Bingham - tjb - (Avid Technology, Inc.)

Class Members

Public members.
OMReferenceVectorIterator**(const OMReferenceVector&ltReferencedObject>& vector, OMIteratorPosition initialPosition = OMBefore)**
Create an **OMReferenceVectorIterator** over the given OMReferenceVector *vector* and initialize it to the given *initialPosition*. If *initialPosition* is specified as OMBefore then this **OMReferenceVectorIterator** is made ready to traverse the associated OMReferenceVector in the forward direction (increasing indexes). If *initialPosition* is specified as OMAfter then this

OMReferenceVectorIterator is made ready to traverse the associated OMReferenceVector in the reverse direction (decreasing indexes).

virtual OMReferenceContainerIterator* copy**(void) const**

Create a copy of this **OMReferenceVectorIterator.**

virtual ~OMReferenceVectorIterator**(void)**

Destroy this **OMReferenceVectorIterator.**

virtual void reset**(OMIteratorPosition initialPosition = OMBefore)**

Reset this **OMReferenceVectorIterator** to the given *initialPosition*. If *initialPosition* is specified as OMBefore then this **OMReferenceVectorIterator** is made ready to traverse the associated OMReferenceVector in the forward direction (increasing indexes). If *initialPosition* is specified as OMAfter then this **OMReferenceVectorIterator** is made ready to traverse the associated OMReferenceVector in the reverse direction (decreasing indexes).

virtual bool before**(void) const**

Is this **OMReferenceVectorIterator** positioned before the first *ReferencedObject* ?

virtual bool after**(void) const**

Is this **OMReferenceVectorIterator** positioned after the last *ReferencedObject* ?

virtual bool valid**(void) const**

Is this **OMReferenceVectorIterator** validly positioned on a *ReferencedObject* ?

virtual size_t count**(void) const**

The number of *ReferencedObject*s in the associated OMReferenceVector.

virtual bool operator++**()**

Advance this **OMReferenceVectorIterator** to the next *ReferencedObject*, if any. If the end of the associated OMReferenceVector is not reached then the result is **true** , **valid** becomes **true** and **after** becomes **false** . If the end of the associated OMReferenceVector is reached then the result is **false** , **valid** becomes **false** and **after** becomes **true** .

virtual bool operator--**()**

Retreat this **OMReferenceVectorIterator** to the previous *ReferencedObject*, if any. If the beginning of the associated OMReferenceVector is not reached then the result is **true** , **valid** becomes **true** and **before** becomes **false** . If the beginning of the associated OMReferenceVector is reached then the result is **false** , **valid** becomes **false** and **before** becomes **true** .

virtual ReferencedObject* value**(void) const**

Return the *ReferencedObject* in the associated OMReferenceVector at the position currently designated by this **OMReferenceVectorIterator.**

virtual ReferencedObject* setValue**(const ReferencedObject* newObject)**

Set the *ReferencedObject* in the associated OMReferenceVector at the position currently designated by this **OMReferenceVectorIterator** to *newObject*. The previous *ReferencedObject*, if any, is returned.

virtual OMObject* currentObject**(void) const**

Return the *OMObject* in the associated reference container at the position currently designated by this **OMReferenceVectorIterator.**

virtual size_t index**(void) const**

Return the index of the *ReferencedObject* in the associated OMReferenceVector at the position currently designated by this **OMReferenceVectorIterator.**

OMReferenceVectorIterator**(const VectorIterator& iter)**

Create an **OMReferenceVectorIterator** given an underlying OMVectorIterator.

OMReferenceVectorIterator::after

template <class *ReferencedObject***>**
bool OMReferenceVectorIterator<*ReferencedObject***>::after(void) const**

Is this OMReferenceVectorIterator positioned after the last *ReferencedObject* ?

Defined in: OMReferenceVectorIterT.h

Return Value

true if this OMReferenceVectorIterator is positioned after the last *ReferencedObject*, **false** otherwise.

Class Template Arguments

ReferencedObject
The type of the contained objects.

Back to OMReferenceVectorIterator

OMReferenceVectorIterator::before

template <class *ReferencedObject***>**
bool OMReferenceVectorIterator<*ReferencedObject***>::before(void) const**

Is this OMReferenceVectorIterator positioned before the first *ReferencedObject* ?

Defined in: OMReferenceVectorIterT.h

Return Value

true if this OMReferenceVectorIterator is positioned before the first *ReferencedObject*, **false** otherwise.

Class Template Arguments

ReferencedObject
The type of the contained objects.

Back to OMReferenceVectorIterator

OMReferenceVectorIterator::copy

template <class *ReferencedObject***>**
OMReferenceContainerIterator* OMReferenceVectorIterator<*ReferencedObject***>::copy(void) const**

Create a copy of this OMReferenceVectorIterator.

Defined in: OMReferenceVectorIterT.h

Return Value

The new OMReferenceVectorIterator.

Class Template Arguments

ReferencedObject
The type of the contained objects.

Back to OMReferenceVectorIterator

OMReferenceVectorIterator::count

template <class *ReferencedObject***>**
size_t OMReferenceVectorIterator<*ReferencedObject***>::count(void) const**

The number of *ReferencedObject*s in the associated OMReferenceVector.

Defined in: OMReferenceVectorIterT.h

Return Value

The number of *ReferencedObject*s

Class Template Arguments

ReferencedObject
The type of the contained objects.

Back to OMReferenceVectorIterator

OMReferenceVectorIterator::currentObject

template <class *ReferencedObject***>**
OMObject* OMReferenceVectorIterator<*ReferencedObject***>::currentObject(void) const**

Return the *OMObject* in the associated reference container at the position currently designated by this OMReferenceVectorIterator.

Defined in: OMReferenceVectorIterT.h

Return Value

The OMObject at the current position.

Class Template Arguments

ReferencedObject
The type of the contained objects.

Back to OMReferenceVectorIterator

OMReferenceVectorIterator::index

template <class *Element***>**
size_t OMReferenceVectorIterator<*Element***>::index(void) const**

Return the index of the *ReferencedObject* in the associated OMReferenceVector at the position currently designated by this OMReferenceVectorIterator.

Defined in: OMReferenceVectorIterT.h

Return Value

The index of the current position.

Class Template Arguments

Element
The type of the contained elements.

Back to OMReferenceVectorIterator

OMReferenceVectorIterator::OMReferenceVectorIterator

template <class *ReferencedObject***>**
OMReferenceVectorIterator<*ReferencedObject***>::OMReferenceVectorIterator(const VectorIterator &** *iter***)**

Create an OMReferenceVectorIterator given an underlying OMVectorIterator.

Defined in: OMReferenceVectorIterT.h

Parameters

iter
The underlying OMVectorIterator.

Class Template Arguments

ReferencedObject
The type of the contained objects.

Back to OMReferenceVectorIterator

OMReferenceVectorIterator::OMReferenceVectorIterator

template <class *ReferencedObject***>**
OMReferenceVectorIterator<*ReferencedObject***>::OMReferenceVectorIterator(const OMReferenceVector&ltReferencedObject>&** *vector***, OMIteratorPosition** *initialPosition***)**

Create an OMReferenceVectorIterator over the given OMReferenceVector *vector* and initialize it to the given *initialPosition*. If *initialPosition* is specified as OMBefore then this OMReferenceVectorIterator is made ready to traverse the associated OMReferenceVector in the forward direction (increasing indexes). If *initialPosition* is specified as OMAfter then this OMReferenceVectorIterator is made ready to traverse the associated OMReferenceVector in the reverse direction (decreasing indexes).

Defined in: OMReferenceVectorIterT.h

Parameters

vector
The OMStrongReferenceVector over which to iterate.
initialPosition
The initial position for this OMReferenceVectorIterator.

Class Template Arguments

ReferencedObject
The type of the contained objects.
Back to OMReferenceVectorIterator

OMReferenceVectorIterator::operator++

template <class *ReferencedObject***>**
bool OMReferenceVectorIterator<*ReferencedObject***>::operator++(void)**

Advance this OMReferenceVectorIterator to the next *ReferencedObject*, if any. If the end of the associated OMReferenceVector is not reached then the result is **true** , **valid** becomes **true** and **after** becomes **false** . If the end of the associated OMReferenceVector is reached then the result is **false** , **valid** becomes **false** and **after** becomes **true** .

Defined in: OMReferenceVectorIterT.h

Return Value

false if this OMReferenceVectorIterator has passed the last *ReferencedObject*, **true** otherwise.

Class Template Arguments

ReferencedObject
The type of the contained objects.
Back to OMReferenceVectorIterator

OMReferenceVectorIterator::operator--

template <class *ReferencedObject***>**
bool OMReferenceVectorIterator<*ReferencedObject***>::operator--(void)**

Retreat this OMReferenceVectorIterator to the previous *ReferencedObject*, if any. If the beginning of the associated OMReferenceVector is not reached then the result is **true** , **valid** becomes **true** and **before** becomes **false** . If the beginning of the associated OMReferenceVector is reached then the result is **false** , **valid** becomes **false** and **before** becomes **true** .

Defined in: OMReferenceVectorIterT.h

Return Value

false if this OMReferenceVectorIterator has passed the first *ReferencedObject*, **true** otherwise.

Class Template Arguments

ReferencedObject
The type of the contained objects.

Back to OMReferenceVectorIterator

OMReferenceVectorIterator::reset

template <class *ReferencedObject***>**
void OMReferenceVectorIterator<*ReferencedObject***>::reset(OMIteratorPosition** *initialPosition***)**

Reset this OMReferenceVectorIterator to the given *initialPosition*. If *initialPosition* is specified as OMBefore then this OMReferenceVectorIterator is made ready to traverse the associated OMReferenceVector in the forward direction (increasing indexes). If *initialPosition* is specified as OMAfter then this OMReferenceVectorIterator is made ready to traverse the associated OMReferenceVector in the reverse direction (decreasing indexes).

Defined in: OMReferenceVectorIterT.h

Parameters

initialPosition
The position to which this OMReferenceVectorIterator should be reset.

Class Template Arguments

ReferencedObject
The type of the contained objects.

Back to OMReferenceVectorIterator

OMReferenceVectorIterator::setValue

template <class *ReferencedObject***>**
ReferencedObject* OMReferenceVectorIterator<*ReferencedObject***>::setValue(const ReferencedObject*** *newObject***)**

Set the *ReferencedObject* in the associated OMReferenceVector at the position currently designated by this OMReferenceVectorIterator to *newObject*. The previous *ReferencedObject*, if any, is returned.

Defined in: OMReferenceVectorIterT.h

Return Value

The previous *ReferencedObject* if any, otherwise 0.

Parameters

newObject
The new *ReferencedObject*.

Class Template Arguments

ReferencedObject
The type of the contained objects.

Back to OMReferenceVectorIterator

OMReferenceVectorIterator::valid

template <class *ReferencedObject***>**
bool OMReferenceVectorIterator<*ReferencedObject***>::valid(void) const**

Is this OMReferenceVectorIterator validly positioned on a *ReferencedObject* ?

Defined in: OMReferenceVectorIterT.h

Return Value

true if this OMReferenceVectorIterator is positioned on a *ReferencedObject*, **false** otherwise.

Class Template Arguments

ReferencedObject
The type of the contained objects.

Back to OMReferenceVectorIterator

OMReferenceVectorIterator::value

template <class *ReferencedObject***>**
ReferencedObject* OMReferenceVectorIterator<*ReferencedObject***>::value(void) const**

Return the *ReferencedObject* in the associated OMReferenceVector at the position currently designated by this OMReferenceVectorIterator.

Defined in: OMReferenceVectorIterT.h

Return Value

The *ReferencedObject* at the current position.

Class Template Arguments

ReferencedObject
The type of the contained objects.

Back to OMReferenceVectorIterator

OMReferenceVectorIterator::~OMReferenceVectorIterator

template <class *ReferencedObject***>**
OMReferenceVectorIterator<*ReferencedObject***>::~OMReferenceVectorIterator(void)**

Destroy this OMReferenceVectorIterator.

Defined in: OMReferenceVectorIterT.h

Class Template Arguments

ReferencedObject
The type of the contained objects.

Back to OMReferenceVectorIterator

OMReferenceVectorProperty class

OMReferenceVectorProperty **class OMReferenceVectorProperty: public OMContainerProperty, public OMObjectVector**

Abstract base class for persistent object reference vector properties supported by the Object Manager.

Defined in: OMRefVectorProperty.h

Author

Tim Bingham - tjb - (Avid Technology, Inc.)

Class Members

Public members.

OMReferenceVectorProperty**(const OMPropertyId propertyId, const OMStoredForm storedForm, const wchar_t* name)**

Constructor.

virtual ~OMReferenceVectorProperty**(void)**

Destructor.

virtual OMReferenceContainer* referenceContainer**(void)**

Convert to OMReferenceContainer.

OMReferenceVectorProperty::OMReferenceVectorProperty

OMReferenceVectorProperty::OMReferenceVectorProperty(const OMPropertyId *propertyId*, **const OMStoredForm** *storedForm*, **const wchar_t*** *name***)**

Constructor.

Defined in: OMRefVectorProperty.cpp

Parameters

propertyId

The property id.

storedForm

The stored form of this property.

name

The name of this property.

Back to OMReferenceVectorProperty

OMReferenceVectorProperty::referenceContainer

OMReferenceContainer* OMReferenceVectorProperty::referenceContainer(void)

Convert to OMReferenceContainer.

Defined in: OMRefVectorProperty.cpp

Return Value

The OMReferenceContainer interface implemented by this OMReferenceVectorProperty

Back to OMReferenceVectorProperty

OMReferenceVectorProperty::~OMReferenceVectorProperty

OMReferenceVectorProperty::~OMReferenceVectorProperty(void)

Destructor.

Defined in: OMRefVectorProperty.cpp

Back to OMReferenceVectorProperty

OMRootStorable class

OMRootStorable **class OMRootStorable: public OMStorable**

Concrete sub-class of the abstract OMStorable for use as the root object in a file that may be managed by the Object Manager.

Defined in: OMRootStorable.h

Author

Tim Bingham - tjb - (Avid Technology, Inc.)

Class Members

virtual void save**(void) const**
Save this **OMRootStorable.**
virtual void close**(void)**
Close this **OMRootStorable.**
virtual void restoreContents**(void)**
Restore the contents of an **OMRootStorable.**

OMRootStorable::close

void OMRootStorable::close(void)

Close this OMRootStorable.

Defined in: OMRootStorable.cpp

Back to OMRootStorable

OMRootStorable::restoreContents

void OMRootStorable::restoreContents(void)

Restore the contents of an OMRootStorable.

Defined in: OMRootStorable.cpp

Back to OMRootStorable

OMRootStorable::save

void OMRootStorable::save(void)

Save this OMRootStorable.

Defined in: OMRootStorable.cpp

Back to OMRootStorable

OMSet class

OMSet **class OMSet: public OMContainer**

Sets of uniquely identified elements. Duplicate elements are not allowed.

Defined in: OMSet.h

Class Template Arguments

Element
: The type of an **OMSet** element. This type must support operator = and operator==.

Key
: The type of the unique key used to identify elements. This type must support operator =, operator != and operator

Author

Tim Bingham - tjb - (Avid Technology, Inc.)

Class Members

Public members.

OMSet()
: Constructor.

virtual ~OMSet(void)
: Destructor.

virtual void insert**(const Key, const Element value)**
: Insert *value* into this **OMSet**.

virtual bool contains**(const Key key) const**
: Does this **OMSet** contain an *Element* identified by *key*?

virtual bool find**(const Key key, Element& value) const**
: Find the *Element* in this **OMSet** identified by *key*. If the element is found it is returned in *value* and the result is true. If the element is not found the result is false.

virtual bool find**(const Key key, Element** value) const**

Find the *Element* in this **OMSet** identified by *key*. If the element is found it is returned in *value* and the result is true. If the element is not found the result is false.

size_t count**(void) const**

The number of elements in this **OMSet. count** returns the actual number of elements in the **OMSet.**

void append**(const Key, const Element value)**

Append the given *Element value* to this **OMSet.**

virtual void remove**(const Key key)**

Remove the *Element* with *Key key* from this **OMSet.**

virtual void clear**(void)**

Remove all elements from this **OMSet.**

Class Members

Private members.

OMSet::append

template <class *Element*, **class** *Key***>**
void OMSet<*Element*, *Key***>::append(const Key** *key*, **const Element** *value***)**

Append the given *Element value* to this OMSet.

Defined in: OMSetT.h

Parameters

key

The *Key*.

value

The *Element* to append.

Class Template Arguments

Element

The type of an OMSet element. This type must support operator = and operator==.

Key

The type of the unique key used to identify elements. This type must support operator =, operator != and operator

Back to OMSet

OMSet::clear

template <class *Element*, **class** *Key***>**
void OMSet<*Element*, *Key***>::clear(** *void***)**

Remove all elements from this OMSet.

Defined in: OMSetT.h

Parameters

void
: The *Element* to remove.

Class Template Arguments

Element
: The type of an OMSet element. This type must support operator = and operator==. Instances of this type must be able to return a unique value of type *Key* to identify themselves through a function with the signature const Key Element::identification(void) const.

Key
: The type of the unique key used to identify elements. This type must support operator =, operator != and operator

Back to OMSet

OMSet::contains

template <class *Element*, **class** *Key***>**
bool OMSet<*Element*, *Key***>::contains(const Key** *key***) const**

Does this OMSet contain an *Element* identified by *key*?

Defined in: OMSetT.h

Return Value

True if this OMSet contains an *Element* identified by *key*, false otherwise.

Parameters

key
: The *Key* for which to search.

Class Template Arguments

Element
: The type of an OMSet element. This type must support operator = and operator==.

Key
: The type of the unique key used to identify elements. This type must support operator =, operator != and operator

Back to OMSet

OMSet::count

template <class *Element*, **class** *Key***>**
size_t OMSet<*Element*, *Key***>::count(void) const**

The number of elements in this OMSet. **count** returns the actual number of elements in the OMSet.

Defined in: OMSetT.h

Return Value

The count of elements in this OMSet.

Class Template Arguments

Element
: The type of an OMSet element. This type must support operator = and operator==.

Key
: The type of the unique key used to identify elements. This type must support operator =, operator != and operator

Back to OMSet

OMSet::find

template <class *Element*, **class** *Key***>**
bool OMSet<*Element*, *Key***>::find(const Key** *key*, **Element**** *value***) const**

Find the *Element* in this OMSet identified by *key*. If the element is found it is returned in *value* and the result is true. If the element is not found the result is false.

Defined in: OMSetT.h

Return Value

True if this OMSet contains an *Element* identified by *key*, false otherwise.

Parameters

key
: The *Key* for which to search.

value
: The value that was found, if any, by pointer.

Class Template Arguments

Element
: The type of an OMSet element. This type must support operator = and operator==.

Key
: The type of the unique key used to identify elements. This type must support operator =, operator != and operator

Back to OMSet

OMSet::find

template <class *Element*, **class** *Key***>**
bool OMSet<*Element*, *Key***>::find(const Key** *key*, **Element&** *value***) const**

Find the *Element* in this OMSet identified by *key*. If the element is found it is returned in *value* and the result is true. If the element is not found the result is false.

Defined in: OMSetT.h

Return Value

True if this OMSet contains an *Element* identified by *key*, false otherwise.

Parameters

key
: The *Key* for which to search.

value
: The value that was found, if any, by reference.

Class Template Arguments

Element
: The type of an OMSet element. This type must support operator = and operator==.

Key
: The type of the unique key used to identify elements. This type must support operator =, operator != and operator

Back to OMSet

OMSet::insert

template <class *Element*, **class** *Key***>**
void OMSet<*Element*, *Key***>::insert(const Key** *key*, **const Element** *value***)**

Insert *value* into this OMSet.

Defined in: OMSetT.h

Parameters

key
The *Key*.
value
The *Element* to insert.

Class Template Arguments

Element
The type of an OMSet element. This type must support operator = and operator==.
Key
The type of the unique key used to identify elements. This type must support operator =, operator != and operator
Back to OMSet

OMSet::remove

template <class *Element*, **class** *Key***>**
void OMSet<*Element*, *Key***>::remove(const Key** *key***)**

Remove the *Element* with *Key key* from this OMSet.

Defined in: OMSetT.h

Parameters

key
The *Key* of the *Element* to remove.

Class Template Arguments

Element
The type of an OMSet element. This type must support operator = and operator==.
Key
The type of the unique key used to identify elements. This type must support operator =, operator != and operator
Back to OMSet

OMSetElement class

OMSetElement **class OMSetElement**

Pointer elements of non-persistent Object Manager sets.

Defined in: OMContainerElement.h

Class Template Arguments

ReferencedObject
The type of the referenced object.

Author

Tim Bingham - tjb - (Avid Technology, Inc.)

Class Members

Public members.
OMSetElement**(void)**
Constructor.
OMSetElement**(const ReferencedObject* pointer)**
Constructor.
OMSetElement**(const OMSetElement&ltUniqueIdentification, ReferencedObject>& rhs)**
Copy constructor.
~OMSetElement**(void)**
Destructor.
OMSetElement&ltUniqueIdentification, ReferencedObject>& operator=**(const OMSetElement&ltUniqueIdentification, ReferencedObject>& rhs)**
Assignment. This operator provides value semantics for OMSet**.**
bool operator==**(const OMSetElement&ltUniqueIdentification, ReferencedObject>& rhs) const**
Equality. This operator provides value semantics for OMSet**.**
UniqueIdentification identification**(void) const**
The unique key of this **OMSetElement.**

OMSetElement::identification

template <class *ReferencedObject***>**
UniqueIdentification OMSetElement<*ReferencedObject***>::identification(void)**

The unique key of this OMSetElement.

Defined in: OMContainerElementT.h

Return Value

The unique key of this OMSetElement**.**

Class Template Arguments

ReferencedObject
The type of the referenced object.
Back to OMSetElement

OMSetElement::OMSetElement

template <class *ReferencedObject***>**
OMSetElement<*ReferencedObject***>::OMSetElement(void)**

Constructor.

Defined in: OMContainerElementT.h

Class Template Arguments

ReferencedObject
The type of the referenced object.

Back to OMSetElement

OMSetElement::OMSetElement

template <class *ReferencedObject***>**
OMSetElement<*ReferencedObject***>::OMSetElement(const OMSetElement<** *UniqueIdentification***)**

Copy constructor.

Defined in: OMContainerElementT.h

Parameters

UniqueIdentification
The OMSetElement to copy.

Class Template Arguments

ReferencedObject
The type of the referenced object.

Back to OMSetElement

OMSetElement::OMSetElement

template <class *ReferencedObject***>**
OMSetElement<*ReferencedObject***>::OMSetElement(const ReferencedObject*** *pointer***)**

Constructor.

Defined in: OMContainerElementT.h

Parameters

pointer
A pointer to a *ReferencedObject*.

Class Template Arguments

ReferencedObject
The type of the referenced object.

Back to OMSetElement

OMSetElement::operator=

template <class *ObjectReference*, **class** *ReferencedObject***>**
OMSetElement&ltUniqueIdentification, ReferencedObject>& OMSetElement<*ObjectReference*, *ReferencedObject***>::operator=(const OMSetElement<** *UniqueIdentification***)**

Assignment. This operator provides value semantics for OMSet.

Defined in: OMContainerElementT.h

Return Value

The OMSetElement resulting from the assignment.

Parameters

UniqueIdentification
The OMSetElement to be assigned.

Class Template Arguments

ObjectReference
The type of the contained object reference

ReferencedObject
The type of the referenced object.

Back to OMSetElement

OMSetElement::operator==

template <class *ReferencedObject***>**
bool OMSetElement<*ReferencedObject***>::operator==(const OMSetElement<** *UniqueIdentification***)**

Equality. This operator provides value semantics for OMSet.

Defined in: OMContainerElementT.h

Return Value

True if the values are the same, false otherwise.

Parameters

UniqueIdentification
The OMSetElement to be compared.

Class Template Arguments

ReferencedObject
The type of the referenced object.

Back to OMSetElement

OMSetElement::~OMSetElement

template <class *ReferencedObject***>**
OMSetElement<*ReferencedObject***>::~OMSetElement(void)**

Destructor.

Defined in: OMContainerElementT.h

Class Template Arguments

ReferencedObject
The type of the referenced object.

Back to OMSetElement

OMSetIterator class

OMSetIterator **class OMSetIterator: public OMContainerIterator**

Iterators over OMSets.

Defined in: OMSetIterator.h

Class Template Arguments

Key
The type of the unique key that identifies the contained elements.

Element
The type of the contained elements.

Author

Tim Bingham - tjb - (Avid Technology, Inc.)

Class Members

Public members.

OMSetIterator**(const OMSet&ltKey, Element>& set, OMIteratorPosition initialPosition)**

Create an **OMSetIterator** over the given OMSet *set* and initialize it to the given *initialPosition*. If *initialPosition* is specified as OMBefore then this **OMSetIterator** is made ready to traverse the associated OMSet in the forward direction (increasing *Key*s). If *initialPosition* is specified as OMAfter then this **OMSetIterator** is made ready to traverse the associated OMSet in the reverse direction (decreasing *Key*s).

virtual ~OMSetIterator**(void)**

Destroy this **OMSetIterator**.

virtual void reset**(OMIteratorPosition initialPosition)**

Reset this **OMSetIterator** to the given *initialPosition*. If *initialPosition* is specified as OMBefore then this **OMSetIterator** is made ready to traverse the associated OMSet in the forward direction (increasing *Key*s). If *initialPosition* is specified as OMAfter then this **OMSetIterator** is made ready to traverse the associated OMSet in the reverse direction (decreasing *Key*s).

virtual bool before**(void) const**

Is this **OMSetIterator** positioned before the first *Element* ?

virtual bool after**(void) const**

Is this **OMSetIterator** positioned after the last *Element* ?

virtual size_t count**(void) const**

The number of *Elements*s in the associated OMSet.

virtual bool operator++**()**

Advance this **OMSetIterator** to the next *Element*, if any. If the end of the associated OMSet is not reached then the result is **true** , **valid** becomes **true** and **after** becomes **false** . If the end of the associated OMSet is reached then the result is **false** , **valid** becomes **false** and **after** becomes **true** .

virtual bool operator--**()**

Retreat this **OMSetIterator** to the previous *Element*, if any. If the beginning of the associated OMSet is not reached then the result is **true** , **valid** becomes **true** and **before** becomes **false** . If the beginning of the associated OMSet is reached then the result is **false** , **valid** becomes **false** and **before** becomes **true** .

virtual Element& value**(void) const**

Return the *Element* in the associated OMSet at the position currently designated by this **OMSetIterator.**

virtual Element setValue**(const Key k, Element newElement)**

Set the *Element* in the associated OMSet at the position currently designated by this **OMSetIterator** to *newElement*. The previous *Element* is returned. To preserve the ordering of *Key*s, the *Key* of *newElement* must be the same as that of the existing *Element*.

Key key**(void) const**

Return the *Key* of the *Element* in the associated OMSet at the position currently designated by this **OMSetIterator.**

OMSetIterator::after

template <class *Key*, **class** *Element***>**
bool OMSetIterator<*Key*, *Element***>::after(void) const**

Is this OMSetIterator positioned after the last *Element* ?

Defined in: OMSetIteratorT.h

Return Value

true if this OMSetIterator is positioned after the last *Element*, **false** otherwise.

Class Template Arguments

Key
The type of the unique key that identifies the contained elements.
Element
The type of the contained elements.
Back to OMSetIterator

OMSetIterator::before

template <class *Key*, **class** *Element***>**
bool OMSetIterator<*Key*, *Element***>::before(void) const**

Is this OMSetIterator positioned before the first *Element* ?

Defined in: OMSetIteratorT.h

Return Value

true if this OMSetIterator is positioned before the first *Element*, **false** otherwise.

Class Template Arguments

Key
The type of the unique key that identifies the contained elements.
Element
The type of the contained elements.
Back to OMSetIterator

OMSetIterator::count

template <class *Key*, **class** *Element***>**
size_t OMSetIterator<*Key*, *Element***>::count(void) const**

The number of *Elementss* in the associated OMSet.

Defined in: OMSetIteratorT.h

Return Value

The number of *Elements*s

Class Template Arguments

Key
: The type of the unique key that identifies the contained elements.

Element
: The type of the contained elements.

Back to OMSetIterator

OMSetIterator::key

template <class *Key*, **class** *Element***>**
Key OMSetIterator<*Key*, *Element***>::key(void) const**

Return the *Key* of the *Element* in the associated OMSet at the position currently designated by this OMSetIterator.

Defined in: OMSetIteratorT.h

Return Value

The *Key* at the current position.

Class Template Arguments

Key
: The type of the unique key that identifies the contained elements.

Element
: The type of the contained elements.

Back to OMSetIterator

OMSetIterator::OMSetIterator

template <class *Key*, **class** *Element***>**
OMSetIterator<*Key*, *Element***>::OMSetIterator(const OMSet&ltKey, Element>&** *set*, **OMIteratorPosition** *initialPosition***)**

Create an OMSetIterator over the given OMSet *set* and initialize it to the given *initialPosition*. If *initialPosition* is specified as OMBefore then this OMSetIterator is made ready to traverse the associated OMSet in the forward direction (increasing *Key*s). If *initialPosition* is specified as OMAfter then this OMSetIterator is made ready to traverse the associated OMSet in the reverse direction (decreasing *Key*s).

Defined in: OMSetIteratorT.h

Parameters

set

The OMSet over which to iterate.

initialPosition

The initial position for this OMSetIterator.

Class Template Arguments

Key

The type of the unique key that identifies the contained elements.

Element

The type of the contained elements.

Back to OMSetIterator

OMSetIterator::operator++

template <class *Key*, **class** *Element***>**
bool OMSetIterator<*Key*, *Element***>::operator++(void)**

Advance this OMSetIterator to the next *Element*, if any. If the end of the associated OMSet is not reached then the result is **true** , **valid** becomes **true** and **after** becomes **false** . If the end of the associated OMSet is reached then the result is **false** , **valid** becomes **false** and **after** becomes **true** .

Defined in: OMSetIteratorT.h

Return Value

false if this OMSetIterator has passed the last *Element*, **true** otherwise.

Class Template Arguments

Key

The type of the unique key that identifies the contained elements.

Element

The type of the contained elements.

Back to OMSetIterator

OMSetIterator::operator--

template <class *Key*, **class** *Element***>**
bool OMSetIterator<*Key*, *Element***>::operator--(void)**

Retreat this OMSetIterator to the previous *Element*, if any. If the beginning of the associated OMSet is not reached then the result is **true** , **valid** becomes **true** and **before** becomes **false** . If the beginning of the associated OMSet is reached then the result is **false** , **valid** becomes **false** and **before** becomes **true** .

Defined in: OMSetIteratorT.h

Return Value

false if this OMSetIterator has passed the first *Element*, **true** otherwise.

Class Template Arguments

Key
: The type of the unique key that identifies the contained elements.

Element
: The type of the contained elements.

Back to OMSetIterator

OMSetIterator::reset

template <class *Key*, **class** *Element***>**
void OMSetIterator<*Key*, *Element***>::reset(OMIteratorPosition** *initialPosition***)**

Reset this OMSetIterator to the given *initialPosition*. If *initialPosition* is specified as OMBefore then this OMSetIterator is made ready to traverse the associated OMSet in the forward direction (increasing *Key*s). If *initialPosition* is specified as OMAfter then this OMSetIterator is made ready to traverse the associated OMSet in the reverse direction (decreasing *Key*s).

Defined in: OMSetIteratorT.h

Parameters

initialPosition
: The position to which this OMSetIterator should be reset.

Class Template Arguments

Key
: The type of the unique key that identifies the contained elements.

Element
: The type of the contained elements.

Back to OMSetIterator

OMSetIterator::setValue

template <class *Key*, **class** *Element***>**
Element OMSetIterator<*Key*, *Element***>::setValue(const Key** *k*, **Element** *newElement***)**

Set the *Element* in the associated OMSet at the position currently designated by this OMSetIterator to *newElement*. The previous *Element* is returned. To preserve the ordering of *Key*s, the *Key* of *newElement* must be the same as that of the existing *Element*.

Defined in: OMSetIteratorT.h

Return Value

The previous *Element*.

Parameters

k
: The *Key*.

newElement
: The new *Element*.

Class Template Arguments

Key
: The type of the unique key that identifies the contained elements.

Element
: The type of the contained elements.

Back to OMSetIterator

OMSetIterator::value

template <class *Key*, **class** *Element***>**
Element& OMSetIterator<*Key*, *Element***>::value(void) const**

Return the *Element* in the associated OMSet at the position currently designated by this OMSetIterator.

Defined in: OMSetIteratorT.h

Return Value

The *Element* at the current position.

Class Template Arguments

Key
: The type of the unique key that identifies the contained elements.

Element
: The type of the contained elements.

Back to OMSetIterator

OMSetIterator::~OMSetIterator

template <class *Key*, **class** *Element***>**
OMSetIterator<*Key*, *Element***>::~OMSetIterator(void)**

Destroy this OMSetIterator.

Defined in: OMSetIteratorT.h

Class Template Arguments

Key
The type of the unique key that identifies the contained elements.
Element
The type of the contained elements.
Back to OMSetIterator

OMSimpleProperty class

OMSimpleProperty **class OMSimpleProperty: public OMProperty**

Abstract base class for simple (data) persistent properties supported by the Object Manager.

Defined in: OMProperty.h

Author

Tim Bingham - tjb - (Avid Technology, Inc.)

Class Members

Public members.
OMSimpleProperty**(const OMPropertyId propertyId, const wchar_t* name, size_t valueSize)**
Constructor.
OMSimpleProperty**(const OMPropertyId propertyId, const wchar_t* name)**
Constructor.
virtual ~OMSimpleProperty**(void)**
Destructor.
virtual void save**(void) const**
Save this **OMSimpleProperty.**
virtual void restore**(size_t externalSize)**
Restore this **OMSimpleProperty**, the external (persisted) size of the **OMSimpleProperty** is *externalSize*.
size_t size**(void) const**
The size of this **OMSimpleProperty.**
virtual size_t bitsSize**(void) const**
The size of the raw bits of this **OMSimpleProperty.** The size is given in bytes.
virtual OMByte* bits(void) const
The raw bits of this **OMSimpleProperty.**
virtual void getBits**(OMByte* bits, size_t size) const**
Get the raw bits of this **OMSimpleProperty.** The raw bits are copied to the buffer at address *bits* which is *size* bytes in size.
virtual void setBits**(const OMByte* bits, size_t size)**
Set the raw bits of this **OMSimpleProperty.** The raw bits are copied from the buffer at address *bits* which is *size* bytes in size.
void setSize**(size_t newSize)**
Set the size of this **OMSimpleProperty** to *newSize* bytes.

Class Members

Protected members.
void get**(void* value, size_t valueSize) const**
Get the value of this **OMSimpleProperty.**
void set**(const void* value, size_t valueSize)**
Set the value of this **OMSimpleProperty.**

OMSimpleProperty::bitsSize

size_t OMSimpleProperty::bitsSize(void) const

The size of the raw bits of this OMSimpleProperty. The size is given in bytes.

Defined in: OMProperty.cpp

Return Value

The size of the raw bits of this OMSimpleProperty in bytes.

Back to OMSimpleProperty

OMSimpleProperty::get

void OMSimpleProperty::get(void* *value***, size_t** *valueSize***) const**

Get the value of this OMSimpleProperty.

Defined in: OMProperty.cpp

Parameters

value
The buffer to receive the property value.
valueSize
The size of the buffer.
Back to OMSimpleProperty

OMSimpleProperty::getBits

void OMSimpleProperty::getBits(OMByte* *bits***, size_t** *bitsSize***) const**

Get the raw bits of this OMSimpleProperty. The raw bits are copied to the buffer at address *bits* which is *size* bytes in size.

Defined in: OMProperty.cpp

Parameters

bits
The address of the buffer into which the raw bits are copied.
bitsSize
The size of the buffer.
Back to OMSimpleProperty

OMSimpleProperty::OMSimpleProperty

OMSimpleProperty::OMSimpleProperty(const OMPropertyId *propertyId*, **const wchar_t*** *name*, **size_t** *valueSize***)**

Constructor.

Defined in: OMProperty.cpp

Parameters

propertyId
The property id.
name
The name of this OMSimpleProperty.
valueSize
The size of this OMSimpleProperty.
Back to OMSimpleProperty

OMSimpleProperty::OMSimpleProperty

OMSimpleProperty::OMSimpleProperty(const OMPropertyId *propertyId*, **const wchar_t*** *name***)**

Constructor.

Defined in: OMProperty.cpp

Parameters

propertyId
The property id.
name
The name of this OMSimpleProperty.
Back to OMSimpleProperty

OMSimpleProperty::restore

void OMSimpleProperty::restore(size_t *externalSize***)**

Restore this OMSimpleProperty, the external (persisted) size of the OMSimpleProperty is *externalSize*.

Defined in: OMProperty.cpp

Parameters

externalSize
The size of the OMSimpleProperty.

Back to OMSimpleProperty

OMSimpleProperty::save

void OMSimpleProperty::save(void) const

Save this OMSimpleProperty.

Defined in: OMProperty.cpp

Back to OMSimpleProperty

OMSimpleProperty::set

void OMSimpleProperty::set(const void* *value*, size_t *valueSize*) const

Set the value of this OMSimpleProperty.

Defined in: OMProperty.cpp

Parameters

value
The address of the property value.

valueSize
The size of the value.

Back to OMSimpleProperty

OMSimpleProperty::setBits

void OMSimpleProperty::setBits(const OMByte* *bits*, size_t *size*)

Set the raw bits of this OMSimpleProperty. The raw bits are copied from the buffer at address *bits* which is *size* bytes in size.

Defined in: OMProperty.cpp

Parameters

bits
The address of the buffer from which the raw bits are copied.
size
The size of the buffer.
Back to OMSimpleProperty

OMSimpleProperty::setSize

void OMSimpleProperty::setSize(size_t *newSize*)

Set the size of this OMSimpleProperty to *newSize* bytes.

Defined in: OMProperty.cpp

Parameters

newSize
The new property size in bytes.
Back to OMSimpleProperty

OMSimpleProperty::size

size_t OMSimpleProperty::size(void) const

The size of this OMSimpleProperty.

Defined in: OMProperty.cpp

Return Value

The property size in bytes.

Back to OMSimpleProperty

OMSimpleProperty::~OMSimpleProperty

OMSimpleProperty::~OMSimpleProperty(void)

Destructor.

Defined in: OMProperty.cpp

Back to OMSimpleProperty

OMSingleton class

OMSingleton **class OMSingleton**

Singleton objects.

References ...

[1] "Design Patterns: Elements of Reusable Object-Oriented Software", Erich Gamma, Richard Helm, Ralph Johnson, John Vlissides, 1994 Addison-Wesley, ISBN 0-201-63361-2, Singleton(127)

Defined in: OMSingleton.h

Author

Tim Bingham - tjb - (Avid Technology, Inc.)

OMStorable class

OMStorable **class OMStorable**

Abstract base class for all objects that may be stored by the Object Manager.

Defined in: OMStorable.h

Author

Tim Bingham - tjb - (Avid Technology, Inc.)

Class Members

Public members.
OMStorable(void)
Constructor.
virtual ~OMStorable(void)
Destructor.
virtual const OMClassId& classId(void) const
The stored object identifier for the class of this **OMStorable.**
virtual void setDefinition**(const OMClassDefinition* definition)**
Set the **OMClassDefinition** defining this **OMStorable.**
virtual const OMClassDefinition* definition**(void) const**
The **OMClassDefinition** defining this **OMStorable.**
void attach**(const OMStorable* container, const wchar_t* name)**
Attach this **OMStorable.**
void detach**(void)**
Detach this **OMStorable.**
void setName**(const wchar_t* name)**
Give this **OMStorable** a name.
virtual void save**(void) const**
Save this **OMStorable.**
virtual void close**(void)**

Close this **OMStorable**.

static OMStorable* restoreFrom(const OMStorable* container, const wchar_t* name, OMStoredObject& s)

Restore an **OMStorable** (of unknown sub-class) from the stored representation *s*.

virtual void restoreContents(void)

Restore the contents of an **OMStorable** (of unknown sub-class).

virtual OMFile* file(void) const

The OMFile in which this **OMStorable** has a persistent representation.

const wchar_t* pathName(void) const

The path to this **OMStorable** from the root of the OMFile in which this **OMStorable** resides.

OMStorable* find(const wchar_t* objectName) const

Find the **OMStorable** named *objectName* contained within this **OMStorable**.

OMProperty* findProperty(const wchar_t* propertyName) const

Find the OMProperty named *propertyName* contained within this **OMStorable**.

bool isRoot(void) const

Is this **OMStorable** the root of the object containment hierarchy.

OMStoredObject* store(void) const

The stored representation of this **OMStorable**.

void setStore(OMStoredObject* store)

Inform this **OMStorable** where it should store its persistent representation.

virtual bool attached(void) const

Is this **OMStorable** attached to (owned by) another **OMStorable** ?

virtual bool inFile(void) const

Is this **OMStorable** associated with an OMFile ?

virtual bool persistent(void) const

Is this **OMStorable** a persistent object ? Persistent objects are associated with a persistent store (disk file).

bool isDirty(void) const

Is this **OMStorable** dirty ? A dirty object is one that has been modified since it was last saved or that has never been saved at all.

virtual OMPropertySet* propertySet(void)

This object's property set.

OMClassFactory* classFactory(void) const

The OMClassFactory that created this object.

void setClassFactory(const OMClassFactory* classFactory)

Inform this **OMStorable** of the OMClassFactory that was used to create it.

OMStorable* shallowCopy(void) const

Create a shallow copy of this **OMStorable**. In a shallow copy, contained objects (strong object references) and streams are not copied.

void deepCopyTo(OMStorable* destination, void* clientContext) const

Create a deep copy of this **OMStorable**, attach the copy to *destination*. In a deep copy, contained objects (strong object references) and streams are copied. This function copies the entire object tree rooted at this **OMStorable**. The root object is treated differently than the contained objects in that only the strong references and streams are copied. Clients may choose to create *destination* using **shallowCopy**. All strong reference properties of this **OMStorable** must be present in the property set of *destination*. The values of the strong reference properties of *destination* must be void and are replaced by those of this **OMStorable**. Any properties of *destination* not also in this **OMStorable** are left unchanged.

virtual void onSave(void* clientContext) const

Inform this **OMStorable** that it is about to be saved. The *clientContext* passed is the one that was specified in the currently active call to **save**.

virtual void onRestore(void* clientContext) const

Inform this **OMStorable** that it has just been restored. The *clientContext* passed is the one that was specified in the currently active call to **restore.**

virtual void onCopy**(void* clientContext) const**

Inform this **OMStorable** that it has just been copied by **deepCopyTo**. The *clientContext* passed is the one that was specified in the currently active call to **deepCopyTo.** When **onCopy** is called only the shallow portion of the deep copy to be performed by **deepCopyTo** has been completed. That is, this **OMStorable** contains copies of all of the properties of the souce **OMStorable** except for strong references (contained objects). Those properties wll be copied after **onCopy** returns.

Class Members

Private members. Protected members.

const wchar_t* name**(void) const**

The name of this **OMStorable.**

OMStorable::attach

void OMStorable::attach(const OMStorable* *container***, const wchar_t*** *name***)**

Attach this OMStorable.

Defined in: OMStorable.cpp

Parameters

container

The containining OMStorable.

name

The name to be given to this OMStorable.

Back to OMStorable

OMStorable::attached

bool OMStorable::attached(void) const

Is this OMStorable attached to (owned by) another OMStorable ?

Defined in: OMStorable.cpp

Return Value

True if this OMStorable is attached, false otherwise.

Back to OMStorable

OMStorable::classFactory

OMClassFactory* OMStorable::classFactory(void) const

The OMClassFactory that created this object.

Defined in: OMStorable.cpp

Return Value

The OMClassFactory that created this object.

Back to OMStorable

OMStorable::close

void OMStorable::close(void)

Close this OMStorable.

Defined in: OMStorable.cpp

Back to OMStorable

OMStorable::definition

const OMClassDefinition* OMStorable::definition(void)

The **OMClassDefinition** defining this OMStorable.

Defined in: OMStorable.cpp

Return Value

TBS

Back to OMStorable

OMStorable::detach

void OMStorable::detach(void)

Detach this OMStorable.

Defined in: OMStorable.cpp

Back to OMStorable

OMStorable::file

OMFile* OMStorable::file(void) const

The OMFile in which this OMStorable has a persistent representation.

Defined in: OMStorable.cpp

Return Value

The OMFile in which this OMStorable has a persistent representation.

Back to OMStorable

OMStorable::find

OMStorable* OMStorable::find(const wchar_t* *objectName***) const**

Find the OMStorable named *objectName* contained within this OMStorable.

Defined in: OMStorable.cpp

Return Value

The object.

Parameters

objectName
The name of the object.

Back to OMStorable

OMStorable::findProperty

OMProperty* OMStorable::findProperty(const wchar_t* *propertyName***) const**

Find the OMProperty named *propertyName* contained within this OMStorable.

Defined in: OMStorable.cpp

Return Value

The property.

Parameters

propertyName
The name of the property.

Back to OMStorable

OMStorable::inFile

bool OMStorable::inFile(void) const

Is this OMStorable associated with an OMFile ?

Defined in: OMStorable.cpp

Return Value

True if this OMStorable is associated with an OMFile , false otherwise.

Back to OMStorable

OMStorable::isDirty

bool OMStorable::isDirty(void)

Is this OMStorable dirty ? A dirty object is one that has been modified since it was last saved or that has never been saved at all.

Defined in: OMStorable.cpp

Back to OMStorable

OMStorable::isRoot

bool OMStorable::isRoot(void) const

Is this OMStorable the root of the object containment hierarchy.

Defined in: OMStorable.cpp

Return Value

True if this is the root object, false otherwise.

Back to OMStorable

OMStorable::name

const wchar_t* OMStorable::name(void) const

The name of this OMStorable.

Defined in: OMStorable.cpp

Return Value

The name of this OMStorable.

Back to OMStorable

OMStorable::onCopy

void OMStorable::onCopy(void * *clientContext***) const**

Inform this OMStorable that it has just been copied by **deepCopyTo**. The *clientContext* passed is the one that was specified in the currently active call to **deepCopyTo**. When **onCopy** is called only the shallow portion of the deep copy to be performed by **deepCopyTo** has been completed. That is, this OMStorable contains copies of all of the properties of the souce OMStorable except for strong references (contained objects). Those properties wll be copied after **onCopy** returns.

Defined in: OMStorable.cpp

Parameters

clientContext
A context for the client.

Back to OMStorable

OMStorable::onRestore

void OMStorable::onRestore(void * *clientContext***) const**

Inform this OMStorable that it has just been restored. The *clientContext* passed is the one that was specified in the currently active call to **restore**.

Defined in: OMStorable.cpp

Parameters

clientContext
A context for the client.

Back to OMStorable

OMStorable::onSave

void OMStorable::onSave(void * *clientContext*) const

Inform this OMStorable that it is about to be saved. The *clientContext* passed is the one that was specified in the currently active call to **save**.

Defined in: OMStorable.cpp

Parameters

clientContext
A context for the client.

Back to OMStorable

OMStorable::pathName

const wchar_t* OMStorable::pathName(void) const

The path to this OMStorable from the root of the OMFile in which this OMStorable resides.

Defined in: OMStorable.cpp

Return Value

The path name of this OMStorable.

Back to OMStorable

OMStorable::persistent

bool OMStorable::persistent(void) const

Is this OMStorable a persistent object ? Persistent objects are associated with a persistent store (disk file).

Defined in: OMStorable.cpp

Return Value

True if this OMStorable is persistent, false otherwise.

Back to OMStorable

OMStorable::propertySet

OMPropertySet* OMStorable::propertySet(void)

This object's property set.

Defined in: OMStorable.cpp

Return Value

A pointer to this object's OMPropertySet.

Back to OMStorable

OMStorable::restoreContents

void OMStorable::restoreContents(void)

Restore the contents of an OMStorable (of unknown sub-class).

Defined in: OMStorable.cpp

Back to OMStorable

OMStorable::restoreFrom

OMStorable* OMStorable::restoreFrom(const OMStorable* *containingObject*, **const wchar_t*** *name*, **OMStoredObject&** *s***)**

Restore an OMStorable (of unknown sub-class) from the stored representation *s*.

Defined in: OMStorable.cpp

Parameters

containingObject
: The OMStorable that will contain (own) the newly restored OMStorable.

name
: The name to be given the newly restored OMStorable.

s
The OMStoredObject from which to restore this OMStorable.
Back to OMStorable

OMStorable::save

void OMStorable::save(void) const

Save this OMStorable.

Defined in: OMStorable.cpp

Back to OMStorable

OMStorable::setClassFactory

void OMStorable::setClassFactory(const OMClassFactory* *classFactory***)**

Inform this OMStorable of the OMClassFactory that was used to create it.

Defined in: OMStorable.cpp

Parameters

classFactory
The OMClassFactory that was used to create this OMStorable.
Back to OMStorable

OMStorable::setDefinition

void OMStorable::setDefinition(const OMClassDefinition* *definition***)**

Set the **OMClassDefinition** defining this OMStorable.

Defined in: OMStorable.cpp

Parameters

definition
TBS
Back to OMStorable

OMStorable::setName

void OMStorable::setName(const wchar_t* *name***)**

Give this OMStorable a name.

Defined in: OMStorable.cpp

Parameters

name
The name to be given to this OMStorable.

Back to OMStorable

OMStorable::setStore

void OMStorable::setStore(OMStoredObject* *store***)**

Inform this OMStorable where it should store its persistent representation.

Defined in: OMStorable.cpp

Parameters

store
The OMStoredObject on which this OMStorable should be persisted.

Back to OMStorable

OMStorable::store

OMStoredObject* OMStorable::store(void) const

The stored representation of this OMStorable.

Defined in: OMStorable.cpp

Return Value

The OMStoredObject representing this OMStorable.

Back to OMStorable

OMStoredObject class

OMStoredObject **class OMStoredObject**

Abstract base class fo the in-memory representation of a persistent object.

Defined in: OMStoredObject.h

Author

Tim Bingham - tjb - (Avid Technology, Inc.)

Class Members

Public members.

virtual ~OMStoredObject**(void)**

Destructor.

virtual OMStoredObject* create(const wchar_t* name)

Create a new **OMStoredObject**, named *name*, contained by this **OMStoredObject.**

Developer Notes

The name argument to this member function doesn't make sense for all derived instances of **OMStoredObject.**

virtual OMStoredObject* open(const wchar_t* name)

Open an exsiting **OMStoredObject**, named *name*, contained by this **OMStoredObject.**

Developer Notes

The name argument to this member function doesn't make sense for all derived instances of **OMStoredObject.**

virtual void close(void)

Close this **OMStoredObject.**

virtual OMByteOrder byteOrder(void) const

The byte order of this **OMStoredObject.**

Developer Notes

This member function doesn't make sense for all derived instances of **OMStoredObject.**

virtual void save(const OMStoredObjectIdentification& id)

Save the **OMStoredObjectIdentification** *id* in this **OMStoredObject.**

virtual void save(const OMPropertySet& properties)

Save the OMPropertySet *properties* in this **OMStoredObject.**

virtual void save(const OMSimpleProperty& property)

Save the OMSimpleProperty *property* in this **OMStoredObject.**

virtual void save(const OMStrongReference& singleton)

Save the OMStrongReference *singleton* in this **OMStoredObject.**

virtual void save(const OMStrongReferenceVector& vector)

Save the OMStrongReferenceVector *vector* in this **OMStoredObject.**

virtual void save(const OMStrongReferenceSet& set)

Save the OMStrongReferenceSet *set* in this **OMStoredObject.**

virtual void save(const OMWeakReference& singleton)

Save the OMWeakReference *singleton* in this **OMStoredObject.**

virtual void save(const OMWeakReferenceVector& vector)

Save the OMWeakReferenceVector *vector* in this **OMStoredObject.**

virtual void save(const OMWeakReferenceSet& set)

Save the OMWeakReferenceSet *set* in this **OMStoredObject.**

virtual void save(const OMPropertyTable* table)

Save the OMPropertyTable *table* in this **OMStoredObject.**

Developer Notes

Does this member function make sense for all derived instances of **OMStoredObject** ?

virtual void save(const OMDataStream& stream)

Save the OMDataStream *stream* in this **OMStoredObject.**

virtual void restore(OMStoredObjectIdentification& id)

Restore the **OMStoredObjectIdentification** of this **OMStoredObject** into *id*.

virtual void restore(OMPropertySet& properties)

Restore the OMPropertySet *properties* into this **OMStoredObject.**

virtual void restore(OMSimpleProperty& property, size_t externalSize)

Restore the OMSimpleProperty *property* into this **OMStoredObject.**

Developer Notes

The externalSize argument to this member function doesn't make sense for all derived instances of **OMStoredObject.**

virtual void restore(OMStrongReference& singleton, size_t externalSize)

Restore the OMStrongReference *singleton* into this **OMStoredObject.**

virtual void restore(OMStrongReferenceVector& vector, size_t externalSize)

Restore the OMStrongReferenceVector *vector* into this **OMStoredObject.**

virtual void restore(OMStrongReferenceSet& set, size_t externalSize)

Restore the OMStrongReferenceSet *set* into this **OMStoredObject.**

virtual void restore(OMWeakReference& singleton, size_t externalSize)

Restore the OMWeakReference *singleton* into this **OMStoredObject.**

virtual void restore(OMWeakReferenceVector& vector, size_t externalSize)

Restore the OMWeakReferenceVector *vector* into this **OMStoredObject.**

virtual void restore(OMWeakReferenceSet& set, size_t externalSize)

Restore the OMWeakReferenceSet *set* into this **OMStoredObject.**

virtual void restore(OMPropertyTable*& table)

Restore the OMPropertyTable in this **OMStoredObject.**

Developer Notes

Does this member function make sense for all derived instances of **OMStoredObject** ?

virtual void restore(OMDataStream& stream, size_t externalSize)

Restore the OMDataStream *stream* into this **OMStoredObject.**

virtual OMStoredStream* openStoredStream(const OMDataStream& property)

Open the OMStoredStream representing the property *stream* contained within this **OMStoredObject.**

virtual OMStoredStream* createStoredStream(const OMDataStream& property)

Create an OMStoredStream representing the property *stream* contained within this **OMStoredObject.**

static wchar_t* streamName(const wchar_t* propertyName, OMPropertyId pid)

Compute the name for a stream.

Developer Notes

This member function doesn't make sense for all derived instances of **OMStoredObject.**

static wchar_t* referenceName(const wchar_t* propertyName, OMPropertyId pid)

Compute the name for an object reference.

Developer Notes

This member function doesn't make sense for all derived instances of **OMStoredObject**.

static wchar_t* collectionName(const wchar_t* propertyName, OMPropertyId pid)

Compute the name for a collection.

Developer Notes

This member function doesn't make sense for all derived instances of **OMStoredObject**.

static wchar_t* elementName(const wchar_t* propertyName, OMPropertyId pid, OMUInt32 localKey)

Compute the name for an object reference that is an element of a collection.

Developer Notes

This member function doesn't make sense for all derived instances of **OMStoredObject**.

Class Members

Protected members.

OMStoredObject::~OMStoredObject

OMStoredObject::~OMStoredObject(void)

Destructor.

Defined in: OMStoredObject.cpp

Back to OMStoredObject

OMStoredPropertySetIndex class

OMStoredPropertySetIndex **class OMStoredPropertySetIndex**

The in-memory representation of the on-disk index for a stored property set.

Defined in: OMStoredPropertySetIndex.h

Author

Tim Bingham - tjb - (Avid Technology, Inc.)

Class Members

Public members.

OMStoredPropertySetIndex(size_t capacity)

Constructor.

~OMStoredPropertySetIndex(void)

Destructor.

void insert**(OMPropertyId propertyId, OMStoredForm storedForm, OMPropertyOffset offset, OMPropertySize length)**

Insert a new property into this **OMStoredPropertySetIndex**. The new property has id *propertyId*. The stored property representation is *storedForm*. The property value occupies *length* bytes starting at offset *offset*.

size_t entries**(void) const**

The number of properties in this **OMStoredPropertySetIndex**.

void iterate**(size_t& context, OMPropertyId& propertyId, OMStoredForm& storedForm, OMPropertyOffset& offset, OMPropertySize& length) const**

Iterate over the properties in this **OMStoredPropertySetIndex**.

bool find**(const OMPropertyId& propertyId, OMStoredForm& storedForm, OMPropertyOffset& offset, OMPropertySize& length) const**

Find the property with property id *propertyId* in this **OMStoredPropertySetIndex**. If found the *storedForm*, *offset* and *length* of the property are returned.

bool isValid**(OMPropertyOffset baseOffset) const**

Is this **OMStoredPropertySetIndex** valid ?

OMStoredPropertySetIndex::entries

size_t OMStoredPropertySetIndex::entries(void) const

The number of properties in this OMStoredPropertySetIndex.

Defined in: OMStoredPropertySetIndex.cpp

Return Value

The number of properties.

Back to OMStoredPropertySetIndex

OMStoredPropertySetIndex::find

bool OMStoredPropertySetIndex::find(const OMPropertyId& *propertyId*, **OMStoredForm&** *storedForm*, **OMPropertyOffset&** *offset*, **OMPropertySize&** *length***) const**

Find the property with property id *propertyId* in this OMStoredPropertySetIndex. If found the *storedForm*, *offset* and *length* of the property are returned.

Defined in: OMStoredPropertySetIndex.cpp

Return Value

True if a property with the given id was found, false otherwise.

Parameters

propertyId
The id of the property to find.
storedForm
The stored form used for the property.
offset
The offset of the property value in bytes.
length
The size of the property value in bytes.
Back to OMStoredPropertySetIndex

OMStoredPropertySetIndex::insert

void OMStoredPropertySetIndex::insert(OMPropertyId *propertyId*, **OMStoredForm** *storedForm*, **OMPropertyOffset** *offset*, **OMPropertySize** *length***)**

Insert a new property into this OMStoredPropertySetIndex. The new property has id *propertyId*. The stored property representation is *storedForm*. The property value occupies *length* bytes starting at offset *offset*.

Defined in: OMStoredPropertySetIndex.cpp

Parameters

propertyId
The id of the property to insert.
storedForm
The stored form to use for the property.
offset
The offset of the property value in bytes.
length
The size of the property value in bytes.
Back to OMStoredPropertySetIndex

OMStoredPropertySetIndex::isValid

bool OMStoredPropertySetIndex::isValid(void) const

Is this OMStoredPropertySetIndex valid ?

Defined in: OMStoredPropertySetIndex.cpp

Return Value

True if this OMStoredPropertySetIndex is valid, false otherwise.

Back to OMStoredPropertySetIndex

OMStoredPropertySetIndex::iterate

void OMStoredPropertySetIndex::iterate(size_t& *context*, **OMPropertyId&** *propertyId*, **OMStoredForm&** *storedForm*, **OMPropertyOffset&** *offset*, **OMPropertySize&** *length*) **const**

Iterate over the properties in this OMStoredPropertySetIndex.

Defined in: OMStoredPropertySetIndex.cpp

Parameters

context
: Iteration context. Set this to 0 to start with the "first" property.

propertyId
: The id of the "current" property.

storedForm
: The stored form used for the "current" property.

offset
: The offset of the "current" property value in bytes.

length
: The size of the "current" property value in bytes.

Back to OMStoredPropertySetIndex

OMStoredSetIndex class

OMStoredSetIndex **class OMStoredSetIndex**

The in-memory representation of the on-disk index for a stored object set.

Defined in: OMStoredSetIndex.h

Author

Tim Bingham - tjb - (Avid Technology, Inc.)

Class Members

Public members.

OMStoredSetIndex**(size_t capacity, OMPropertyId keyPid, OMKeySize keySize)**
: Constructor.

~OMStoredSetIndex**(void)**
: Destructor.

OMUInt32 firstFreeKey**(void) const**
: The first free key in the set of local keys assigned to this **OMStoredSetIndex.**

void setFirstFreeKey**(OMUInt32 firstFreeKey)**
: Set the first free key in the set of local keys assigned to this **OMStoredSetIndex.**

OMUInt32 lastFreeKey**(void) const**

The last free key in the set of local keys assigned to this **OMStoredSetIndex.**

void setLastFreeKey**(OMUInt32 lastFreeKey)**

Set the last free key in the set of local keys assigned to this **OMStoredSetIndex.**

void insert**(size_t position, OMUInt32 localKey, OMUInt32 referenceCount, void* key)**

Insert a new element in this **OMStoredSetIndex.** The local key of an element is an integer. The local key is used to derive the name of the storage on which an element is saved. Local keys are assigned such that the names of existing elements do not have to change when other elements are added to or removed from the associated OMStrongReferenceSet. The local key is independent of the element's logical or physical position within the associated OMStrongReferenceSet. The local key is also independent of the element's unique key.

size_t entries**(void) const**

The number of elements in this **OMStoredSetIndex.**

void iterate**(size_t& context, OMUInt32& localKey, OMUInt32& referenceCount, void* key) const**

Iterate over the elements in this **OMStoredSetIndex.**

bool isValid**(void) const**

Is this **OMStoredSetIndex** valid ?

OMStoredSetIndex::entries

size_t OMStoredSetIndex::entries(void) const

The number of elements in this OMStoredSetIndex.

Defined in: OMStoredSetIndex.cpp

Return Value

The number of elements.

Back to OMStoredSetIndex

OMStoredSetIndex::firstFreeKey

OMUInt32 OMStoredSetIndex::firstFreeKey(void) const

The first free key in the set of local keys assigned to this OMStoredSetIndex.

Defined in: OMStoredSetIndex.cpp

Return Value

The highest previously allocated local key.

Back to OMStoredSetIndex

OMStoredSetIndex::insert

void OMStoredSetIndex::insert(size_t *position*, **OMUInt32** *localKey*, **OMUInt32** *referenceCount*, **void*** *key***)**

Insert a new element in this OMStoredSetIndex. The local key of an element is an integer. The local key is used to derive the name of the storage on which an element is saved. Local keys are assigned such that the names of existing elements do not have to change when other elements are added to or removed from the associated OMStrongReferenceSet. The local key is independent of the element's logical or physical position within the associated OMStrongReferenceSet. The local key is also independent of the element's unique key.

Defined in: OMStoredSetIndex.cpp

Parameters

position
: The position at which the new element should be inserted.

localKey
: The local key assigned to the element.

referenceCount
: The count of references to the element.

key
: The unique key of the element.

Back to OMStoredSetIndex

OMStoredSetIndex::isValid

bool OMStoredSetIndex::isValid(void) const

Is this OMStoredSetIndex valid ?

Defined in: OMStoredSetIndex.cpp

Return Value

True if this is a valid OMStoredSetIndex, false otherwise.

Back to OMStoredSetIndex

OMStoredSetIndex::iterate

void OMStoredSetIndex::iterate(size_t& *context*, **OMUInt32&** *localKey*, **OMUInt32&** *referenceCount*, **void*** *key***) const**

Iterate over the elements in this OMStoredSetIndex.

Defined in: OMStoredSetIndex.cpp

Parameters

context
Iteration context. Set this to 0 to start with the "first" element.
localKey
The local key of the "current" element.
referenceCount
The count of references to the "current" element.
key
The unique key of the "current" element.
Back to OMStoredSetIndex

OMStoredSetIndex::lastFreeKey

OMUInt32 OMStoredSetIndex::lastFreeKey(void) const

The last free key in the set of local keys assigned to this OMStoredSetIndex.

Defined in: OMStoredSetIndex.cpp

Return Value

The highest previously allocated local key.

Back to OMStoredSetIndex

OMStoredSetIndex::OMStoredSetIndex

OMStoredSetIndex::OMStoredSetIndex(size_t *capacity***)**

Constructor.

Defined in: OMStoredSetIndex.cpp

Parameters

capacity
The capacity of this OMStoredSetIndex.
Back to OMStoredSetIndex

OMStoredSetIndex::setFirstFreeKey

void OMStoredSetIndex::setFirstFreeKey(OMUInt32 *firstFreeKey***)**

Set the first free key in the set of local keys assigned to this OMStoredSetIndex.

Defined in: OMStoredSetIndex.cpp

Parameters

firstFreeKey
The highest allocated local key.

Back to OMStoredSetIndex

OMStoredSetIndex::setLastFreeKey

void OMStoredSetIndex::setLastFreeKey(OMUInt32 *lastFreeKey***)**

Set the last free key in the set of local keys assigned to this OMStoredSetIndex.

Defined in: OMStoredSetIndex.cpp

Parameters

lastFreeKey
The highest allocated local key.

Back to OMStoredSetIndex

OMStoredSetIndex::~OMStoredSetIndex

OMStoredSetIndex::~OMStoredSetIndex(void)

Destructor.

Defined in: OMStoredSetIndex.cpp

Back to OMStoredSetIndex

OMStoredStream class

OMStoredStream **class OMStoredStream**

Persistent data streams, contained with OMStoredObjects, supported by the Object Manager.

Defined in: OMStoredStream.h

Author

Tim Bingham - tjb - (Avid Technology, Inc.)

Class Members

Public members.

virtual ~OMStoredStream(void)

Destructor.

virtual void read(void* data, size_t size) const

Read *size* bytes from this **OMStoredStream** into the buffer at address *data*.

virtual void read(OMByte* data, const OMUInt32 bytes, OMUInt32& bytesRead) const

Attempt to read *bytes* bytes from this **OMStoredStream** into the buffer at address *data*. The actual number of bytes read is returned in *bytesRead*.

virtual void write(void* data, size_t size)

Write *size* bytes from the buffer at address *data* to this **OMStoredStream.**

virtual void write(const OMByte* data, const OMUInt32 bytes, OMUInt32& bytesWritten)

Attempt to write *bytes* bytes from the buffer at address *data* to this **OMStoredStream.** The actual number of bytes written is returned in *bytesWritten*.

virtual OMUInt64 size(void) const

The size of this **OMStoredStream** in bytes.

virtual void setSize(const OMUInt64 newSize)

Set the size of this **OMStoredStream** to *bytes*.

virtual OMUInt64 position(void) const

The current position for **read()** and **write()**, as an offset in bytes from the begining of this **OMStoredStream.**

virtual void setPosition(const OMUInt64 offset)

Set the current position for **read()** and **write()**, as an offset in bytes from the begining of this **OMStoredStream.**

virtual void close(void)

Close this **OMStoredStream.**

OMStoredVectorIndex class

OMStoredVectorIndex **class OMStoredVectorIndex**

The in-memory representation of the on-disk index for a stored object vector.

Defined in: OMStoredVectorIndex.h

Author

Tim Bingham - tjb - (Avid Technology, Inc.)

Class Members

Public members.

OMStoredVectorIndex**(size_t capacity)**

Constructor.

~OMStoredVectorIndex**(void)**

Destructor.

OMUInt32 firstFreeKey**(void) const**

The first free key in the set of local keys assigned to this **OMStoredVectorIndex.**

void setFirstFreeKey**(OMUInt32 firstFreeKey)**

Set the first free key in the set of local keys assigned to this **OMStoredVectorIndex.**

OMUInt32 lastFreeKey**(void) const**

The last free key in the set of local keys assigned to this **OMStoredVectorIndex.**

void setLastFreeKey**(OMUInt32 lastFreeKey)**

Set the last free key in the set of local keys assigned to this **OMStoredVectorIndex.**

void insert**(size_t position, OMUInt32 localKey)**

Insert a new element in this **OMStoredVectorIndex** at position *position* with local key *localKey*. The local key of an element is an integer. The local key is used to derive the name of the storage on which an element is saved. Local keys are assigned such that the names of existing elements do not have to change when other elements are added to or removed from the associated OMStrongReferenceVector. The local key is independent of the element's logical or physical position within the associated OMStrongReferenceVector.

size_t entries**(void) const**

The number of elements in this **OMStoredVectorIndex.**

void iterate**(size_t& context, OMUInt32& localKey) const**

Iterate over the elements in this **OMStoredVectorIndex.**

bool isValid**(void) const**

Is this **OMStoredVectorIndex** valid ?

OMStoredVectorIndex::entries

size_t OMStoredVectorIndex::entries(void) const

The number of elements in this OMStoredVectorIndex.

Defined in: OMStoredVectorIndex.cpp

Return Value

The number of elements.

Back to OMStoredVectorIndex

OMStoredVectorIndex::firstFreeKey

OMUInt32 OMStoredVectorIndex::firstFreeKey(void) const

The first free key in the set of local keys assigned to this OMStoredVectorIndex.

Defined in: OMStoredVectorIndex.cpp

Return Value

The highest previously allocated local key.

Back to OMStoredVectorIndex

OMStoredVectorIndex::insert

void OMStoredVectorIndex::insert(size_t *position*, **OMUInt32** *localKey***)**

Insert a new element in this OMStoredVectorIndex at position *position* with local key *localKey*. The local key of an element is an integer. The local key is used to derive the name of the storage on which an element is saved. Local keys are assigned such that the names of existing elements do not have to change when other elements are added to or removed from the associated OMStrongReferenceVector. The local key is independent of the element's logical or physical position within the associated OMStrongReferenceVector.

Defined in: OMStoredVectorIndex.cpp

Parameters

position
: The position at which the new element should be inserted.

localKey
: The local key assigned to the element.

Back to OMStoredVectorIndex

OMStoredVectorIndex::isValid

bool OMStoredVectorIndex::isValid(void) const

Is this OMStoredVectorIndex valid ?

Defined in: OMStoredVectorIndex.cpp

Return Value

True if this is a valid OMStoredVectorIndex, false otherwise.

Back to OMStoredVectorIndex

OMStoredVectorIndex::iterate

void OMStoredVectorIndex::iterate(size_t& *context*, **OMUInt32&** *localKey***) const**

Iterate over the elements in this OMStoredVectorIndex.

Defined in: OMStoredVectorIndex.cpp

Parameters

context
: Iteration context. Set this to 0 to start with the "first" element.

localKey
: The local key of the "current" element.

Back to OMStoredVectorIndex

OMStoredVectorIndex::lastFreeKey

OMUInt32 OMStoredVectorIndex::lastFreeKey(void) const

The last free key in the set of local keys assigned to this OMStoredVectorIndex.

Defined in: OMStoredVectorIndex.cpp

Return Value

The highest previously allocated local key.

Back to OMStoredVectorIndex

OMStoredVectorIndex::OMStoredVectorIndex

OMStoredVectorIndex::OMStoredVectorIndex(size_t *capacity***)**

Constructor.

Defined in: OMStoredVectorIndex.cpp

Parameters

capacity
The capacity of this OMStoredVectorIndex.

Back to OMStoredVectorIndex

OMStoredVectorIndex::setFirstFreeKey

void OMStoredVectorIndex::setFirstFreeKey(OMUInt32 *firstFreeKey***)**

Set the first free key in the set of local keys assigned to this OMStoredVectorIndex.

Defined in: OMStoredVectorIndex.cpp

Parameters

firstFreeKey
The highest allocated local key.

Back to OMStoredVectorIndex

OMStoredVectorIndex::setLastFreeKey

void OMStoredVectorIndex::setLastFreeKey(OMUInt32 *lastFreeKey***)**

Set the last free key in the set of local keys assigned to this OMStoredVectorIndex.

Defined in: OMStoredVectorIndex.cpp

Parameters

lastFreeKey
The highest allocated local key.

Back to OMStoredVectorIndex

OMStoredVectorIndex::~OMStoredVectorIndex

OMStoredVectorIndex::~OMStoredVectorIndex(void)

Destructor.

Defined in: OMStoredVectorIndex.cpp

Back to OMStoredVectorIndex

OMStreamProperty class

OMStreamProperty **class OMStreamProperty: public OMDataStreamProperty.**

Persistent typed data stream properties supported by the Object Manager.

Defined in: OMStreamProperty.h

Class Template Arguments

Element
The type of an **OMStreamProperty** element.

Author

Tim Bingham - tjb - (Avid Technology, Inc.)

Class Members

Public members.
OMStreamProperty**(const OMPropertyId propertyId, const wchar_t* name)**
Constructor.
virtual ~OMStreamProperty**(void)**
Destructor.
void readElements**(OMUInt64 index, OMUInt32 elementCount, Element* elements) const**

Read *elementCount Element*s, starting at *index*, from this **OMStreamProperty** into *elements*.

void writeElements**(OMUInt64 index, OMUInt32 elementCount, const Element* elements)**

Write *elementCount Element*s, starting at *index*, to this **OMStreamProperty** from *elements*.

void readElement**(OMUInt64 index, Element* element) const**

Read a single *Element*, at *index*, from this **OMStreamProperty** into *element*.

void writeElement**(OMUInt64 index, const Element* element)**

Write a single *Element*s, at *index*, to this **OMStreamProperty** from *element*.

void readElements**(OMUInt32 elementCount, Element* elements) const**

Read *elementCount Element*s from the current position in this **OMStreamProperty** into *elements*.

void writeElements**(OMUInt32 elementCount, const Element* elements)**

Write *elementCount Element*s to the current position in this **OMStreamProperty** from *elements*.

void readElement**(Element* element) const**

Read a single *Element* from the current position in this **OMStreamProperty** into *element*.

void writeElement**(const Element* element)**

Write a single *Element* to the current position in this **OMStreamProperty** from *element*.

void appendElements**(OMUInt32 elementCount, const Element* elements)**

Write *elementCount Element*s to the end of this **OMStreamProperty** from *elements*.

void appendElement**(const Element* element)**

Write a single *Element* to the end of this **OMStreamProperty.**

OMUInt64 index**(void) const**

The index of the current *Element*.

void setIndex**(const OMUInt64 newIndex) const**

Make the *Element* at *newIndex* the current one.

OMUInt64 elementCount**(void) const**

The count of *Element*s in this **OMStreamProperty.**

void setElementCount**(OMUInt64 newElementCount)**

Set the count of *Element*s in this **OMStreamProperty** to *newElementCount*.

OMStreamProperty::appendElement

template <class *Element***>**
void OMStreamProperty<*Element***>::appendElement(const Element*** *element***)**

Write a single *Element* to the end of this OMStreamProperty.

Defined in: OMStreamPropertyT.h

Parameters

element

The element.

Class Template Arguments

Element

The type of an OMStreamProperty element.

Back to OMStreamProperty

OMStreamProperty::appendElements

template <class *Element***>**
void OMStreamProperty<*Element***>::appendElements(OMUInt32** *elementCount***, const Element*** *elements***)**

Write *elementCount Element*s to the end of this OMStreamProperty from *elements*.

Defined in: OMStreamPropertyT.h

Parameters

elementCount
The element count.
elements
The elements.

Class Template Arguments

Element
The type of an OMStreamProperty element.
Back to OMStreamProperty

OMStreamProperty::elementCount

template <class *Element***>**
OMUInt64 OMStreamProperty<*Element***>::elementCount(void) const**

The count of *Element*s in this OMStreamProperty.

Defined in: OMStreamPropertyT.h

Return Value

The element count.

Class Template Arguments

Element
The type of an OMStreamProperty element.
Back to OMStreamProperty

OMStreamProperty::index

template <class *Element***>**
OMUInt64 OMStreamProperty<*Element***>::index(void) const**

The index of the current *Element*.

Defined in: OMStreamPropertyT.h

Return Value

The current index.

Class Template Arguments

Element
The type of an OMStreamProperty element.

Back to OMStreamProperty

OMStreamProperty::OMStreamProperty

template <class *Element***>**
OMStreamProperty<*Element***>::OMStreamProperty(const OMPropertyId** *propertyId***, const wchar_t*** *name***)**

Constructor.

Defined in: OMStreamPropertyT.h

Parameters

propertyId
The property id.

name
The name of this OMStreamProperty.

Class Template Arguments

Element
The type of an OMStreamProperty element.

Back to OMStreamProperty

OMStreamProperty::readElement

template <class *Element***>**
void OMStreamProperty<*Element***>::readElement(Element*** *element***) const**

Read a single *Element* from the current position in this OMStreamProperty into *element*.

Defined in: OMStreamPropertyT.h

Parameters

element
The element.

Class Template Arguments

Element
The type of an OMStreamProperty element.

Back to OMStreamProperty

OMStreamProperty::readElement

template <class *Element***>**
void OMStreamProperty<*Element***>::readElement(OMUInt64** *index***, Element*** *element***) const**

Read a single *Element*, at *index*, from this OMStreamProperty into *element*.

Defined in: OMStreamPropertyT.h

Parameters

index
The index from which to read the element.

element
The element.

Class Template Arguments

Element
The type of an OMStreamProperty element.

Back to OMStreamProperty

OMStreamProperty::readElements

template <class *Element***>**
void OMStreamProperty<*Element***>::readElements(OMUInt64** *index***, OMUInt32** *count***, Element*** *elements***) const**

Read *elementCount Elements*, starting at *index*, from this OMStreamProperty into *elements*.

Defined in: OMStreamPropertyT.h

Parameters

index

The index from which to read the elements.

count

The element count.

elements

The elements.

Class Template Arguments

Element

The type of an OMStreamProperty element.

Back to OMStreamProperty

OMStreamProperty::readElements

template <class *Element***>**
void OMStreamProperty<*Element***>::readElements(OMUInt32** *elementCount***, Element*** *elements***) const**

Read *elementCount Element*s from the current position in this OMStreamProperty into *elements*.

Defined in: OMStreamPropertyT.h

Parameters

elementCount

The element count.

elements

The elements.

Class Template Arguments

Element

The type of an OMStreamProperty element.

Back to OMStreamProperty

OMStreamProperty::setElementCount

template <class *Element***>**
void OMStreamProperty<*Element***>::setElementCount(OMUInt64** *newElementCount***)**

Set the count of *Element*s in this OMStreamProperty to *newElementCount*.

Defined in: OMStreamPropertyT.h

Parameters

newElementCount

The new element count.

Class Template Arguments

Element
The type of an OMStreamProperty element.

Back to OMStreamProperty

OMStreamProperty::setIndex

template <class *Element***>**
void OMStreamProperty<*Element***>::setIndex(const OMUInt64** *newIndex***) const**

Make the *Element* at *newIndex* the current one.

Defined in: OMStreamPropertyT.h

Parameters

newIndex
The new value for the current index.

Class Template Arguments

Element
The type of an OMStreamProperty element.

Back to OMStreamProperty

OMStreamProperty::writeElement

template <class *Element***>**
void OMStreamProperty<*Element***>::writeElement(const Element*** *element***)**

Write a single *Element* to the current position in this OMStreamProperty from *element*.

Defined in: OMStreamPropertyT.h

Parameters

element
The element.

Class Template Arguments

Element
The type of an OMStreamProperty element.

Back to OMStreamProperty

OMStreamProperty::writeElement

template <class *Element***>**
void OMStreamProperty<*Element***>::writeElement(OMUInt64** *index***, const Element*** *element***)**

Write a single *Element*s, at *index*, to this OMStreamProperty from *element*.

Defined in: OMStreamPropertyT.h

Parameters

index
: The index at which to write the element.

element
: The element.

Class Template Arguments

Element
: The type of an OMStreamProperty element.

Back to OMStreamProperty

OMStreamProperty::writeElements

template <class *Element***>**
void OMStreamProperty<*Element***>::writeElements(OMUInt64** *index***, OMUInt32** *elementCount***, const Element*** *elements***)**

Write *elementCount Element*s, starting at *index*, to this OMStreamProperty from *elements*.

Defined in: OMStreamPropertyT.h

Parameters

index
: The index at which to write the elements.

elementCount
: The element count.

elements
: The elements.

Class Template Arguments

Element

The type of an OMStreamProperty element.

Back to OMStreamProperty

OMStreamProperty::writeElements

template <class *Element***>**
void OMStreamProperty<*Element***>::writeElements(OMUInt32** *elementCount***, const Element*** *elements***)**

Write *elementCount Element*s to the current position in this OMStreamProperty from *elements*.

Defined in: OMStreamPropertyT.h

Parameters

elementCount
The element count.

elements
The elements.

Class Template Arguments

Element
The type of an OMStreamProperty element.

Back to OMStreamProperty

OMStreamProperty::~OMStreamProperty

template <class *Element***>**
OMStreamProperty<*Element***>::~OMStreamProperty(void)**

Destructor.

Defined in: OMStreamPropertyT.h

Class Template Arguments

Element
The type of an OMStreamProperty element.

Back to OMStreamProperty

OMStrongObjectReference class

OMStrongObjectReference **class OMStrongObjectReference: public OMObjectReference**

Persistent strong references to persistent objects.

Defined in: OMObjectReference.h

Author

Tim Bingham - tjb - (Avid Technology, Inc.)

Class Members

Public members.

OMStrongObjectReference**(void)**
: Constructor.

OMStrongObjectReference**(OMProperty* property, const wchar_t* name)**
: Constructor.

OMStrongObjectReference**(const OMStrongObjectReference&)**
: Copy constructor.

virtual ~OMStrongObjectReference**(void)**
: Destructor.

virtual bool isVoid**(void) const**
: Is this **OMStrongObjectReference** void ?

OMStrongObjectReference& operator=**(const OMStrongObjectReference& rhs)**
: Assignment. This operator provides value semantics for OMContainer. This operator does not provide assignment of object references.

bool operator==**(const OMStrongObjectReference& rhs) const**
: Equality. This operator provides value semantics for OMContainer. This operator does not provide equality of object references.

virtual void save**(void) const**
: Save this **OMStrongObjectReference**.

virtual void close**(void)**
: Close this **OMStrongObjectReference**.

virtual void detach**(void)**
: Detach this **OMStrongObjectReference**.

virtual void restore**(void)**
: Restore this **OMStrongObjectReference**.

virtual OMStorable* getValue**(void) const**
: Get the value of this **OMStrongObjectReference**. The value is a pointer to the referenced OMStorable.

virtual OMStorable* setValue**(const OMStorable* value)**
: Set the value of this **OMStrongObjectReference**. The value is a pointer to the referenced OMStorable.

Class Members

Protected members.

bool isLoaded**(void) const**
: Is this **OMStrongObjectReference** in the loaded state. If false there is a persisted representation of this **OMStrongObjectReference** that can be loaded.

void setLoaded**(void)**
: Put this **OMStrongObjectReference** into the loaded state.

void clearLoaded**(void)**
: Remove this **OMStrongObjectReference** from the loaded state.

virtual void load**(void)**
: Load the persisted representation of this **OMStrongObjectReference**.

bool _isLoaded

The state of this **OMStrongObjectReference.** This is false if a persisted representation of this element exists that has not yet been loaded, true otherwise.

wchar_t* _name

The name of this **OMStrongObjectReference.**

OMStrongObjectReference::clearLoaded

void OMStrongObjectReference::clearLoaded(void)

Remove this OMStrongObjectReference from the loaded state.

Defined in: OMObjectReference.cpp

Back to OMStrongObjectReference

OMStrongObjectReference::close

void OMStrongObjectReference::close(void)

Close this OMStrongObjectReference.

Defined in: OMObjectReference.cpp

Back to OMStrongObjectReference

OMStrongObjectReference::detach

void OMStrongObjectReference::detach(void)

Detach this OMStrongObjectReference.

Defined in: OMObjectReference.cpp

Back to OMStrongObjectReference

OMStrongObjectReference::getValue

OMStorable* OMStrongObjectReference::getValue(void) const

Get the value of this OMStrongObjectReference. The value is a pointer to the referenced OMStorable.

Defined in: OMObjectReference.cpp

Return Value

A pointer to the referenced OMStorable.

Back to OMStrongObjectReference

OMStrongObjectReference::isLoaded

bool OMStrongObjectReference::isLoaded(void) const

Is this OMStrongObjectReference in the loaded state. If false there is a persisted representation of this OMStrongObjectReference that can be loaded.

Defined in: OMObjectReference.cpp

Back to OMStrongObjectReference

OMStrongObjectReference::isVoid

bool OMStrongObjectReference::isVoid(void) const

Is this OMStrongObjectReference void ?

Defined in: OMObjectReference.cpp

Return Value

True if this OMStrongObjectReference is void, false otherwise.

Back to OMStrongObjectReference

OMStrongObjectReference::load

void OMStrongObjectReference::load(void)

Load the persisted representation of this OMStrongObjectReference.

Defined in: OMObjectReference.cpp

Back to OMStrongObjectReference

OMStrongObjectReference::OMStrongObjectReference

OMStrongObjectReference::OMStrongObjectReference(OMProperty* *property***, const wchar_t*** *name***)**

Constructor.

Defined in: OMObjectReference.cpp

Parameters

property
The OMProperty that contains this OMStrongObjectReference.

name
The name of this OMStrongObjectReference.

Back to OMStrongObjectReference

OMStrongObjectReference::OMStrongObjectReference

OMStrongObjectReference::OMStrongObjectReference(void)

Constructor.

Defined in: OMObjectReference.cpp

Back to OMStrongObjectReference

OMStrongObjectReference::OMStrongObjectReference

OMStrongObjectReference::OMStrongObjectReference(const OMStrongObjectReference& *rhs***)**

Copy constructor.

Defined in: OMObjectReference.cpp

Parameters

rhs
The OMStrongObjectReference to copy.

Back to OMStrongObjectReference

OMStrongObjectReference::operator=

OMStrongObjectReference& OMStrongObjectReference::operator=(const OMStrongObjectReference& *rhs***)**

Assignment. This operator provides value semantics for OMContainer. This operator does not provide assignment of object references.

Defined in: OMObjectReference.cpp

Return Value

The OMStrongObjectReference resulting from the assignment.

Parameters

rhs
The OMStrongObjectReference to be assigned.

Back to OMStrongObjectReference

OMStrongObjectReference::operator==

bool OMStrongObjectReference::operator==(const OMStrongObjectReference& *rhs*) const

Equality. This operator provides value semantics for OMContainer. This operator does not provide equality of object references.

Defined in: OMObjectReference.cpp

Return Value

True if the values are the same, false otherwise.

Parameters

rhs
The OMStrongObjectReference to be compared.

Back to OMStrongObjectReference

OMStrongObjectReference::restore

void OMStrongObjectReference::restore(void)

Restore this OMStrongObjectReference.

Defined in: OMObjectReference.cpp

Back to OMStrongObjectReference

OMStrongObjectReference::save

void OMStrongObjectReference::save(void) const

Save this OMStrongObjectReference.

Defined in: OMObjectReference.cpp

Back to OMStrongObjectReference

OMStrongObjectReference::setLoaded

void OMStrongObjectReference::setLoaded(void)

Put this OMStrongObjectReference into the loaded state.

Defined in: OMObjectReference.cpp

Back to OMStrongObjectReference

OMStrongObjectReference::setValue

OMStorable* OMStrongObjectReference::setValue(const OMStorable* *value***)**

Set the value of this OMStrongObjectReference. The value is a pointer to the referenced OMStorable.

Defined in: OMObjectReference.cpp

Return Value

A pointer to previous OMStorable, if any.

Parameters

value
A pointer to the new OMStorable.

Back to OMStrongObjectReference

OMStrongObjectReference::~OMStrongObjectReference

OMStrongObjectReference::~OMStrongObjectReference(void)

Destructor.

Defined in: OMObjectReference.cpp

Back to OMStrongObjectReference

OMStrongReference class

OMStrongReference **class OMStrongReference: public OMReferenceProperty**

Persistent strong reference (contained object) properties supported by the Object Manager.

Defined in: OMStrongReference.h

Author

Tim Bingham - tjb - (Avid Technology, Inc.)

Class Members

Public members.
OMStrongReference**(const OMPropertyId propertyId, const wchar_t* name)**
Constructor.
~OMStrongReference**(void)**
Destructor.

OMStrongReference::OMStrongReference

OMStrongReference::OMStrongReference(void)

Constructor.

Defined in: OMStrongReference.cpp

Back to OMStrongReference

OMStrongReference::~OMStrongReference

OMStrongReference::~OMStrongReference(void)

Destructor.

Defined in: OMStrongReference.cpp

Back to OMStrongReference

OMStrongReferenceProperty class

OMStrongReferenceProperty **class OMStrongReferenceProperty: public OMStrongReference**

Persistent strong reference (contained object) properties supported by the Object Manager.

Defined in: OMStrongRefProperty.h

Class Template Arguments

ReferencedObject
The type of the referenced (contained) object. This type must be a descendant of OMStorable.

Author

Tim Bingham - tjb - (Avid Technology, Inc.)

Class Members

Public members.
OMStrongReferenceProperty(const OMPropertyId propertyId, const wchar_t* name)
Constructor.
virtual ~OMStrongReferenceProperty(void)
Destructor.
virtual void getValue**(ReferencedObject*& object) const**
Get the value of this **OMStrongReferenceProperty.**
virtual ReferencedObject* setValue**(const ReferencedObject* object)**
Set the value of this **OMStrongReferenceProperty.**
virtual ReferencedObject* clearValue**(void)**
Clear the value of this **OMStrongReferenceProperty.**
OMStrongReferenceProperty&ltReferencedObject>& operator=
Assignment operator.
ReferencedObject* operator->**(void)**
Dereference operator.
const ReferencedObject* operator->**(void) const**
Dereference operator.
operator ReferencedObject*() const
Type conversion. Convert an **OMStrongReferenceProperty** into a pointer to the referenced (contained) *ReferencedObject*.
virtual void save**(void) const**
Save this **OMStrongReferenceProperty.**
virtual void close**(void)**
Close this OMProperty.
virtual void detach**(void)**
Detach this OMProperty.
virtual void restore**(size_t externalSize)**
Restore this **OMStrongReferenceProperty,** the external (persisted) size of the **OMStrongReferenceProperty** is *externalSize*.
virtual bool isVoid**(void) const**
Is this **OMStrongReferenceProperty** void ?
virtual void removeProperty**(void)**
Remove this optional **OMStrongReferenceProperty.**
virtual void getBits**(OMByte* bits, size_t size) const**
Get the raw bits of this **OMStrongReferenceProperty.** The raw bits are copied to the buffer at address *bits* which is *size* bytes in size.
virtual void setBits**(const OMByte* bits, size_t size)**
Set the raw bits of this **OMStrongReferenceProperty.** The raw bits are copied from the buffer at address *bits* which is *size* bytes in size.
virtual OMObject* getObject**(void) const**

Get the value of this **OMStrongReferenceProperty.**

virtual OMObject* setObject**(const OMObject* object)**

set the value of this **OMStrongReferenceProperty.**

virtual OMStorable* storable**(void) const**

The value of this **OMStrongReferenceProperty** as an OMStorable**.**

OMStrongReferenceProperty::clearValue

template <class *ReferencedObject***>**
ReferencedObject* OMStrongReferenceProperty<*ReferencedObject***>::clearValue(void)**

Clear the value of this OMStrongReferenceProperty.

Defined in: OMStrongRefPropertyT.h

Return Value

A pointer to the old *ReferencedObject*. If lazy loading is enabled and the referenced object was never loaded the value returned is 0.

Class Template Arguments

ReferencedObject

The type of the referenced (contained) object. This type must be a descendant of OMStorable**.**

Back to OMStrongReferenceProperty

OMStrongReferenceProperty::close

template <class *ReferencedObject***>**
void OMStrongReferenceProperty<*ReferencedObject***>::close(void)**

Close this OMStrongReferenceProperty.

Defined in: OMStrongRefPropertyT.h

Class Template Arguments

ReferencedObject

The type of the referenced (contained) object. This type must be a descendant of OMStorable**.**

Back to OMStrongReferenceProperty

OMStrongReferenceProperty::detach

template <class *ReferencedObject***>**
void OMStrongReferenceProperty<*ReferencedObject***>::detach(void)**

Detach this OMStrongReferenceProperty.

Defined in: OMStrongRefPropertyT.h

Class Template Arguments

ReferencedObject
: The type of the referenced (contained) object. This type must be a descendant of OMStorable.

Back to OMStrongReferenceProperty

OMStrongReferenceProperty::getBits

template <class *ReferencedObject***>**
void OMStrongReferenceProperty<*ReferencedObject***>::getBits(OMByte*** *bits***, size_t** *size***) const**

Get the raw bits of this OMStrongReferenceProperty. The raw bits are copied to the buffer at address *bits* which is *size* bytes in size.

Defined in: OMStrongRefPropertyT.h

Parameters

bits
: The address of the buffer into which the raw bits are copied.

size
: The size of the buffer.

Class Template Arguments

ReferencedObject
: The type of the referenced object. This type must be a descendant of OMStorable.

Back to OMStrongReferenceProperty

OMStrongReferenceProperty::getObject

template <class *ReferencedObject***>**
OMObject* OMStrongReferenceProperty<*ReferencedObject***>::getObject(void) const**

Get the value of this OMStrongReferenceProperty.

Defined in: OMStrongRefPropertyT.h

Return Value

A pointer to an OMObject.

Class Template Arguments

ReferencedObject
: The type of the referenced (pointed to) object. This type must be a descendant of OMStorable.

Back to OMStrongReferenceProperty

OMStrongReferenceProperty::getValue

template <class *ReferencedObject***>**
void OMStrongReferenceProperty<*ReferencedObject***>::getValue(ReferencedObject*& *object***) const**

Get the value of this OMStrongReferenceProperty.

Defined in: OMStrongRefPropertyT.h

Parameters

object
: A pointer to a *ReferencedObject* by reference.

Class Template Arguments

ReferencedObject
: The type of the referenced (contained) object. This type must be a descendant of OMStorable.

Back to OMStrongReferenceProperty

OMStrongReferenceProperty::isVoid

template <class *ReferencedObject***>**
bool OMStrongReferenceProperty<*ReferencedObject***>::isVoid(void) const**

Is this OMStrongReferenceProperty void ?

Defined in: OMStrongRefPropertyT.h

Return Value

True if this OMStrongReferenceProperty is void, false otherwise

Class Template Arguments

ReferencedObject

The type of the referenced (contained) object. This type must be a descendant of OMStorable.

Back to OMStrongReferenceProperty

OMStrongReferenceProperty::operator ReferencedObject*

template <class *ReferencedObject***>**
OMStrongReferenceProperty<*ReferencedObject***>::operator ReferencedObject* (void) const**

Type conversion. Convert an OMStrongReferenceProperty into a pointer to the referenced (contained) *ReferencedObject*.

Defined in: OMStrongRefPropertyT.h

Return Value

The result of the conversion as a value of type pointer to *ReferencedObject*.

Class Template Arguments

ReferencedObject
The type of the referenced (contained) object. This type must be a descendant of OMStorable.

Back to OMStrongReferenceProperty

OMStrongReferenceProperty::operator->

template <class *ReferencedObject***>**
const ReferencedObject* OMStrongReferenceProperty<*ReferencedObject***>::operator->(void) const**

Dereference operator.

Defined in: OMStrongRefPropertyT.h

Return Value

A pointer to a *ReferencedObject* by value.

Class Template Arguments

ReferencedObject
The type of the referenced (contained) object. This type must be a descendant of OMStorable

Back to OMStrongReferenceProperty

OMStrongReferenceProperty::operator->

template <class *ReferencedObject***>**
ReferencedObject* OMStrongReferenceProperty<*ReferencedObject***>::operator->(void)**

Dereference operator.

Defined in: OMStrongRefPropertyT.h

Return Value

A pointer to a *ReferencedObject* by value.

Class Template Arguments

ReferencedObject
: The type of the referenced (contained) object. This type must be a descendant of OMStorable

Back to OMStrongReferenceProperty

OMStrongReferenceProperty::operator=

template <class *ReferencedObject***>**
OMStrongReferenceProperty&ltReferencedObject>&
OMStrongReferenceProperty<*ReferencedObject***>::operator=(const ReferencedObject*** *value***)**

Assignment operator.

Defined in: OMStrongRefPropertyT.h

Return Value

The result of the assignment.

Parameters

value
: A pointer to a *ReferencedObject* by value.

Class Template Arguments

ReferencedObject
: The type of the referenced (contained) object. This type must be a descendant of OMStorable

Back to OMStrongReferenceProperty

OMStrongReferenceProperty::removeProperty

template <class *ReferencedObject***>**
void OMStrongReferenceProperty<*ReferencedObject***>::removeProperty(void)**

Remove this optional OMStrongReferenceProperty.

Defined in: OMStrongRefPropertyT.h

Class Template Arguments

ReferencedObject
The type of the referenced (contained) object. This type must be a descendant of OMStorable.

Back to OMStrongReferenceProperty

OMStrongReferenceProperty::restore

template <class *ReferencedObject***>**
void OMStrongReferenceProperty<*ReferencedObject***>::restore(size_t** *externalSize***)**

Restore this OMStrongReferenceProperty, the external (persisted) size of the OMStrongReferenceProperty is *externalSize*.

Defined in: OMStrongRefPropertyT.h

Parameters

externalSize
The external (persisted) size of the OMStrongReferenceProperty.

Class Template Arguments

ReferencedObject
The type of the referenced (contained) object. This type must be a descendant of OMStorable.

Back to OMStrongReferenceProperty

OMStrongReferenceProperty::save

template <class *ReferencedObject***>**
void OMStrongReferenceProperty<*ReferencedObject***>::save(void) const**

Save this OMStrongReferenceProperty.

Defined in: OMStrongRefPropertyT.h

Class Template Arguments

ReferencedObject
: The type of the referenced (contained) object. This type must be a descendant of OMStorable.

Back to OMStrongReferenceProperty

OMStrongReferenceProperty::setBits

template <class *ReferencedObject***>**
void OMStrongReferenceProperty<*ReferencedObject***>::setBits(const OMByte*** *bits***, size_t** *size***)**

Set the raw bits of this OMStrongReferenceProperty. The raw bits are copied from the buffer at address *bits* which is *size* bytes in size.

Defined in: OMStrongRefPropertyT.h

Parameters

bits
: The address of the buffer into which the raw bits are copied.

size
: The size of the buffer.

Class Template Arguments

ReferencedObject
: The type of the referenced object. This type must be a descendant of OMStorable.

Back to OMStrongReferenceProperty

OMStrongReferenceProperty::setObject

template <class *ReferencedObject***>**
OMObject* OMStrongReferenceProperty<*ReferencedObject***>::setObject(const OMObject*** *object***)**

Set the value of this OMStrongReferenceProperty.

Defined in: OMStrongRefPropertyT.h

Return Value

A pointer to the old OMObject. If lazy loading is enabled and the referenced object was never loaded the value returned is 0.

Parameters

object
: A pointer to the new OMObject.

Class Template Arguments

ReferencedObject
: The type of the referenced (contained) object. This type must be a descendant of OMStorable.

Back to OMStrongReferenceProperty

OMStrongReferenceProperty::setValue

template <class *ReferencedObject***>**
ReferencedObject* OMStrongReferenceProperty<*ReferencedObject***>::setValue(const ReferencedObject*** *object***)**

Set the value of this OMStrongReferenceProperty.

Defined in: OMStrongRefPropertyT.h

Return Value

A pointer to the old *ReferencedObject*. If lazy loading is enabled and the referenced object was never loaded the value returned is 0.

Parameters

object
: A pointer to the new *ReferencedObject*.

Class Template Arguments

ReferencedObject
: The type of the referenced (contained) object. This type must be a descendant of OMStorable.

Back to OMStrongReferenceProperty

OMStrongReferenceProperty::storable

OMStorable* OMStrongReferenceProperty::storable(void) const

The value of this OMStrongReferenceProperty as an OMStorable.

Defined in: OMStrongRefPropertyT.h

Return Value

The OMStorable represented by this OMStrongReferenceProperty

Back to OMStrongReferenceProperty

OMStrongReferenceSet class

OMStrongReferenceSet **class OMStrongReferenceSet: public OMSReferenceSetProperty**

Persistent sets of uniquely identified strongly referenced (contained) objects supported by the Object Manager. Objects are accessible by unique identifier (the key). The objects are not ordered. Duplicates objects are not allowed.

Defined in: OMStrongReferenceSet.h

Author

Tim Bingham - tjb - (Avid Technology, Inc.)

Class Members

Public members.

OMStrongReferenceSet**(const OMPropertyId propertyId, const wchar_t* name)**
: Constructor.

virtual ~OMStrongReferenceSet**(void)**
: Destructor.

OMStrongReferenceSet::OMStrongReferenceSet

OMStrongReferenceSet::OMStrongReferenceSet(const OMPropertyId *propertyId*, **const wchar_t*** *name***)**

Constructor.

Defined in: OMStrongReferenceSet.cpp

Parameters

propertyId
: The property id.

name
: The name of this OMStrongReferenceSet.

Back to OMStrongReferenceSet

OMStrongReferenceSet::~OMStrongReferenceSet

OMStrongReferenceSet::~OMStrongReferenceSet(void)

Destructor.

Defined in: OMStrongReferenceSet.cpp

Back to OMStrongReferenceSet

OMStrongReferenceSetElement class

OMStrongReferenceSetElement **class OMStrongReferenceSetElement: public OMContainerElement**

Elements of Object Manager reference sets.

Defined in: OMContainerElement.h

Author

Tim Bingham - tjb - (Avid Technology, Inc.)

Class Members

Public members.

OMStrongReferenceSetElement**(void)**
: Constructor.

OMStrongReferenceSetElement**(OMProperty* property, const wchar_t* name, OMUInt32 localKey, OMUInt32 referenceCount, void* identification, size_t identificationSize)**
: Constructor.

OMStrongReferenceSetElement**(const OMStrongReferenceSetElement& rhs)**
: Copy constructor.

~OMStrongReferenceSetElement**(void)**
: Destructor.

OMStrongReferenceSetElement& operator=**(const OMStrongReferenceSetElement& rhs)**
: Assignment. This operator provides value semantics for OMSet. This operator does not provide assignment of object references.

bool operator==**(const OMStrongReferenceSetElement& rhs) const**
: Equality. This operator provides value semantics for OMSet. This operator does not provide equality of object references.

OMStorable* setValue**(void* identification, const OMStorable* value)**
: Set the value of this **OMStrongReferenceSetElement.**

void* identification**(void) const**
: The unique key of this **OMStrongReferenceSetElement.**

OMUInt32 referenceCount**(void) const**
: The count of weak references to this **OMStrongReferenceSetElement.**

Class Members

Private members.

void* _identification
: The unique key of this **OMStrongReferenceSetElement.** The element's unique key is in memory even when the referenced object is not. That way we can tell if an object is present in a container without loading the object.

OMUInt32 _referenceCount
: The count of weak references to this **OMStrongReferenceSetElement.**

OMStrongReferenceSetElement::identification

void* OMStrongReferenceSetElement::identification(void)

The unique key of this OMStrongReferenceSetElement.

Defined in: OMContainerElement.cpp

Return Value

The unique key of this OMStrongReferenceSetElement.

Back to OMStrongReferenceSetElement

OMStrongReferenceSetElement::OMStrongReferenceSetElement

OMStrongReferenceSetElement::OMStrongReferenceSetElement(OMProperty* *property*, **const wchar_t*** *name*, **OMUInt32** *localKey*, **OMUInt32** *referenceCount*, **void*** *identification***)**

Constructor.

Defined in: OMContainerElement.cpp

Parameters

property
: The OMProperty (a set property) that contains this OMStrongReferenceSetElement.

name
: The name of this OMStrongReferenceSetElement.

localKey
: The local key of this OMStrongReferenceSetElement within it's set.

referenceCount
: The unique key of this OMStrongReferenceSetElement.

identification
: TBS

Back to OMStrongReferenceSetElement

OMStrongReferenceSetElement::OMStrongReferenceSetElement

OMStrongReferenceSetElement::OMStrongReferenceSetElement(const OMStrongReferenceSetElement& *rhs***)**

Copy constructor.

Defined in: OMContainerElement.cpp

Parameters

rhs
The OMStrongReferenceSetElement to copy.
Back to OMStrongReferenceSetElement

OMStrongReferenceSetElement::OMStrongReferenceSetElement

OMStrongReferenceSetElement::OMStrongReferenceSetElement(void)

Constructor.

Defined in: OMContainerElement.cpp

Back to OMStrongReferenceSetElement

OMStrongReferenceSetElement::operator=

OMStrongReferenceSetElement& OMStrongReferenceSetElement::operator=(const OMStrongReferenceSetElement& *rhs*)

Assignment. This operator provides value semantics for OMSet. This operator does not provide assignment of object references.

Defined in: OMContainerElement.cpp

Return Value

The OMStrongReferenceSetElement resulting from the assignment.

Parameters

rhs
The OMStrongReferenceSetElement to be assigned.
Back to OMStrongReferenceSetElement

OMStrongReferenceSetElement::operator==

bool OMStrongReferenceSetElement::operator==(const OMStrongReferenceSetElement& *rhs*)

Equality. This operator provides value semantics for OMSet. This operator does not provide equality of object references.

Defined in: OMContainerElement.cpp

Return Value

True if the values are the same, false otherwise.

Parameters

rhs
The OMStrongReferenceSetElement to be compared.

Back to OMStrongReferenceSetElement

OMStrongReferenceSetElement::referenceCount

OMUInt32 OMStrongReferenceSetElement::referenceCount(void)

The count of weak references to this OMStrongReferenceSetElement.

Defined in: OMContainerElement.cpp

Return Value

The count of weak references.

Back to OMStrongReferenceSetElement

OMStrongReferenceSetElement::setValue

OMStorable* OMStrongReferenceSetElement::setValue(void* *identification***)**

Set the value of this OMContainerElement.

Defined in: OMContainerElement.cpp

Return Value

A pointer to previous *ReferencedObject*, if any.

Parameters

identification
A pointer to the new *ReferencedObject*.

Back to OMStrongReferenceSetElement

OMStrongReferenceSetElement::~OMStrongReferenceSetElement

OMStrongReferenceSetElement::~OMStrongReferenceSetElement(void)

Destructor.

Defined in: OMContainerElement.cpp

Back to OMStrongReferenceSetElement

OMStrongReferenceSetIterator class

OMStrongReferenceSetIterator **class OMStrongReferenceSetIterator: public OMReferenceContainerIterator**

Iterators over OMStrongReferenceSetPropertys.

Defined in: OMStrongReferenceSetIter.h

Class Template Arguments

ReferencedObject
The type of the contained objects.

Author

Tim Bingham - tjb - (Avid Technology, Inc.)

Class Members

Public members.

OMStrongReferenceSetIterator**(const OMStrongReferenceSetProperty&ltUniqueIdentification, ReferencedObject>& set, OMIteratorPosition initialPosition = OMBefore)**
Create an **OMStrongReferenceSetIterator** over the given OMStrongReferenceSetProperty *set* and initialize it to the given *initialPosition*. If *initialPosition* is specified as OMBefore then this **OMStrongReferenceSetIterator** is made ready to traverse the associated OMStrongReferenceSetProperty in the forward direction (increasing *Key*s). If *initialPosition* is specified as OMAfter then this **OMStrongReferenceSetIterator** is made ready to traverse the associated OMStrongReferenceSetProperty in the reverse direction (decreasing *Key*s).

virtual ~OMStrongReferenceSetIterator**(void)**
Destroy this **OMStrongReferenceSetIterator.**

virtual OMReferenceContainerIterator* copy**(void) const**
Create a copy of this **OMStrongReferenceSetIterator.**

virtual void reset**(OMIteratorPosition initialPosition = OMBefore)**
Reset this **OMStrongReferenceSetIterator** to the given *initialPosition*. If *initialPosition* is specified as OMBefore then this **OMStrongReferenceSetIterator** is made ready to traverse the associated OMStrongReferenceSetProperty in the forward direction (increasing *Key*s). If *initialPosition* is specified as OMAfter then this **OMStrongReferenceSetIterator** is made ready to traverse the associated OMStrongReferenceSetProperty in the reverse direction (decreasing *Key*s).

virtual bool before**(void) const**
Is this **OMStrongReferenceSetIterator** positioned before the first *ReferencedObject* ?

virtual bool after**(void) const**
Is this **OMStrongReferenceSetIterator** positioned after the last *ReferencedObject* ?

virtual bool valid**(void) const**
Is this **OMStrongReferenceSetIterator** validly positioned on a *ReferencedObject* ?

virtual size_t count**(void) const**

The number of *ReferencedObject*s in the associated OMStrongReferenceSetProperty.

virtual bool operator++**()**

Advance this **OMStrongReferenceSetIterator** to the next *ReferencedObject*, if any. If the end of the associated OMStrongReferenceSetProperty is not reached then the result is **true** , **valid** becomes **true** and **after** becomes **false** . If the end of the associated OMStrongReferenceSetProperty is reached then the result is **false** , **valid** becomes **false** and **after** becomes **true** .

virtual bool operator--**()**

Retreat this **OMStrongReferenceSetIterator** to the previous *ReferencedObject*, if any. If the beginning of the associated OMStrongReferenceSetProperty is not reached then the result is **true** , **valid** becomes **true** and **before** becomes **false** . If the beginning of the associated OMStrongReferenceSetProperty is reached then the result is **false** , **valid** becomes **false** and **before** becomes **true** .

virtual ReferencedObject* value**(void) const**

Return the *ReferencedObject* in the associated OMStrongReferenceSetProperty at the position currently designated by this **OMStrongReferenceSetIterator.**

virtual ReferencedObject* setValue**(const ReferencedObject* newObject)**

Set the *ReferencedObject* in the associated OMStrongReferenceSetProperty at the position currently designated by this **OMStrongReferenceSetIterator** to *newObject*. The previous *ReferencedObject*, if any, is returned. To preserve the ordering of *Key*s, the *Key* of *newObject* must be the same as that of the existing *ReferencedObject*.

virtual ReferencedObject* clearValue**(void)**

Set the *ReferencedObject* in the associated OMStrongReferenceSetProperty at the position currently designated by this **OMStrongReferenceSetIterator** to 0. The previous *ReferencedObject*, if any, is returned.

UniqueIdentification identification**(void) const**

Return the *Key* of the *ReferencedObject* in the associated OMStrongReferenceSetProperty at the position currently designated by this **OMStrongReferenceSetIterator.**

virtual OMObject* currentObject**(void) const**

Return the *OMObject* in the associated reference container property at the position currently designated by this **OMStrongReferenceSetIterator.**

OMStrongReferenceSetIterator**(const SetIterator& iter)**

Create an **OMStrongReferenceSetIterator** given an underlying OMSetIterator.

OMStrongReferenceSetIterator::after

template <class *ReferencedObject***>**
bool OMStrongReferenceSetIterator<*ReferencedObject***>::after(void) const**

Is this OMStrongReferenceSetIterator positioned after the last *ReferencedObject* ?

Defined in: OMStrongReferenceSetIterT.h

Return Value

true if this OMStrongReferenceSetIterator is positioned after the last *ReferencedObject*, **false** otherwise.

Class Template Arguments

ReferencedObject
The type of the contained objects.

Back to OMStrongReferenceSetIterator

OMStrongReferenceSetIterator::before

template <class *ReferencedObject***>**
bool OMStrongReferenceSetIterator<*ReferencedObject***>::before(void) const**

Is this OMStrongReferenceSetIterator positioned before the first *ReferencedObject* ?

Defined in: OMStrongReferenceSetIterT.h

Return Value

true if this OMStrongReferenceSetIterator is positioned before the first *ReferencedObject*, **false** otherwise.

Class Template Arguments

ReferencedObject
The type of the contained objects.

Back to OMStrongReferenceSetIterator

OMStrongReferenceSetIterator::clearValue

template <class *ReferencedObject***>**
ReferencedObject* OMStrongReferenceSetIterator<*ReferencedObject***>::clearValue(void)**

Set the *ReferencedObject* in the associated OMStrongReferenceSetProperty at the position currently designated by this OMStrongReferenceSetIterator to 0. The previous *ReferencedObject*, if any, is returned.

Defined in: OMStrongReferenceSetIterT.h

Return Value

The previous *ReferencedObject* if any, otherwise 0.

Class Template Arguments

ReferencedObject
The type of the contained objects.

Back to OMStrongReferenceSetIterator

OMStrongReferenceSetIterator::copy

template <class *ReferencedObject***>**
OMReferenceContainerIterator* OMStrongReferenceSetIterator<*ReferencedObject***>::copy(void) const**

Create a copy of this OMStrongReferenceSetIterator.

Defined in: OMStrongReferenceSetIterT.h

Return Value

The new OMStrongReferenceSetIterator.

Class Template Arguments

ReferencedObject
The type of the contained objects.

Back to OMStrongReferenceSetIterator

OMStrongReferenceSetIterator::count

template <class *ReferencedObject***>**
size_t OMStrongReferenceSetIterator<*ReferencedObject***>::count(void) const**

The number of *ReferencedObject*s in the associated OMStrongReferenceSetProperty.

Defined in: OMStrongReferenceSetIterT.h

Return Value

The number of *ReferencedObject*s

Class Template Arguments

ReferencedObject
The type of the contained objects.

Back to OMStrongReferenceSetIterator

OMStrongReferenceSetIterator::currentObject

template <class *ReferencedObject***>**
OMObject* OMStrongReferenceSetIterator<*ReferencedObject***>::currentObject(void) const**

Return the OMObject in the associated OMStrongReferenceSetProperty at the position currently designated by this OMStrongReferenceSetIterator.

Defined in: OMStrongReferenceSetIterT.h

Return Value

The OMObject at the current position.

Class Template Arguments

ReferencedObject
The type of the contained objects.

Back to OMStrongReferenceSetIterator

OMStrongReferenceSetIterator::identification

template <class *ReferencedObject***>**
UniqueIdentification OMStrongReferenceSetIterator<*ReferencedObject***>::identification(void) const**

Return the *Key* of the *ReferencedObject* in the associated OMStrongReferenceSetProperty at the position currently designated by this OMStrongReferenceSetIterator.

Defined in: OMStrongReferenceSetIterT.h

Return Value

The *Key* at the current position.

Class Template Arguments

ReferencedObject
The type of the contained objects.

Back to OMStrongReferenceSetIterator

OMStrongReferenceSetIterator::OMStrongReferenceSetIterator

template <class *ReferencedObject***>**
OMStrongReferenceSetIterator<*ReferencedObject***>::OMStrongReferenceSetIterator(const OMStrongReferenceSetProperty<** *UniqueIdentification***, ReferencedObject>&** *set***)**

Create an OMStrongReferenceSetIterator over the given OMStrongReferenceSetProperty *set* and initialize it to the given *initialPosition*. If *initialPosition* is specified as OMBefore then this OMStrongReferenceSetIterator is made ready to traverse the associated OMStrongReferenceSetProperty in the forward direction (increasing *Keys*). If *initialPosition* is specified as OMAfter then this OMStrongReferenceSetIterator is made ready to traverse the associated OMStrongReferenceSetProperty in the reverse direction (decreasing *Keys*).

Defined in: OMStrongReferenceSetIterT.h

Parameters

UniqueIdentification

The OMStrongReferenceSet over which to iterate.

set

The initial position for this OMStrongReferenceSetIterator.

Class Template Arguments

ReferencedObject

The type of the contained objects.

Back to OMStrongReferenceSetIterator

OMStrongReferenceSetIterator::OMStrongReferenceSetIterator

template <class *ReferencedObject***>**
OMStrongReferenceSetIterator<*ReferencedObject***>::OMStrongReferenceSetIterator(const SetIterator&** *iter***)**

Create an OMStrongReferenceSetIterator given an underlying OMSetIterator.

Defined in: OMStrongReferenceSetIterT.h

Parameters

iter

The underlying OMSetIterator.

Class Template Arguments

ReferencedObject

The type of the contained objects.

Back to OMStrongReferenceSetIterator

OMStrongReferenceSetIterator::operator++

template <class *ReferencedObject***>**
bool OMStrongReferenceSetIterator<*ReferencedObject***>::operator++(void)**

Advance this OMStrongReferenceSetIterator to the next *ReferencedObject*, if any. If the end of the associated OMStrongReferenceSetProperty is not reached then the result is **true** , **valid** becomes **true** and **after** becomes **false** . If the end of the associated OMStrongReferenceSetProperty is reached then the result is **false** , **valid** becomes **false** and **after** becomes **true** .

Defined in: OMStrongReferenceSetIterT.h

Return Value

false if this OMStrongReferenceSetIterator has passed the last *ReferencedObject*, **true** otherwise.

Class Template Arguments

ReferencedObject
The type of the contained objects.

Back to OMStrongReferenceSetIterator

OMStrongReferenceSetIterator::operator--

template <class *ReferencedObject***>**
bool OMStrongReferenceSetIterator<*ReferencedObject***>::operator--(void)**

Retreat this OMStrongReferenceSetIterator to the previous *ReferencedObject*, if any. If the beginning of the associated OMStrongReferenceSetProperty is not reached then the result is **true** , **valid** becomes **true** and **before** becomes **false** . If the beginning of the associated OMStrongReferenceSetProperty is reached then the result is **false** , **valid** becomes **false** and **before** becomes **true** .

Defined in: OMStrongReferenceSetIterT.h

Return Value

false if this OMStrongReferenceSetIterator has passed the first *ReferencedObject*, **true** otherwise.

Class Template Arguments

ReferencedObject
The type of the contained objects.

Back to OMStrongReferenceSetIterator

OMStrongReferenceSetIterator::reset

template <class *ReferencedObject***>**
void OMStrongReferenceSetIterator<*ReferencedObject***>::reset(OMIteratorPosition** *initialPosition***)**

Reset this OMStrongReferenceSetIterator to the given *initialPosition*. If *initialPosition* is specified as OMBefore then this OMStrongReferenceSetIterator is made ready to traverse the associated OMStrongReferenceSetProperty in the forward direction (increasing *Keys*). If *initialPosition* is specified as OMAfter then this OMStrongReferenceSetIterator is made ready to traverse the associated OMStrongReferenceSetProperty in the reverse direction (decreasing *Keys*).

Defined in: OMStrongReferenceSetIterT.h

Parameters

initialPosition
The position to which this OMStrongReferenceSetIterator should be reset.

Class Template Arguments

ReferencedObject

The type of the contained objects.

Back to OMStrongReferenceSetIterator

OMStrongReferenceSetIterator::setValue

template <class *ReferencedObject***>**
ReferencedObject* OMStrongReferenceSetIterator<*ReferencedObject***>::setValue(const ReferencedObject*** *newObject***)**

Set the *ReferencedObject* in the associated OMStrongReferenceSetProperty at the position currently designated by this OMStrongReferenceSetIterator to *newObject*. The previous *ReferencedObject*, if any, is returned. To preserve the ordering of *Keys*, the *Key* of *newObject* must be the same as that of the existing *ReferencedObject*.

Defined in: OMStrongReferenceSetIterT.h

Return Value

The previous *ReferencedObject* if any, otherwise 0.

Parameters

newObject
The new *ReferencedObject*.

Class Template Arguments

ReferencedObject
The type of the contained objects.

Back to OMStrongReferenceSetIterator

OMStrongReferenceSetIterator::valid

template <class *ReferencedObject***>**
bool OMStrongReferenceSetIterator<*ReferencedObject***>::valid(void) const**

Is this OMStrongReferenceSetIterator validly positioned on a *ReferencedObject* ?

Defined in: OMStrongReferenceSetIterT.h

Return Value

true if this OMStrongReferenceSetIterator is positioned on a *ReferencedObject*, **false** otherwise.

Class Template Arguments

ReferencedObject
The type of the contained objects.

Back to OMStrongReferenceSetIterator

OMStrongReferenceSetIterator::value

template <class *ReferencedObject***>**
ReferencedObject* OMStrongReferenceSetIterator<*ReferencedObject***>::value(void) const**

Return the *ReferencedObject* in the associated OMStrongReferenceSetProperty at the position currently designated by this OMStrongReferenceSetIterator.

Defined in: OMStrongReferenceSetIterT.h

Return Value

The *ReferencedObject* at the current position.

Class Template Arguments

ReferencedObject
The type of the contained objects.

Back to OMStrongReferenceSetIterator

OMStrongReferenceSetIterator::~OMStrongReferenceSetIterator

template <class *ReferencedObject***>**
OMStrongReferenceSetIterator<*ReferencedObject***>::~OMStrongReferenceSetIterator(void)**

Destroy this OMStrongReferenceSetIterator.

Defined in: OMStrongReferenceSetIterT.h

Class Template Arguments

ReferencedObject
The type of the contained objects.

Back to OMStrongReferenceSetIterator

OMStrongReferenceSetProperty class

OMStrongReferenceSetProperty **class OMStrongReferenceSetProperty: public OMStrongReferenceSet**

Persistent sets of uniquely identified strongly referenced (contained) objects supported by the Object Manager. Objects are accessible by unique identifier (the key). The objects are not ordered. Duplicates objects are not allowed.

Defined in: OMStrongRefSetProperty.h

Class Template Arguments

ReferencedObject
: The type of the referenced (contained) object. This type must be a descendant of OMStorable.

UniqueIdentification
: The type of the unique key used to identify the referenced objects.

Author

Tim Bingham - tjb - (Avid Technology, Inc.)

Class Members

Public members.

OMStrongReferenceSetProperty**(const OMPropertyId propertyId, const wchar_t* name, const OMPropertyId keyPropertyId)**
: Constructor.

virtual ~OMStrongReferenceSetProperty**(void)**
: Destructor.

virtual void save**(void) const**
: Save this **OMStrongReferenceSetProperty.**

virtual void close**(void)**
: Close this **OMStrongReferenceSetProperty.**

virtual void detach**(void)**
: Detach this **OMStrongReferenceSetProperty.**

virtual void restore**(size_t externalSize)**
: Restore this **OMStrongReferenceSetProperty**, the external (persisted) size of the **OMStrongReferenceSetProperty** is *externalSize*.

size_t count**(void) const**
: The number of *ReferencedObject*s in this **OMStrongReferenceSetProperty.**

void insert**(const ReferencedObject* object)**
: Insert *object* into this **OMStrongReferenceSetProperty.**

bool ensurePresent**(const ReferencedObject* object)**
: If it is not already present, insert *object* into this **OMStrongReferenceSetProperty** and return true, otherwise return false.

void appendValue**(const ReferencedObject* object)**
: Append the given *ReferencedObject object* to this **OMStrongReferenceSetProperty.**

ReferencedObject* remove**(const UniqueIdentification& identification)**
: Remove the *ReferencedObject* identified by *identification* from this **OMStrongReferenceSetProperty.**

bool ensureAbsent**(const UniqueIdentification& identification)**
: If it is present, remove the *ReferencedObject* identified by *identification* from this **OMStrongReferenceSetProperty** and return true, otherwise return false.

void removeValue**(const ReferencedObject* object)**
: Remove *object* from this **OMStrongReferenceSetProperty.**

bool ensureAbsent**(const ReferencedObject* object)**
: If it is present, remove *object* from this **OMStrongReferenceSetProperty** and return true, otherwise return false.

bool containsValue**(const ReferencedObject* object) const**
: Does this **OMStrongReferenceSetProperty** contain *object* ?

virtual bool contains**(const UniqueIdentification& identification) const**

Does this **OMStrongReferenceSetProperty** contain a *ReferencedObject* identified by *identification*?

ReferencedObject* value**(const UniqueIdentification& identification) const**

The *ReferencedObject* in this **OMStrongReferenceSetProperty** identified by *identification*.

virtual bool find**(const UniqueIdentification& identification, ReferencedObject*& object) const**

Find the *ReferencedObject* in this **OMStrongReferenceSetProperty** identified by *identification*. If the object is found it is returned in *object* and the result is true. If the element is not found the result is false.

virtual bool isVoid**(void) const**

Is this **OMStrongReferenceSetProperty** void ?

virtual void removeProperty**(void)**

Remove this optional **OMStrongReferenceSetProperty.**

virtual size_t bitsSize**(void) const**

The size of the raw bits of this **OMStrongReferenceSetProperty.** The size is given in bytes.

virtual void getBits**(OMByte* bits, size_t size) const**

Get the raw bits of this **OMStrongReferenceSetProperty.** The raw bits are copied to the buffer at address *bits* which is *size* bytes in size.

virtual void setBits**(const OMByte* bits, size_t size)**

Set the raw bits of this **OMStrongReferenceSetProperty.** The raw bits are copied from the buffer at address *bits* which is *size* bytes in size.

virtual void insertObject**(const OMObject* object)**

Insert *object* into this **OMStrongReferenceSetProperty.**

virtual bool containsObject**(const OMObject* object) const**

Does this **OMStrongReferenceSetProperty** contain *object* ?

virtual void removeObject**(const OMObject* object)**

Remove *object* from this **OMStrongReferenceSetProperty.**

virtual void removeAllObjects**(void)**

Remove all objects from this **OMStrongReferenceSetProperty.**

virtual OMReferenceContainerIterator* createIterator**(void) const**

Create an OMReferenceContainerIterator over this **OMStrongReferenceSetProperty.**

virtual OMObject* remove**(void* identification)**

Remove the OMObject identified by *identification* from this **OMStrongReferenceSetProperty.**

virtual bool contains**(void* identification) const**

Does this **OMStrongReferenceSetProperty** contain an OMObject identified by *identification* ?

virtual bool findObject**(void* identification, OMObject*& object) const**

Find the OMObject in this **OMStrongReferenceSetProperty** identified by *identification*. If the object is found it is returned in *object* and the result is **true** . If the object is not found the result is **false** .

OMPropertyId _keyPropertyId

The id of the property whose value is the unique identifier of objects in this set.

OMStrongReferenceSetProperty::appendValue

template <class *ReferencedObject***>**
void OMStrongReferenceSetProperty<*ReferencedObject***>::appendValue(const ReferencedObject*** *object***)**

Append the given *ReferencedObject object* to this OMStrongReferenceSetProperty.

Defined in: OMStrongRefSetPropertyT.h

Parameters

object
: A pointer to a *ReferencedObject*.

Class Template Arguments

ReferencedObject
: The type of the referenced (contained) object. This type must be a descendant of OMStorable.

Back to OMStrongReferenceSetProperty

OMStrongReferenceSetProperty::bitsSize

template <class *ReferencedObject***>**
size_t OMStrongReferenceSetProperty<*ReferencedObject***>::bitsSize(void) const**

The size of the raw bits of this OMStrongReferenceSetProperty. The size is given in bytes.

Defined in: OMStrongRefSetPropertyT.h

Return Value

The size of the raw bits of this OMStrongReferenceSetProperty in bytes.

Class Template Arguments

ReferencedObject
: The type of the referenced (contained) object. This type must be a descendant of OMStorable and **OMUnique** .

Back to OMStrongReferenceSetProperty

OMStrongReferenceSetProperty::close

template <class *ReferencedObject***>**
void OMStrongReferenceSetProperty<*ReferencedObject***>::close(void)**

Close this OMStrongReferenceSetProperty.

Defined in: OMStrongRefSetPropertyT.h

Class Template Arguments

ReferencedObject
: The type of the referenced (contained) object. This type must be a descendant of OMStorable and **OMUnique** .

Back to OMStrongReferenceSetProperty

OMStrongReferenceSetProperty::contains

template <class *ReferencedObject***>**
bool OMStrongReferenceSetProperty<*ReferencedObject***>::contains(const UniqueIdentification&** *identification***)**

Does this OMStrongReferenceSetProperty contain a *ReferencedObject* identified by *identification*?

Defined in: OMStrongRefSetPropertyT.h

Return Value

True if the object is found, false otherwise.

Parameters

identification
: The unique identification of the desired object, the search key.

Class Template Arguments

ReferencedObject
: The type of the referenced (contained) object. This type must be a descendant of OMStorable.

Back to OMStrongReferenceSetProperty

OMStrongReferenceSetProperty::contains

template <class *ReferencedObject***>**
bool OMStrongReferenceSetProperty<*ReferencedObject***>::contains(void*** *identification***) const**

Does this OMStrongReferenceSetProperty contain an OMObject identified by *identification* ?

Defined in: OMStrongRefSetPropertyT.h

Return Value

True if the object was found, false otherwise.

Parameters

identification
: The unique identification of the object for which to search.

Class Template Arguments

ReferencedObject
: The type of the referenced (contained) object. This type must be a descendant of OMStorable.

Back to OMStrongReferenceSetProperty

OMStrongReferenceSetProperty::containsObject

template <class *ReferencedObject***>**
bool OMStrongReferenceSetProperty<*ReferencedObject***>::containsObject(const OMObject*** *object***) const**

Does this OMStrongReferenceSetProperty contain *object* ?

Defined in: OMStrongRefSetPropertyT.h

Return Value

True if *object* is present, false otherwise.

Parameters

object
: The OMObject for which to search.

Class Template Arguments

ReferencedObject
: The type of the referenced (contained) object. This type must be a descendant of OMStorable and **OMUnique** .

Back to OMStrongReferenceSetProperty

OMStrongReferenceSetProperty::containsValue

template <class *ReferencedObject***>**
bool OMStrongReferenceSetProperty<*ReferencedObject***>::containsValue(const ReferencedObject*** *object***)**

Does this OMStrongReferenceSetProperty contain *object* ?

Defined in: OMStrongRefSetPropertyT.h

Parameters

object
: A pointer to a *ReferencedObject*.

Class Template Arguments

ReferencedObject
The type of the referenced (contained) object. This type must be a descendant of OMStorable.

Back to OMStrongReferenceSetProperty

OMStrongReferenceSetProperty::count

size_t OMStrongReferenceSetProperty::count(void) const

The number of *ReferencedObjects* in this OMStrongReferenceSetProperty.

Defined in: OMStrongRefSetPropertyT.h

Back to OMStrongReferenceSetProperty

OMStrongReferenceSetProperty::createIterator

template <class *ReferencedObject***>**
OMReferenceContainerIterator* OMStrongReferenceSetProperty<*ReferencedObject***>::createIterator(void) const**

Create an OMReferenceContainerIterator over this OMStrongReferenceSetProperty.

Defined in: OMStrongRefSetPropertyT.h

Return Value

An OMReferenceContainerIterator over this OMStrongReferenceSetProperty.

Class Template Arguments

ReferencedObject
The type of the referenced (contained) object. This type must be a descendant of OMStorable and **OMUnique** .

Back to OMStrongReferenceSetProperty

OMStrongReferenceSetProperty::detach

template <class *ReferencedObject***>**
void OMStrongReferenceSetProperty<*ReferencedObject***>::detach(void)**

Detach this OMStrongReferenceSetProperty.

Defined in: OMStrongRefSetPropertyT.h

Class Template Arguments

ReferencedObject
The type of the referenced (contained) object. This type must be a descendant of OMStorable and **OMUnique** .

Back to OMStrongReferenceSetProperty

OMStrongReferenceSetProperty::ensureAbsent

template <class *ReferencedObject***>**
bool OMStrongReferenceSetProperty<*ReferencedObject***>::ensureAbsent(const ReferencedObject*** *object***)**

If it is present, remove *object* from this OMStrongReferenceSetProperty and return true, otherwise return false.

Defined in: OMStrongRefSetPropertyT.h

Return Value

True if the object was removed, false if it was already absent.

Parameters

object
The object to remove.

Class Template Arguments

ReferencedObject
The type of the referenced (contained) object. This type must be a descendant of OMStorable.

Back to OMStrongReferenceSetProperty

OMStrongReferenceSetProperty::ensureAbsent

template <class *ReferencedObject***>**
bool OMStrongReferenceSetProperty<*ReferencedObject***>::ensureAbsent(const UniqueIdentification&** *identification***)**

If it is present, remove the *ReferencedObject* identified by *identification* from this OMStrongReferenceSetProperty and return true, otherwise return false.

Defined in: OMStrongRefSetPropertyT.h

Return Value

True if the object was removed, false if it was already absent.

Parameters

identification
The object to remove.

Class Template Arguments

ReferencedObject
The type of the referenced (contained) object. This type must be a descendant of OMStorable.

Back to OMStrongReferenceSetProperty

OMStrongReferenceSetProperty::ensurePresent

template <class *ReferencedObject***>**
bool OMStrongReferenceSetProperty<*ReferencedObject***>::ensurePresent(const ReferencedObject*** *object***)**

If it is not already present, insert *object* into this OMStrongReferenceSetProperty and return true, otherwise return false.

Defined in: OMStrongRefSetPropertyT.h

Return Value

True if the object was inserted, false if it was already present.

Parameters

object
The object to insert.

Class Template Arguments

ReferencedObject
The type of the referenced (contained) object. This type must be a descendant of OMStorable.

Back to OMStrongReferenceSetProperty

OMStrongReferenceSetProperty::find

template <class *ReferencedObject***>**
bool OMStrongReferenceSetProperty<*ReferencedObject***>::find(const UniqueIdentification&** *identification***, ReferencedObject*&** *object***) const**

Find the *ReferencedObject* in this OMStrongReferenceSetProperty identified by *identification*. If the object is found it is returned in *object* and the result is true. If the element is not found the result is false.

Defined in: OMStrongRefSetPropertyT.h

Return Value

True if the object is found, false otherwise.

Parameters

identification
: The unique identification of the desired object, the search key.

object
: A pointer to a *ReferencedObject* by reference.

Class Template Arguments

ReferencedObject
: The type of the referenced (contained) object. This type must be a descendant of OMStorable.

Back to OMStrongReferenceSetProperty

OMStrongReferenceSetProperty::findObject

template <class *ReferencedObject***>**
bool OMStrongReferenceSetProperty<*ReferencedObject***>::findObject(void*** *identification*, **OMObject*&** *object***) const**

Find the OMObject in this OMStrongReferenceSetProperty identified by *identification*. If the object is found it is returned in *object* and the result is **true** . If the object is not found the result is **false** .

Defined in: OMStrongRefSetPropertyT.h

Return Value

True if the object was found, false otherwise.

Parameters

identification
: The unique identification of the object for which to search.

object
: The object.

Class Template Arguments

ReferencedObject

The type of the referenced (contained) object. This type must be a descendant of OMStorable.

Back to OMStrongReferenceSetProperty

OMStrongReferenceSetProperty::getBits

template <class *ReferencedObject***>**
void OMStrongReferenceSetProperty<*ReferencedObject***>::getBits(OMByte*** *bits***, size_t** *ANAME***) const**

Get the raw bits of this OMStrongReferenceSetProperty. The raw bits are copied to the buffer at address *bits* which is *size* bytes in size.

Defined in: OMStrongRefSetPropertyT.h

Parameters

bits
The address of the buffer into which the raw bits are copied.

ANAME
The size of the buffer.

Class Template Arguments

ReferencedObject
The type of the referenced (contained) object. This type must be a descendant of OMStorable and **OMUnique** .

Back to OMStrongReferenceSetProperty

OMStrongReferenceSetProperty::insert

template <class *ReferencedObject***>**
void OMStrongReferenceSetProperty<*ReferencedObject***>::insert(const ReferencedObject*** *object***)**

Insert *object* into this OMStrongReferenceSetProperty.

Defined in: OMStrongRefSetPropertyT.h

Parameters

object
The object to insert.

Class Template Arguments

ReferencedObject

The type of the referenced (contained) object. This type must be a descendant of OMStorable.

Back to OMStrongReferenceSetProperty

OMStrongReferenceSetProperty::insertObject

template <class *ReferencedObject***>**
void OMStrongReferenceSetProperty<*ReferencedObject***>::insertObject(const OMObject*** *object***)**

Insert *object* into this OMStrongReferenceSetProperty.

Defined in: OMStrongRefSetPropertyT.h

Parameters

object
The OMObject to insert.

Class Template Arguments

ReferencedObject
The type of the referenced (contained) object. This type must be a descendant of OMStorable and **OMUnique** .

Back to OMStrongReferenceSetProperty

OMStrongReferenceSetProperty::isVoid

template <class *ReferencedObject***>**
bool OMStrongReferenceSetProperty<*ReferencedObject***>::isVoid(void) const**

Is this OMStrongReferenceSetProperty void ?

Defined in: OMStrongRefSetPropertyT.h

Return Value

True if this OMStrongReferenceSetProperty is void, false otherwise.

Class Template Arguments

ReferencedObject
The type of the referenced (contained) object. This type must be a descendant of OMStorable and **OMUnique** .

Back to OMStrongReferenceSetProperty

OMStrongReferenceSetProperty::OMStrongReferenceSetProperty

OMStrongReferenceSetProperty::OMStrongReferenceSetProperty(const OMPropertyId *propertyId*, **const wchar_t*** *name*, **const OMPropertyId** *keyPropertyId***)**

Constructor.

Defined in: OMStrongRefSetPropertyT.h

Parameters

propertyId
: The property id.

name
: The name of this OMStrongReferenceSetProperty.

keyPropertyId
: The id of the property whose value is the unique identifier of objects in this set.

Back to OMStrongReferenceSetProperty

OMStrongReferenceSetProperty::remove

template <class *ReferencedObject***>**
OMObject* OMStrongReferenceSetProperty<*ReferencedObject***>::remove(void*** *identification***)**

Remove the OMObject identified by *identification* from this OMStrongReferenceSetProperty.

Defined in: OMStrongRefSetPropertyT.h

Return Value

The object that was removed.

Parameters

identification
: The unique identification of the object to remove.

Class Template Arguments

ReferencedObject
: The type of the referenced (contained) object. This type must be a descendant of OMStorable.

Back to OMStrongReferenceSetProperty

OMStrongReferenceSetProperty::remove

template <class *ReferencedObject***>**
ReferencedObject* OMStrongReferenceSetProperty<*ReferencedObject***>::remove(const UniqueIdentification&** *identification***)**

Remove the *ReferencedObject* identified by *identification* from this OMStrongReferenceSetProperty.

Defined in: OMStrongRefSetPropertyT.h

Return Value

A pointer to the removed *ReferencedObject*. If lazy loading is enabled and the referenced object was never loaded the value returned is 0.

Parameters

identification
: The unique identification of the object to be removed, the search key.

Class Template Arguments

ReferencedObject
: The type of the referenced (contained) object. This type must be a descendant of OMStorable.

Back to OMStrongReferenceSetProperty

OMStrongReferenceSetProperty::removeAllObjects

template <class *ReferencedObject***>**
void OMStrongReferenceSetProperty<*ReferencedObject***>::removeAllObjects(void)**

Remove all objects from this OMStrongReferenceSetProperty.

Defined in: OMStrongRefSetPropertyT.h

Class Template Arguments

ReferencedObject
: The type of the referenced (contained) object. This type must be a descendant of OMStorable and **OMUnique** .

Back to OMStrongReferenceSetProperty

OMStrongReferenceSetProperty::removeObject

template <class *ReferencedObject***>**
void OMStrongReferenceSetProperty<*ReferencedObject***>::removeObject(const OMObject*** *object***)**

Remove *object* from this OMStrongReferenceSetProperty.

Defined in: OMStrongRefSetPropertyT.h

Parameters

object
: The OMObject to remove.

Class Template Arguments

ReferencedObject
: The type of the referenced (contained) object. This type must be a descendant of OMStorable and **OMUnique** .

Back to OMStrongReferenceSetProperty

OMStrongReferenceSetProperty::removeProperty

template <class *ReferencedObject***>**
void OMStrongReferenceSetProperty<*ReferencedObject***>::removeProperty(void)**

Remove this optional OMStrongReferenceSetProperty.

Defined in: OMStrongRefSetPropertyT.h

Class Template Arguments

ReferencedObject
: The type of the referenced (contained) object. This type must be a descendant of OMStorable and **OMUnique** .

Back to OMStrongReferenceSetProperty

OMStrongReferenceSetProperty::removeValue

template <class *ReferencedObject***>**
void OMStrongReferenceSetProperty<*ReferencedObject***>::removeValue(const ReferencedObject*** *object***)**

Remove *object* from this OMStrongReferenceSetProperty.

Defined in: OMStrongRefSetPropertyT.h

Parameters

object
: A pointer to a *ReferencedObject*.

Class Template Arguments

ReferencedObject
The type of the referenced (contained) object. This type must be a descendant of OMStorable.

Back to OMStrongReferenceSetProperty

OMStrongReferenceSetProperty::restore

template <class *ReferencedObject***>**
void OMStrongReferenceSetProperty<*ReferencedObject***>::restore(size_t** *externalSize***)**

Restore this OMStrongReferenceSetProperty, the external (persisted) size of the OMStrongReferenceSetProperty is *externalSize*.

Defined in: OMStrongRefSetPropertyT.h

Parameters

externalSize
The external (persisted) size of the OMStrongReferenceSetProperty.

Class Template Arguments

ReferencedObject
The type of the referenced (contained) object. This type must be a descendant of OMStorable and **OMUnique** .

Back to OMStrongReferenceSetProperty

OMStrongReferenceSetProperty::save

template <class *ReferencedObject***>**
void OMStrongReferenceSetProperty<*ReferencedObject***>::save(void) const**

Save this OMStrongReferenceSetProperty.

Defined in: OMStrongRefSetPropertyT.h

Class Template Arguments

ReferencedObject
The type of the referenced (contained) object. This type must be a descendant of OMStorable and **OMUnique** .

Back to OMStrongReferenceSetProperty

OMStrongReferenceSetProperty::setBits

template <class *ReferencedObject***>**
void OMStrongReferenceSetProperty<*ReferencedObject***>::setBits(const OMByte*** *bits***, size_t** *size***)**

Set the raw bits of this OMStrongReferenceSetProperty. The raw bits are copied from the buffer at address *bits* which is *size* bytes in size.

Defined in: OMStrongRefSetPropertyT.h

Parameters

bits
: The address of the buffer from which the raw bits are copied.

size
: The size of the buffer.

Class Template Arguments

ReferencedObject
: The type of the referenced (contained) object. This type must be a descendant of OMStorable and **OMUnique** .

Back to OMStrongReferenceSetProperty

OMStrongReferenceSetProperty::value

template <class *ReferencedObject***>**
ReferencedObject* OMStrongReferenceSetProperty<*ReferencedObject***>::value(const UniqueIdentification&** *identification***) const**

The *ReferencedObject* in this OMStrongReferenceSetProperty identified by *identification*.

Defined in: OMStrongRefSetPropertyT.h

Return Value

A pointer to the *ReferencedObject*.

Parameters

identification
: The unique identification of the desired object, the search key.

Class Template Arguments

ReferencedObject
: The type of the referenced (contained) object. This type must be a descendant of OMStorable.

Back to OMStrongReferenceSetProperty

OMStrongReferenceSetProperty::~OMStrongReferenceSetProperty

OMStrongReferenceSetProperty::~OMStrongReferenceSetProperty(void)

Destructor.

Defined in: OMStrongRefSetPropertyT.h

Back to OMStrongReferenceSetProperty

OMStrongReferenceVector class

OMStrongReferenceVector **class OMStrongReferenceVector: public OMReferenceVectorProperty**

Persistent elastic sequential collections of strongly referenced (contained) objects supported by the Object Manager. Objects are accessible by index. The order of objects is determined externally. Duplicate objects are allowed.

Defined in: OMStrongReferenceVector.h

Author

Tim Bingham - tjb - (Avid Technology, Inc.)

Class Members

Public members.

OMStrongReferenceVector**(const OMPropertyId propertyId, const wchar_t* name)**
: Constructor.

virtual ~OMStrongReferenceVector**(void)**
: Destructor.

virtual void grow(const size_t capacity)
: Increase the capacity of this **OMStrongReferenceVector** so that it can contain at least *capacity OMObject*s without having to be resized.

OMStrongReferenceVector::OMStrongReferenceVector

OMStrongReferenceVector::OMStrongReferenceVector(const OMPropertyId *propertyId*, **const wchar_t*** *name***)**

Constructor.

Defined in: OMStrongReferenceVector.cpp

Parameters

propertyId

The property id.

name

The name of this OMStrongReferenceVector.

Back to OMStrongReferenceVector

OMStrongReferenceVector::~OMStrongReferenceVector

OMStrongReferenceVector::~OMStrongReferenceVector(void)

Destructor.

Defined in: OMStrongReferenceVector.cpp

Back to OMStrongReferenceVector

OMStrongReferenceVectorElement class

OMStrongReferenceVectorElement **class OMStrongReferenceVectorElement: public OMContainerElement**

Elements of Object Manager reference vectors.

Defined in: OMContainerElement.h

Author

Tim Bingham - tjb - (Avid Technology, Inc.)

Class Members

Public members.

OMStrongReferenceVectorElement**(void)**

Constructor.

OMStrongReferenceVectorElement**(OMProperty* property, const wchar_t* name, OMUInt32 localKey)**

Constructor.

OMStrongReferenceVectorElement**(const OMStrongReferenceVectorElement& rhs)**

Copy constructor.

~OMStrongReferenceVectorElement**(void)**

Destructor.

OMStrongReferenceVectorElement& operator=**(const OMStrongReferenceVectorElement& rhs)**

Assignment. This operator provides value semantics for OMVector. This operator does not provide assignment of object references.

bool operator==**(const OMStrongReferenceVectorElement& rhs) const**

Equality. This operator provides value semantics for OMVector. This operator does not provide equality of object references.

OMStorable* setValue**(const OMStorable* value)**

Set the value of this **OMStrongReferenceVectorElement.**

OMUInt32 localKey**(void) const**

The local key of this **OMStrongReferenceVectorElement.**

OMUInt32 _localKey

The local key of this **OMStrongReferenceVectorElement.** The key is unique only within a given container instance and is assigned to each element of the container in such way as to be independent of the element's position within the container.

OMStrongReferenceVectorElement::localKey

OMUInt32 OMStrongReferenceVectorElement::localKey(void) const

The local key of this OMStrongReferenceVectorElement.

Defined in: OMContainerElement.cpp

Return Value

The local key of this OMStrongReferenceVectorElement.

Back to OMStrongReferenceVectorElement

OMStrongReferenceVectorElement::OMStrongReferenceVectorElement

OMStrongReferenceVectorElement::OMStrongReferenceVectorElement(const OMStrongReferenceVectorElement& *rhs*)

Copy constructor.

Defined in: OMContainerElement.cpp

Parameters

rhs

The OMStrongReferenceVectorElement to copy.

Back to OMStrongReferenceVectorElement

OMStrongReferenceVectorElement::OMStrongReferenceVectorElement

OMStrongReferenceVectorElement::OMStrongReferenceVectorElement(void)

Constructor.

Defined in: OMContainerElement.cpp

Back to OMStrongReferenceVectorElement

OMStrongReferenceVectorElement::OMStrongReferenceVectorElement

OMStrongReferenceVectorElement::OMStrongReferenceVectorElement(OMProperty* *property***, const wchar_t*** *name***, OMUInt32** *localKey***)**

Constructor.

Defined in: OMContainerElement.cpp

Parameters

property
The OMProperty (a vector property) that contains this OMStrongReferenceVectorElement.
name
The name of this OMStrongReferenceVectorElement.
localKey
The local key of this OMStrongReferenceVectorElement within it's vector.
Back to OMStrongReferenceVectorElement

OMStrongReferenceVectorElement::operator=

OMStrongReferenceVectorElement& OMStrongReferenceVectorElement::operator=(const OMStrongReferenceVectorElement& *rhs***)**

Assignment. This operator provides value semantics for OMVector. This operator does not provide assignment of object references.

Defined in: OMContainerElement.cpp

Return Value

The OMStrongReferenceVectorElement resulting from the assignment.

Parameters

rhs
The OMStrongReferenceVectorElement to be assigned.
Back to OMStrongReferenceVectorElement

OMStrongReferenceVectorElement::operator==

bool OMStrongReferenceVectorElement::operator==(const OMStrongReferenceVectorElement& *rhs***)**

Equality. This operator provides value semantics for OMVector. This operator does not provide equality of object references.

Defined in: OMContainerElement.cpp

Return Value

True if the values are the same, false otherwise.

Parameters

rhs
The OMStrongReferenceVectorElement to be compared.

Back to OMStrongReferenceVectorElement

OMStrongReferenceVectorElement::setValue

OMStorable* OMStrongReferenceVectorElement::setValue(const OMStorable* *value***)**

Set the value of this **OMOMStrongReferenceVectorElement** .

Defined in: OMContainerElement.cpp

Return Value

A pointer to previous *ReferencedObject*, if any.

Parameters

value
A pointer to the new *ReferencedObject*.

Back to OMStrongReferenceVectorElement

OMStrongReferenceVectorElement::~OMStrongReferenceVectorElement

OMStrongReferenceVectorElement::~OMStrongReferenceVectorElement(void)

Destructor.

Defined in: OMContainerElement.cpp

Back to OMStrongReferenceVectorElement

OMStrongReferenceVectorIterator class

OMStrongReferenceVectorIterator **class OMStrongReferenceVectorIterator: public OMReferenceContainerIterator**

Iterators over OMStrongReferenceVectorPropertys.

Defined in: OMStrongReferenceVectorIter.h

Class Template Arguments

ReferencedObject
The type of the contained objects.

Author

Tim Bingham - tjb - (Avid Technology, Inc.)

Class Members

Public members.

OMStrongReferenceVectorIterator(const OMStrongReferenceVectorProperty&ltReferencedObject>& vector, OMIteratorPosition initialPosition = OMBefore)
Create an **OMStrongReferenceVectorIterator** over the given OMStrongReferenceVectorProperty *vector* and initialize it to the given *initialPosition*. If *initialPosition* is specified as OMBefore then this **OMStrongReferenceVectorIterator** is made ready to traverse the associated OMStrongReferenceVectorProperty in the forward direction (increasing indexes). If *initialPosition* is specified as OMAfter then this **OMStrongReferenceVectorIterator** is made ready to traverse the associated OMStrongReferenceVectorProperty in the reverse direction (decreasing indexes).

virtual OMReferenceContainerIterator* copy(void) const
Create a copy of this **OMStrongReferenceVectorIterator.**

virtual ~OMStrongReferenceVectorIterator(void)
Destroy this **OMStrongReferenceVectorIterator.**

virtual void reset(OMIteratorPosition initialPosition = OMBefore)
Reset this **OMStrongReferenceVectorIterator** to the given *initialPosition*. If *initialPosition* is specified as OMBefore then this **OMStrongReferenceVectorIterator** is made ready to traverse the associated OMStrongReferenceVectorProperty in the forward direction (increasing indexes). If *initialPosition* is specified as OMAfter then this **OMStrongReferenceVectorIterator** is made ready to traverse the associated OMStrongReferenceVectorProperty in the reverse direction (decreasing indexes).

virtual bool before(void) const
Is this **OMStrongReferenceVectorIterator** positioned before the first *ReferencedObject* ?

virtual bool after(void) const
Is this **OMStrongReferenceVectorIterator** positioned after the last *ReferencedObject* ?

virtual bool valid(void) const
Is this **OMStrongReferenceVectorIterator** validly positioned on a *ReferencedObject* ?

virtual size_t count(void) const
The number of *ReferencedObject*s in the associated OMStrongReferenceVectorProperty.

virtual bool operator++()
Advance this **OMStrongReferenceVectorIterator** to the next *ReferencedObject*, if any. If the end of the associated OMStrongReferenceVectorProperty is not reached then the result is **true** , **valid** becomes **true** and **after** becomes **false** . If the end of the associated OMStrongReferenceVectorProperty is reached then the result is **false** , **valid** becomes **false** and **after** becomes **true** .

virtual bool operator--()

Retreat this **OMStrongReferenceVectorIterator** to the previous *ReferencedObject*, if any. If the beginning of the associated OMStrongReferenceVectorProperty is not reached then the result is **true** , **valid** becomes **true** and **before** becomes **false** . If the beginning of the associated OMStrongReferenceVectorProperty is reached then the result is **false** , **valid** becomes **false** and **before** becomes **true** .

virtual ReferencedObject* value**(void) const**

Return the *ReferencedObject* in the associated OMStrongReferenceVectorProperty at the position currently designated by this **OMStrongReferenceVectorIterator.**

virtual ReferencedObject* setValue**(const ReferencedObject* newObject)**

Set the *ReferencedObject* in the associated OMStrongReferenceVectorProperty at the position currently designated by this **OMStrongReferenceVectorIterator** to *newObject*. The previous *ReferencedObject*, if any, is returned.

virtual ReferencedObject* clearValue**(void)**

Set the *Element* in the associated OMContainer at the position currently designated by this **OMStrongReferenceVectorIterator** to 0. The previous *ReferencedObject*, if any is returned.

virtual size_t index**(void) const**

Return the index of the *ReferencedObject* in the associated OMStrongReferenceVectorProperty at the position currently designated by this **OMStrongReferenceVectorIterator.**

virtual OMObject* currentObject**(void) const**

Return the *OMObject* in the associated reference container property at the position currently designated by this **OMStrongReferenceVectorIterator.**

OMStrongReferenceVectorIterator(const VectorIterator& iter)

Create an **OMStrongReferenceVectorIterator** given an underlying OMVectorIterator.

OMStrongReferenceVectorIterator::

template <class *ReferencedObject***>**
OMStrongReferenceVectorIterator<*ReferencedObject***>::(void)**

Destroy this OMStrongReferenceVectorIterator.

Defined in: OMStrongReferenceVectorIterT.h

Class Template Arguments

ReferencedObject

The type of the contained objects.

Back to OMStrongReferenceVectorIterator

OMStrongReferenceVectorIterator::

template <class *ReferencedObject***>**
OMStrongReferenceVectorIterator<*ReferencedObject***>::(** *OMStrongReferenceVectorIterator***)**

Create an OMStrongReferenceVectorIterator given an underlying OMVectorIterator.

Defined in: OMStrongReferenceVectorIterT.h

Parameters

OMStrongReferenceVectorIterator
The underlying OMVectorIterator.

Class Template Arguments

ReferencedObject
The type of the contained objects.

Back to OMStrongReferenceVectorIterator

OMStrongReferenceVectorIterator::

template <class *ReferencedObject***>**
OMStrongReferenceVectorIterator<*ReferencedObject***>::(** *OMStrongReferenceVectorIterator***, const OMStrongReferenceVectorProperty&ltReferencedObject>&** *vector***)**

Create an OMStrongReferenceVectorIterator over the given OMStrongReferenceVectorProperty *vector* and initialize it to the given *initialPosition*. If *initialPosition* is specified as OMBefore then this OMStrongReferenceVectorIterator is made ready to traverse the associated OMStrongReferenceVectorProperty in the forward direction (increasing indexes). If *initialPosition* is specified as OMAfter then this OMStrongReferenceVectorIterator is made ready to traverse the associated OMStrongReferenceVectorProperty in the reverse direction (decreasing indexes).

Defined in: OMStrongReferenceVectorIterT.h

Parameters

OMStrongReferenceVectorIterator
The OMStrongReferenceVector over which to iterate.

vector
The initial position for this OMStrongReferenceVectorIterator.

Class Template Arguments

ReferencedObject
The type of the contained objects.

Back to OMStrongReferenceVectorIterator

OMStrongReferenceVectorIterator::after

template <class *ReferencedObject***>**
bool OMStrongReferenceVectorIterator<*ReferencedObject***>::after(void) const**

Is this OMStrongReferenceVectorIterator positioned after the last *ReferencedObject* ?

Defined in: OMStrongReferenceVectorIterT.h

Return Value

true if this OMStrongReferenceVectorIterator is positioned after the last *ReferencedObject*, **false** otherwise.

Class Template Arguments

ReferencedObject
The type of the contained objects.

Back to OMStrongReferenceVectorIterator

OMStrongReferenceVectorIterator::before

template <class *ReferencedObject***>**
bool OMStrongReferenceVectorIterator<*ReferencedObject***>::before(void) const**

Is this OMStrongReferenceVectorIterator positioned before the first *ReferencedObject* ?

Defined in: OMStrongReferenceVectorIterT.h

Return Value

true if this OMStrongReferenceVectorIterator is positioned before the first *ReferencedObject*, **false** otherwise.

Class Template Arguments

ReferencedObject
The type of the contained objects.

Back to OMStrongReferenceVectorIterator

OMStrongReferenceVectorIterator::clearValue

template <class *ReferencedObject***>**
ReferencedObject* OMStrongReferenceVectorIterator<*ReferencedObject***>::clearValue(void)**

Set the *ReferencedObject* in the associated OMStrongReferenceVectorProperty at the position currently designated by this OMStrongReferenceVectorIterator to 0. The previous *ReferencedObject*, if any, is returned.

Defined in: OMStrongReferenceVectorIterT.h

Return Value

The previous *ReferencedObject* if any, otherwise 0.

Class Template Arguments

ReferencedObject
The type of the contained objects.
Back to OMStrongReferenceVectorIterator

OMStrongReferenceVectorIterator::copy

template <class *ReferencedObject***>**
OMReferenceContainerIterator* OMStrongReferenceVectorIterator<*ReferencedObject***>::copy(void) const**

Create a copy of this OMStrongReferenceVectorIterator.

Defined in: OMStrongReferenceVectorIterT.h

Return Value

The new OMStrongReferenceVectorIterator.

Class Template Arguments

ReferencedObject
The type of the contained objects.
Back to OMStrongReferenceVectorIterator

OMStrongReferenceVectorIterator::count

template <class *ReferencedObject***>**
size_t OMStrongReferenceVectorIterator<*ReferencedObject***>::count(void) const**

The number of *ReferencedObject*s in the associated OMStrongReferenceVectorProperty.

Defined in: OMStrongReferenceVectorIterT.h

Return Value

The number of *ReferencedObject*s

Class Template Arguments

ReferencedObject
The type of the contained objects.
Back to OMStrongReferenceVectorIterator

OMStrongReferenceVectorIterator::currentObject

template <class *ReferencedObject***>**
OMObject* OMStrongReferenceVectorIterator<*ReferencedObject***>::currentObject(void) const**

Return the OMObject in the associated OMStrongReferenceVectorProperty at the position currently designated by this OMStrongReferenceVectorIterator.

Defined in: OMStrongReferenceVectorIterT.h

Return Value

The OMObject at the current position.

Class Template Arguments

ReferencedObject
The type of the contained objects.

Back to OMStrongReferenceVectorIterator

OMStrongReferenceVectorIterator::index

template <class *Element***>**
size_t OMStrongReferenceVectorIterator<*Element***>::index(void) const**

Return the index of the *ReferencedObject* in the associated OMStrongReferenceVectorProperty at the position currently designated by this OMStrongReferenceVectorIterator.

Defined in: OMStrongReferenceVectorIterT.h

Return Value

The index of the current position.

Class Template Arguments

Element
The type of the contained elements.

Back to OMStrongReferenceVectorIterator

OMStrongReferenceVectorIterator::operator++

template <class *ReferencedObject***>**
bool OMStrongReferenceVectorIterator<*ReferencedObject***>::operator++(void)**

Advance this OMStrongReferenceVectorIterator to the next *ReferencedObject*, if any. If the end of the associated OMStrongReferenceVectorProperty is not reached then the result is **true** , **valid** becomes **true** and **after** becomes **false** . If the end of the associated OMStrongReferenceVectorProperty is reached then the result is **false** , **valid** becomes **false** and **after** becomes **true** .

Defined in: OMStrongReferenceVectorIterT.h

Return Value

false if this OMStrongReferenceVectorIterator has passed the last *ReferencedObject*, **true** otherwise.

Class Template Arguments

ReferencedObject
The type of the contained objects.

Back to OMStrongReferenceVectorIterator

OMStrongReferenceVectorIterator::operator--

template <class *ReferencedObject***>**
bool OMStrongReferenceVectorIterator<*ReferencedObject***>::operator--(void)**

Retreat this OMStrongReferenceVectorIterator to the previous *ReferencedObject*, if any. If the beginning of the associated OMStrongReferenceVectorProperty is not reached then the result is **true** , **valid** becomes **true** and **before** becomes **false** . If the beginning of the associated OMStrongReferenceVectorProperty is reached then the result is **false** , **valid** becomes **false** and **before** becomes **true** .

Defined in: OMStrongReferenceVectorIterT.h

Return Value

false if this OMStrongReferenceVectorIterator has passed the first *ReferencedObject*, **true** otherwise.

Class Template Arguments

ReferencedObject
The type of the contained objects.

Back to OMStrongReferenceVectorIterator

OMStrongReferenceVectorIterator::reset

template <class *ReferencedObject***>**
void OMStrongReferenceVectorIterator<*ReferencedObject***>::reset(OMIteratorPosition** *initialPosition***)**

Reset this OMStrongReferenceVectorIterator to the given *initialPosition*. If *initialPosition* is specified as OMBefore then this OMStrongReferenceVectorIterator is made ready to traverse the associated OMStrongReferenceVectorProperty in the forward direction (increasing indexes). If *initialPosition* is specified as OMAfter then this OMStrongReferenceVectorIterator is made ready to traverse the associated OMStrongReferenceVectorProperty in the reverse direction (decreasing indexes).

Defined in: OMStrongReferenceVectorIterT.h

Parameters

initialPosition
The position to which this OMStrongReferenceVectorIterator should be reset.

Class Template Arguments

ReferencedObject
The type of the contained objects.

Back to OMStrongReferenceVectorIterator

OMStrongReferenceVectorIterator::setValue

template <class *ReferencedObject***>**
ReferencedObject* OMStrongReferenceVectorIterator<*ReferencedObject***>::setValue(const ReferencedObject*** *newObject***)**

Set the *ReferencedObject* in the associated OMStrongReferenceVectorProperty at the position currently designated by this OMStrongReferenceVectorIterator to *newObject*. The previous *ReferencedObject*, if any, is returned.

Defined in: OMStrongReferenceVectorIterT.h

Return Value

The previous *ReferencedObject* if any, otherwise 0.

Parameters

newObject
The new *ReferencedObject*.

Class Template Arguments

ReferencedObject
The type of the contained objects.

Back to OMStrongReferenceVectorIterator

OMStrongReferenceVectorIterator::valid

template <class *ReferencedObject***>**
bool OMStrongReferenceVectorIterator<*ReferencedObject***>::valid(void) const**

Is this OMStrongReferenceVectorIterator validly positioned on a *ReferencedObject* ?

Defined in: OMStrongReferenceVectorIterT.h

Return Value

true if this OMStrongReferenceVectorIterator is positioned on a *ReferencedObject*, **false** otherwise.

Class Template Arguments

ReferencedObject
The type of the contained objects.

Back to OMStrongReferenceVectorIterator

OMStrongReferenceVectorIterator::value

template <class *ReferencedObject***>**
ReferencedObject* OMStrongReferenceVectorIterator<*ReferencedObject***>::value(void) const**

Return the *ReferencedObject* in the associated OMStrongReferenceVectorProperty at the position currently designated by this OMStrongReferenceVectorIterator.

Defined in: OMStrongReferenceVectorIterT.h

Return Value

The *ReferencedObject* at the current position.

Class Template Arguments

ReferencedObject
The type of the contained objects.

Back to OMStrongReferenceVectorIterator

OMStrongReferenceVectorProperty class

OMStrongReferenceVectorProperty **class OMStrongReferenceVectorProperty: public OMStrongReferenceVector**

Persistent elastic sequential collections of strongly referenced (contained) objects supported by the Object Manager. Objects are accessible by index. The order of objects is determined externally. Duplicate objects are allowed.

Defined in: OMStrongRefVectorProperty.h

Class Template Arguments

ReferencedObject
The type of the referenced (contained) object. This type must be a descendant of OMStorable.

Author

Tim Bingham - tjb - (Avid Technology, Inc.)

Class Members

Public members.

OMStrongReferenceVectorProperty(const OMPropertyId propertyId, const wchar_t* name)

Constructor.

virtual ~OMStrongReferenceVectorProperty(void)

Destructor.

virtual void save**(void) const**

Save this **OMStrongReferenceVectorProperty.**

virtual void close**(void)**

Close this **OMStrongReferenceVectorProperty.**

virtual void detach**(void)**

Detach this **OMStrongReferenceVectorProperty.**

virtual void restore**(size_t externalSize)**

Restore this **OMStrongReferenceVectorProperty**, the external (persisted) size of the **OMStrongReferenceVectorProperty** is *externalSize*.

size_t count**(void) const**

The number of *ReferencedObject*s in this **OMStrongReferenceVectorProperty.**

ReferencedObject* setValueAt**(const ReferencedObject* object, const size_t index)**

Set the value of this **OMStrongReferenceVectorProperty** at position *index* to *object*.

ReferencedObject* clearValueAt**(const size_t index)**

Set the value of this **OMStrongReferenceVectorProperty** at position *index* to 0.

ReferencedObject* valueAt**(const size_t index) const**

The value of this **OMStrongReferenceVectorProperty** at position *index*.

void getValueAt**(ReferencedObject*& object, const size_t index) const**

Get the value of this **OMStrongReferenceVectorProperty** at position *index* into *object*.

bool find**(const size_t index, ReferencedObject*& object) const**

If *index* is valid, get the value of this **OMStrongReferenceVectorProperty** at position *index* into *object* and return true, otherwise return false.

void appendValue**(const ReferencedObject* object)**

Append the given *ReferencedObject object* to this **OMStrongReferenceVectorProperty.**

void prependValue**(const ReferencedObject* object)**

Prepend the given *ReferencedObject object* to this **OMStrongReferenceVectorProperty.**

void insert**(const ReferencedObject* object)**

Insert *object* into this **OMStrongReferenceVectorProperty**. This function is redefined from OMContainerProperty as **appendValue.**

void insertAt**(const ReferencedObject* object, const size_t index)**

Insert *object* into this **OMStrongReferenceVectorProperty** at position *index*. Existing objects at *index* and higher are shifted up one index position.

bool containsValue**(const ReferencedObject* object) const**

Does this **OMStrongReferenceVectorProperty** contain *object* ?

void removeValue**(const ReferencedObject* object)**

Remove *object* from this **OMStrongReferenceVectorProperty.**

ReferencedObject* removeAt**(const size_t index)**

Remove the object from this **OMStrongReferenceVectorProperty** at position *index*. Existing objects in this **OMStrongReferenceVectorProperty** at *index* + 1 and higher are shifted down one index position.

ReferencedObject* removeLast**(void)**

Remove the last (index == count() - 1) object from this **OMStrongReferenceVectorProperty.**

ReferencedObject* removeFirst**(void)**

Remove the first (index == 0) object from this **OMStrongReferenceVectorProperty**. Existing objects in this **OMStrongReferenceVectorProperty** are shifted down one index position.

size_t indexOfValue**(const ReferencedObject* object) const**

The index of the *ReferencedObject* object*.

size_t countOfValue**(const ReferencedObject* object) const**

The number of occurrences of *object* in this **OMStrongReferenceVectorProperty.**

bool containsIndex**(const size_t index) const**

Does this **OMStrongReferenceVectorProperty** contain *index* ? Is *index* valid ?

bool findIndex**(const ReferencedObject* object, size_t& index) const**

If this OMStrongReferenceProperty contains *object* then place its index in *index* and return true, otherwise return false.

virtual void grow**(const size_t capacity)**

Increase the capacity of this **OMStrongReferenceVectorProperty** so that it can contain at least *capacity ReferencedObject*s without having to be resized.

virtual bool isVoid**(void) const**

Is this **OMStrongReferenceVectorProperty** void ?

virtual void removeProperty**(void)**

Remove this optional **OMStrongReferenceVectorProperty.**

virtual size_t bitsSize**(void) const**

The size of the raw bits of this **OMStrongReferenceVectorProperty.** The size is given in bytes.

virtual void getBits**(OMByte* bits, size_t size) const**

Get the raw bits of this **OMStrongReferenceVectorProperty.** The raw bits are copied to the buffer at address *bits* which is *size* bytes in size.

virtual void setBits**(const OMByte* bits, size_t size)**

Set the raw bits of this **OMStrongReferenceVectorProperty.** The raw bits are copied from the buffer at address *bits* which is *size* bytes in size.

virtual void insertObject**(const OMObject* object)**

Insert *object* into this **OMStrongReferenceVectorProperty.**

virtual bool containsObject**(const OMObject* object) const**

Does this **OMStrongReferenceVectorProperty** contain *object* ?

virtual void removeObject**(const OMObject* object)**

Remove *object* from this **OMStrongReferenceVectorProperty.**

virtual void removeAllObjects**(void)**

Remove all objects from this **OMStrongReferenceVectorProperty.**

virtual OMReferenceContainerIterator* createIterator**(void) const**

Create an OMReferenceContainerIterator over this **OMStrongReferenceVectorProperty.**

virtual OMObject* setObjectAt**(const OMObject* object, const size_t index)**

Set the value of this **OMStrongReferenceVectorProperty** at position *index* to *object*.

virtual OMObject* getObjectAt**(const size_t index) const**

The value of this **OMStrongReferenceVectorProperty** at position *index*.

virtual void appendObject**(const OMObject* object)**

Append the given *OMObject object* to this **OMStrongReferenceVectorProperty.**

virtual void prependObject**(const OMObject* object)**

Prepend the given *OMObject object* to this **OMStrongReferenceVectorProperty.**

virtual OMObject* removeObjectAt**(const size_t index)**

Remove the object from this **OMStrongReferenceVectorProperty** at position *index*. Existing objects in this **OMStrongReferenceVectorProperty** at *index* + 1 and higher are shifted down one index position.

virtual void insertObjectAt**(const OMObject* object, const size_t index)**

Insert *object* into this **OMStrongReferenceVectorProperty** at position *index*. Existing objects at *index* and higher are shifted up one index position.

OMStrongReferenceVectorProperty::

OMStrongReferenceVectorProperty::(*OMStrongReferenceVectorProperty*, **const OMPropertyId** *propertyId***)**

Constructor.

Defined in: OMStrongRefVectorPropertyT.h

Parameters

OMStrongReferenceVectorProperty
: The property id.

propertyId
: The name of this OMStrongReferenceVectorProperty.

Back to OMStrongReferenceVectorProperty

OMStrongReferenceVectorProperty::

OMStrongReferenceVectorProperty::(void)

Destructor.

Defined in: OMStrongRefVectorPropertyT.h

Back to OMStrongReferenceVectorProperty

OMStrongReferenceVectorProperty::appendObject

template <class *ReferencedObject***>**
void OMStrongReferenceVectorProperty<*ReferencedObject***>::appendObject(const OMObject*** *object***)**

Append the given *OMObject object* to this OMStrongReferenceVectorProperty.

Defined in: OMStrongRefVectorPropertyT.h

Parameters

object
: The object to append.

Class Template Arguments

ReferencedObject
: The type of the referenced (contained) object. This type must be a descendant of OMStorable.

Back to OMStrongReferenceVectorProperty

OMStrongReferenceVectorProperty::appendValue

template <class *ReferencedObject***>**
void OMStrongReferenceVectorProperty<*ReferencedObject***>::appendValue(const ReferencedObject*** *object***)**

Append the given *ReferencedObject object* to this OMStrongReferenceVectorProperty.

Defined in: OMStrongRefVectorPropertyT.h

Parameters

object
: A pointer to a *ReferencedObject*.

Class Template Arguments

ReferencedObject
: The type of the referenced (contained) object. This type must be a descendant of OMStorable.

Back to OMStrongReferenceVectorProperty

OMStrongReferenceVectorProperty::bitsSize

template <class *ReferencedObject***>**
size_t OMStrongReferenceVectorProperty<*ReferencedObject***>::bitsSize(void) const**

The size of the raw bits of this OMStrongReferenceVectorProperty. The size is given in bytes.

Defined in: OMStrongRefVectorPropertyT.h

Return Value

The size of the raw bits of this OMStrongReferenceVectorProperty in bytes.

Class Template Arguments

ReferencedObject
: The type of the referenced (contained) object. This type must be a descendant of OMStorable.

Back to OMStrongReferenceVectorProperty

OMStrongReferenceVectorProperty::clearValueAt

template <class *ReferencedObject***>**
ReferencedObject* OMStrongReferenceVectorProperty<*ReferencedObject***>::clearValueAt(const size_t** *index***)**

Set the value of this OMStrongReferenceVectorProperty at position *index* to 0.

Defined in: OMStrongRefVectorPropertyT.h

Return Value

A pointer to the old *ReferencedObject*. If lazy loading is enabled and the referenced object was never loaded the value returned is 0.

Parameters

index
: The position to clear.

Class Template Arguments

ReferencedObject
: The type of the referenced (contained) object. This type must be a descendant of OMStorable.

Back to OMStrongReferenceVectorProperty

OMStrongReferenceVectorProperty::close

template <class *ReferencedObject***>**
void OMStrongReferenceVectorProperty<*ReferencedObject***>::close(void)**

Close this OMStrongReferenceVectorProperty.

Defined in: OMStrongRefVectorPropertyT.h

Class Template Arguments

ReferencedObject
: The type of the referenced (contained) object. This type must be a descendant of OMStorable.

Back to OMStrongReferenceVectorProperty

OMStrongReferenceVectorProperty::containsIndex

template <class *ReferencedObject***>**
bool OMStrongReferenceVectorProperty<*ReferencedObject***>::containsIndex(const size_t** *index***) const**

Does this OMStrongReferenceVectorProperty contain *index* ? Is *index* valid ?

Defined in: OMStrongRefVectorPropertyT.h

Return Value

True if the index is valid, false otherwise.

Parameters

index
: The index.

Class Template Arguments

ReferencedObject
: The type of the referenced (contained) object. This type must be a descendant of OMStorable.

Back to OMStrongReferenceVectorProperty

OMStrongReferenceVectorProperty::containsObject

template <class *ReferencedObject***>**
bool OMStrongReferenceVectorProperty<*ReferencedObject***>::containsObject(const OMObject*** *object***) const**

Does this OMStrongReferenceVectorProperty contain *object* ?

Defined in: OMStrongRefVectorPropertyT.h

Return Value

True if *object* is present, false otherwise.

Parameters

object
: The OMObject for which to search.

Class Template Arguments

ReferencedObject
: The type of the referenced (contained) object. This type must be a descendant of OMStorable.

Back to OMStrongReferenceVectorProperty

OMStrongReferenceVectorProperty::containsValue

template <class *ReferencedObject***>**
bool OMStrongReferenceVectorProperty<*ReferencedObject***>::containsValue(const ReferencedObject*** *object***) const**

Does this OMStrongReferenceVectorProperty contain *object* ?

Defined in: OMStrongRefVectorPropertyT.h

Parameters

object
: A pointer to a *ReferencedObject*.

Class Template Arguments

ReferencedObject
: The type of the referenced (contained) object. This type must be a descendant of OMStorable.

Back to OMStrongReferenceVectorProperty

OMStrongReferenceVectorProperty::count

size_t OMStrongReferenceVectorProperty::count(void) const

The number of *ReferencedObjects* in this OMStrongReferenceVectorProperty.

Defined in: OMStrongRefVectorPropertyT.h

Back to OMStrongReferenceVectorProperty

OMStrongReferenceVectorProperty::countOfValue

template <class *ReferencedObject***>**
size_t OMStrongReferenceVectorProperty<*ReferencedObject***>::countOfValue(const ReferencedObject*** *object***) const**

The number of occurrences of *object* in this OMStrongReferenceVectorProperty.

Defined in: OMStrongRefVectorPropertyT.h

Return Value

The number of occurrences.

Parameters

object
The object to count.

Class Template Arguments

ReferencedObject
The type of the referenced (contained) object. This type must be a descendant of OMStorable.

Back to OMStrongReferenceVectorProperty

OMStrongReferenceVectorProperty::createIterator

template <class *ReferencedObject***>**
OMReferenceContainerIterator*
OMStrongReferenceVectorProperty<*ReferencedObject***>::createIterator(void) const**

Create an OMReferenceContainerIterator over this OMStrongReferenceVectorProperty.

Defined in: OMStrongRefVectorPropertyT.h

Return Value

An OMReferenceContainerIterator over this OMStrongReferenceVectorProperty.

Class Template Arguments

ReferencedObject
The type of the referenced (contained) object. This type must be a descendant of OMStorable.

Back to OMStrongReferenceVectorProperty

OMStrongReferenceVectorProperty::detach

template <class *ReferencedObject***>**
void OMStrongReferenceVectorProperty<*ReferencedObject***>::detach(void)**

Detach this OMStrongReferenceVectorProperty.

Defined in: OMStrongRefVectorPropertyT.h

Class Template Arguments

ReferencedObject

The type of the referenced (contained) object. This type must be a descendant of OMStorable.

Back to OMStrongReferenceVectorProperty

OMStrongReferenceVectorProperty::find

template <class *ReferencedObject***>**
bool OMStrongReferenceVectorProperty<*ReferencedObject***>::find(const size_t** *index***, ReferencedObject*& *object***) const**

If *index* is valid, get the value of this OMStrongReferenceVectorProperty at position *index* into *object* and return true, otherwise return false.

Defined in: OMStrongRefVectorPropertyT.h

Return Value

True if *index* is valid, false otherwise.

Parameters

index
The position from which to get the *ReferencedObject*.

object
A pointer to a *ReferencedObject*.

Class Template Arguments

ReferencedObject
The type of the referenced (contained) object. This type must be a descendant of OMStorable.

Back to OMStrongReferenceVectorProperty

OMStrongReferenceVectorProperty::findIndex

template <class *ReferencedObject***>**
bool OMStrongReferenceVectorProperty<*ReferencedObject***>::findIndex(const ReferencedObject*** *object***, size_t&** *index***) const**

If this OMStrongReferenceProperty contains *object* then place its index in *index* and return true, otherwise return false.

Defined in: OMStrongRefVectorPropertyT.h

Return Value

True if the object was found, false otherwise.

Parameters

object
: The object for which to search.

index
: The index of the object.

Class Template Arguments

ReferencedObject
: The type of the referenced (contained) object. This type must be a descendant of OMStorable.

Back to OMStrongReferenceVectorProperty

OMStrongReferenceVectorProperty::getBits

template <class *ReferencedObject***>**
void OMStrongReferenceVectorProperty<*ReferencedObject***>::getBits(OMByte*** *bits***, size_t** *size***) const**

Get the raw bits of this OMStrongReferenceVectorProperty. The raw bits are copied to the buffer at address *bits* which is *size* bytes in size.

Defined in: OMStrongRefVectorPropertyT.h

Parameters

bits
: The address of the buffer into which the raw bits are copied.

size
: The size of the buffer.

Class Template Arguments

ReferencedObject
: The type of the referenced (contained) object. This type must be a descendant of OMStorable.

Back to OMStrongReferenceVectorProperty

OMStrongReferenceVectorProperty::getObjectAt

template <class *ReferencedObject***>**
OMObject* OMStrongReferenceVectorProperty<*ReferencedObject***>::getObjectAt(const size_t** *index***) const**

The value of this OMStrongReferenceVectorProperty at position *index*.

Defined in: OMStrongRefVectorPropertyT.h

Return Value

The object.

Parameters

index
: The index.

Class Template Arguments

ReferencedObject
: The type of the referenced (contained) object. This type must be a descendant of OMStorable.

Back to OMStrongReferenceVectorProperty

OMStrongReferenceVectorProperty::getValueAt

template <class *ReferencedObject***>**
void OMStrongReferenceVectorProperty<*ReferencedObject***>::getValueAt(ReferencedObject*& *object*, const size_t** *index***) const**

Get the value of this OMStrongReferenceVectorProperty at position *index* into *object*.

Defined in: OMStrongRefVectorPropertyT.h

Parameters

object
: A pointer to a *ReferencedObject* by reference.

index
: The position from which to get the *ReferencedObject*.

Class Template Arguments

ReferencedObject
: The type of the referenced (contained) object. This type must be a descendant of OMStorable.

Back to OMStrongReferenceVectorProperty

OMStrongReferenceVectorProperty::grow

void OMStrongReferenceVectorProperty::grow(const size_t *capacity***)**

Increase the capacity of this OMStrongReferenceVectorProperty so that it can contain at least *capacity* *ReferencedObject*s without having to be resized.

Defined in: OMStrongRefVectorPropertyT.h

Parameters

capacity
The desired capacity.

Back to OMStrongReferenceVectorProperty

OMStrongReferenceVectorProperty::indexOfValue

template <class *ReferencedObject***>**
size_t OMStrongReferenceVectorProperty<*ReferencedObject***>::indexOfValue(const ReferencedObject*** *object***) const**

The index of the *ReferencedObject** *object*.

Defined in: OMStrongRefVectorPropertyT.h

Return Value

The index.

Parameters

object
A pointer to the *ReferencedObject* to find.

Class Template Arguments

ReferencedObject
The type of the referenced (contained) object. This type must be a descendant of OMStorable.

Back to OMStrongReferenceVectorProperty

OMStrongReferenceVectorProperty::insert

template <class *ReferencedObject***>**
void OMStrongReferenceVectorProperty<*ReferencedObject***>::insert(const ReferencedObject*** *object***)**

Insert *object* into this OMStrongReferenceVectorProperty. This function is redefined from OMContainerProperty as **appendValue**.

Defined in: OMStrongRefVectorPropertyT.h

Parameters

object

A pointer to a *ReferencedObject*.

Class Template Arguments

ReferencedObject
: The type of the referenced (contained) object. This type must be a descendant of OMStorable.

Back to OMStrongReferenceVectorProperty

OMStrongReferenceVectorProperty::insertAt

template <class *ReferencedObject***>**
void OMStrongReferenceVectorProperty<*ReferencedObject***>::insertAt(const ReferencedObject*** *object***, const size_t** *index***)**

Insert *value* into this OMStrongReferenceVectorProperty at position *index*. Existing values at *index* and higher are shifted up one index position.

Defined in: OMStrongRefVectorPropertyT.h

Parameters

object
: A pointer to a *ReferencedObject*.

index
: The position at which to insert the *ReferencedObject*.

Class Template Arguments

ReferencedObject
: The type of the referenced (contained) object. This type must be a descendant of OMStorable.

Back to OMStrongReferenceVectorProperty

OMStrongReferenceVectorProperty::insertObject

template <class *ReferencedObject***>**
void OMStrongReferenceVectorProperty<*ReferencedObject***>::insertObject(const OMObject*** *object***)**

Insert *object* into this OMStrongReferenceVectorProperty.

Defined in: OMStrongRefVectorPropertyT.h

Parameters

object

The OMObject to insert.

Class Template Arguments

ReferencedObject
The type of the referenced (contained) object. This type must be a descendant of OMStorable.

Back to OMStrongReferenceVectorProperty

OMStrongReferenceVectorProperty::insertObjectAt

template <class *ReferencedObject***>**
void OMStrongReferenceVectorProperty<*ReferencedObject***>::insertObjectAt(const OMObject*** *object***, const size_t** *index***)**

Insert *object* into this OMStrongReferenceVectorProperty at position *index*. Existing objects at *index* and higher are shifted up one index position.

Defined in: OMStrongRefVectorPropertyT.h

Parameters

object
The object to insert.

index
The index at which to insert the object.

Class Template Arguments

ReferencedObject
The type of the referenced (contained) object. This type must be a descendant of OMStorable.

Back to OMStrongReferenceVectorProperty

OMStrongReferenceVectorProperty::isVoid

template <class *ReferencedObject***>**
bool OMStrongReferenceVectorProperty<*ReferencedObject***>::isVoid(void) const**

Is this OMStrongReferenceVectorProperty void ?

Defined in: OMStrongRefVectorPropertyT.h

Return Value

True if this OMStrongReferenceVectorProperty is void, false otherwise.

Class Template Arguments

ReferencedObject
: The type of the referenced (contained) object. This type must be a descendant of OMStorable.

Back to OMStrongReferenceVectorProperty

OMStrongReferenceVectorProperty::prependObject

template <class *ReferencedObject***>**
void OMStrongReferenceVectorProperty<*ReferencedObject***>::prependObject(const OMObject*** *object***)**

Prepend the given *OMObject object* to this OMStrongReferenceVectorProperty.

Defined in: OMStrongRefVectorPropertyT.h

Parameters

object
: The object to prepend.

Class Template Arguments

ReferencedObject
: The type of the referenced (contained) object. This type must be a descendant of OMStorable.

Back to OMStrongReferenceVectorProperty

OMStrongReferenceVectorProperty::prependValue

template <class *ReferencedObject***>**
void OMStrongReferenceVectorProperty<*ReferencedObject***>::prependValue(const ReferencedObject*** *object***)**

Prepend the given *ReferencedObject object* to this OMStrongReferenceVectorProperty.

Defined in: OMStrongRefVectorPropertyT.h

Parameters

object
: A pointer to a *ReferencedObject*.

Class Template Arguments

ReferencedObject

The type of the referenced (contained) object. This type must be a descendant of OMStorable.

Back to OMStrongReferenceVectorProperty

OMStrongReferenceVectorProperty::removeAllObjects

template <class *ReferencedObject***>**
void OMStrongReferenceVectorProperty<*ReferencedObject***>::removeAllObjects(void)**

Remove all objects from this OMStrongReferenceVectorProperty.

Defined in: OMStrongRefVectorPropertyT.h

Class Template Arguments

ReferencedObject
The type of the referenced (contained) object. This type must be a descendant of OMStorable.

Back to OMStrongReferenceVectorProperty

OMStrongReferenceVectorProperty::removeAt

template <class *ReferencedObject***>**
ReferencedObject* OMStrongReferenceVectorProperty<*ReferencedObject***>::removeAt(const size_t** *index***)**

Remove the object from this OMStrongReferenceVectorProperty at position *index*. Existing objects in this OMStrongReferenceVectorProperty at *index* + 1 and higher are shifted down one index position.

Defined in: OMStrongRefVectorPropertyT.h

Return Value

A pointer to the removed *ReferencedObject*. If lazy loading is enabled and the referenced object was never loaded the value returned is 0.

Parameters

index
The position from which to remove the *ReferencedObject*.

Class Template Arguments

ReferencedObject
The type of the referenced (contained) object. This type must be a descendant of OMStorable.

Back to OMStrongReferenceVectorProperty

OMStrongReferenceVectorProperty::removeFirst

template <class *ReferencedObject***>**
ReferencedObject* OMStrongReferenceVectorProperty<*ReferencedObject***>::removeFirst(void)**

Remove the first (index == 0) object from this OMStrongReferenceVectorProperty. Existing objects in this OMStrongReferenceVectorProperty are shifted down one index position.

Defined in: OMStrongRefVectorPropertyT.h

Return Value

A pointer to the removed *ReferencedObject*. If lazy loading is enabled and the referenced object was never loaded the value returned is 0.

Class Template Arguments

ReferencedObject
: The type of the referenced (contained) object. This type must be a descendant of OMStorable.

Back to OMStrongReferenceVectorProperty

OMStrongReferenceVectorProperty::removeLast

template <class *ReferencedObject***>**
ReferencedObject* OMStrongReferenceVectorProperty<*ReferencedObject***>::removeLast(void)**

Remove the last (index == count() - 1) object from this OMStrongReferenceVectorProperty.

Defined in: OMStrongRefVectorPropertyT.h

Return Value

A pointer to the removed *ReferencedObject*. If lazy loading is enabled and the referenced object was never loaded the value returned is 0.

Class Template Arguments

ReferencedObject
: The type of the referenced (contained) object. This type must be a descendant of OMStorable.

Back to OMStrongReferenceVectorProperty

OMStrongReferenceVectorProperty::removeObject

template <class *ReferencedObject***>**
void OMStrongReferenceVectorProperty<*ReferencedObject***>::removeObject(const OMObject*** *object***)**

Remove *object* from this OMStrongReferenceVectorProperty.

Defined in: OMStrongRefVectorPropertyT.h

Parameters

object
: The OMObject to remove.

Class Template Arguments

ReferencedObject
: The type of the referenced (contained) object. This type must be a descendant of OMStorable.

Back to OMStrongReferenceVectorProperty

OMStrongReferenceVectorProperty::removeObjectAt

template <class *ReferencedObject***>**
OMObject* OMStrongReferenceVectorProperty<*ReferencedObject***>::removeObjectAt(const size_t** *index***)**

Remove the object from this OMStrongReferenceVectorProperty at position *index*. Existing objects in this OMStrongReferenceVectorProperty at *index* + 1 and higher are shifted down one index position.

Defined in: OMStrongRefVectorPropertyT.h

Parameters

index
: The index of the object to remove.

Class Template Arguments

ReferencedObject
: The type of the referenced (contained) object. This type must be a descendant of OMStorable.

Back to OMStrongReferenceVectorProperty

OMStrongReferenceVectorProperty::removeProperty

template <class *ReferencedObject***>**
void OMStrongReferenceVectorProperty<*ReferencedObject***>::removeProperty(void)**

Remove this optional OMStrongReferenceVectorProperty.

Defined in: OMStrongRefVectorPropertyT.h

Class Template Arguments

ReferencedObject
: The type of the referenced (contained) object. This type must be a descendant of OMStorable.

Back to OMStrongReferenceVectorProperty

OMStrongReferenceVectorProperty::removeValue

template <class *ReferencedObject***>**
void OMStrongReferenceVectorProperty<*ReferencedObject***>::removeValue(const ReferencedObject*** *object***)**

Remove *object* from this OMStrongReferenceVectorProperty.

Defined in: OMStrongRefVectorPropertyT.h

Parameters

object
: A pointer to a *ReferencedObject*.

Class Template Arguments

ReferencedObject
: The type of the referenced (contained) object. This type must be a descendant of OMStorable.

Back to OMStrongReferenceVectorProperty

OMStrongReferenceVectorProperty::restore

template <class *ReferencedObject***>**
void OMStrongReferenceVectorProperty<*ReferencedObject***>::restore(size_t** *externalSize***)**

Restore this OMStrongReferenceVectorProperty, the external (persisted) size of the OMStrongReferenceVectorProperty is *externalSize*.

Defined in: OMStrongRefVectorPropertyT.h

Parameters

externalSize
: The external (persisted) size of the OMStrongReferenceVectorProperty.

Class Template Arguments

ReferencedObject
: The type of the referenced (contained) object. This type must be a descendant of OMStorable.

Back to OMStrongReferenceVectorProperty

OMStrongReferenceVectorProperty::save

template <class *ReferencedObject***>**
void OMStrongReferenceVectorProperty<*ReferencedObject***>::save(void) const**

Save this OMStrongReferenceVectorProperty.

Defined in: OMStrongRefVectorPropertyT.h

Class Template Arguments

ReferencedObject
: The type of the referenced (contained) object. This type must be a descendant of OMStorable.

Back to OMStrongReferenceVectorProperty

OMStrongReferenceVectorProperty::setBits

template <class *ReferencedObject***>**
void OMStrongReferenceVectorProperty<*ReferencedObject***>::setBits(const OMByte*** *bits***, size_t** *size***)**

Set the raw bits of this OMStrongReferenceVectorProperty. The raw bits are copied from the buffer at address *bits* which is *size* bytes in size.

Defined in: OMStrongRefVectorPropertyT.h

Parameters

bits
: The address of the buffer from which the raw bits are copied.

size
: The size of the buffer.

Class Template Arguments

ReferencedObject
: The type of the referenced (contained) object. This type must be a descendant of OMStorable.

Back to OMStrongReferenceVectorProperty

OMStrongReferenceVectorProperty::setObjectAt

template <class *ReferencedObject***>**
OMObject* OMStrongReferenceVectorProperty<*ReferencedObject***>::setObjectAt(const OMObject*** *object***, const size_t** *index***)**

Set the value of this OMStrongReferenceVectorProperty at position *index* to *object*.

Defined in: OMStrongRefVectorPropertyT.h

Return Value

The old object.

Parameters

object
: The new object.

index
: The index.

Class Template Arguments

ReferencedObject
: The type of the referenced (contained) object. This type must be a descendant of OMStorable.

Back to OMStrongReferenceVectorProperty

OMStrongReferenceVectorProperty::setValueAt

template <class *ReferencedObject***>**
ReferencedObject* OMStrongReferenceVectorProperty<*ReferencedObject***>::setValueAt(const ReferencedObject*** *object***, const size_t** *index***)**

Set the value of this OMStrongReferenceVectorProperty at position *index* to *object*.

Defined in: OMStrongRefVectorPropertyT.h

Return Value

A pointer to the old *ReferencedObject*. If lazy loading is enabled and the referenced object was never loaded the value returned is 0.

Parameters

object
: A pointer to the new *ReferencedObject*.

index
: The position at which to insert the *ReferencedObject*.

Class Template Arguments

ReferencedObject
: The type of the referenced (contained) object. This type must be a descendant of OMStorable.

Back to OMStrongReferenceVectorProperty

OMStrongReferenceVectorProperty::valueAt

template <class *ReferencedObject***>**
ReferencedObject* OMStrongReferenceVectorProperty<_ReferencedObject_**>::valueAt(const size_t** *index***) const**

The value of this OMStrongReferenceVectorProperty at position *index*.

Defined in: OMStrongRefVectorPropertyT.h

Return Value

A pointer to the *ReferencedObject*.

Parameters

index
: The position from which to get the *ReferencedObject*.

Class Template Arguments

ReferencedObject
: The type of the referenced (contained) object. This type must be a descendant of OMStorable.

Back to OMStrongReferenceVectorProperty

OMType class

OMType **class OMType**

Abstract base class describing the data types that may be assumed by persistent properties supported by the Object Manager.

Defined in: OMType.h

Author

Tim Bingham - tjb - (Avid Technology, Inc.)

Class Members

Public members.

virtual void reorder(OMByte* externalBytes, size_t externalBytesSize) const

Reorder (swap) the given *externalBytes* according to the data type described by this **OMType**. The number of bytes is given by *externalBytesSize*. Object Manager clients must provide a suitable implementation of this virtual function for the data type being described. This class provides static functions to aid in such an implementation. The bytes to be swapped are assumed to be in external form.

virtual size_t externalSize(OMByte* internalBytes, size_t internalBytesSize) const

The size, in bytes, of an entity described by *internalBytes*, *internalBytesSize* and this **OMType** when persisted. Object Manager clients must provide a suitable implementation of this virtual function for the data type being described. This class provides static functions to aid in such an implementation.

virtual void externalize(OMByte* internalBytes, size_t internalBytesSize, OMByte* externalBytes, size_t externalBytesSize, OMByteOrder byteOrder) const

Convert the given *internalBytes* from internal (in memory) representation to external (persisted) representation, according to the data type described by this **OMType**. The number of bytes is given by *internalBytesSize*. Together *externalBytes* and *externalBytesSize* define a buffer that must be of appropriate size to receive the converted value. Note that the resulting external value may be smaller than, larger than, or the same size as the original internal value depending on the relationship between **internalSize** and **externalSize**. The value is in the *byteOrder* specified. Object Manager clients must provide a suitable implementation of this virtual function for the data type being described. This class provides static functions to aid in such an implementation.

virtual size_t internalSize(OMByte* externalBytes, size_t externalSize) const

The size, in bytes, of an entity described by *externalBytes*, *externalBytesSize* and this **OMType** when in memory. Object Manager clients must provide a suitable implementation of this virtual function for the data type being described. This class provides static functions to aid in such an implementation.

virtual void internalize(OMByte* externalBytes, size_t externalBytesSize, OMByte* internalBytes, size_t internalBytesSize, OMByteOrder byteOrder) const

Convert the given *externalBytes* from external (persisted) representation to internal (in memory) representation, according to the data type described by this **OMType**. The number of bytes is given by *externalBytesSize*. Together *internalBytes* and *internalBytesSize* define a buffer that must be of appropriate size to receive the converted value. Note that the resulting internal value may be smaller than, larger than, or the same size as the original external value depending on the relationship between **externalSize** and **internalSize**. The value is in the *byteOrder* specified. Object Manager clients must provide a suitable implementation of this virtual function for the data type being described. This class provides static functions to aid in such an implementation.

Class Members

Static members.

static void reorderInteger**(OMByte* bytes, size_t bytesSize)**

Reorder (swap) the integer described by *bytes* and *bytesSize*. This static function is provided to aid Object Manager clients in providing suitable implementations of the virtual functions in this class. In particular, Object Manager clients may wish to use this function when implementing **reorder**.

static void expand**(OMByte* inputBytes, size_t inputBytesSize, OMByte* outputBytes, size_t outputBytesSize, OMByteOrder byteOrder)**

Expand, by padding with leading zeros, the value decribed by *inputBytes* and *inputBytesSize* into the buffer described by *outputBytes* and *outputBytesSize*. The value is in the *byteOrder* specified. This static function is provided to aid Object Manager clients in providing suitable implementations of the virtual functions in this class. In particular, Object Manager clients may wish to use this function when implementing **internalize** and **externalize**.

static void contract**(OMByte* inputBytes, size_t inputBytesSize, OMByte* outputBytes, size_t outputBytesSize, OMByteOrder byteOrder)**

Contract, by discarding leading bytes, the value decribed by *inputBytes* and *inputBytesSize* into the buffer described by *outputBytes* and *outputBytesSize*. The value is in the *byteOrder* specified. This static function is provided to aid Object Manager clients in providing suitable implementations of the virtual functions in this class. In particular, Object Manager clients may wish to use this function when implementing **internalize** and **externalize**.

static void copy**(OMByte* inputBytes, OMByte* outputBytes, size_t bytesSize)**

Copy the value decribed by *inputBytes* and *bytesSize* into the buffer described by *outputBytes* and *bytesSize*. This static function is provided to aid Object Manager clients in providing suitable implementations of the virtual functions in this class. In particular, Object Manager clients may wish to use this function when implementing **internalize** and **externalize**.

OMType::contract

void OMType::contract(OMByte* *inputBytes*, **size_t** *inputBytesSize*, **OMByte*** *outputBytes*, **size_t** *outputBytesSize*, **OMByteOrder** *byteOrder***)**

Contract, by discarding leading bytes, the value decribed by *inputBytes* and *inputBytesSize* into the buffer described by *outputBytes* and *outputBytesSize*. The value is in the *byteOrder* specified. This static function is provided to aid Object Manager clients in providing suitable implementations of the virtual functions in this class. In particular, Object Manager clients may wish to use this function when implementing **internalize** and **externalize**.

Defined in: OMType.cpp

Parameters

inputBytes
: The buffer containing the input bytes.

inputBytesSize
: The size of the input buffer.

outputBytes
: The buffer to receive the output bytes.

outputBytesSize

The size of the output buffer.

byteOrder

The byte order.

Back to OMType

OMType::copy

void OMType::copy(OMByte* *inputBytes*, **OMByte*** *outputBytes*, **size_t** *bytesSize***)**

Copy the value decribed by *inputBytes* and *bytesSize* into the buffer described by *outputBytes* and *bytesSize*. This static function is provided to aid Object Manager clients in providing suitable implementations of the virtual functions in this class. In particular, Object Manager clients may wish to use this function when implementing **internalize** and **externalize**.

Defined in: OMType.cpp

Parameters

inputBytes

The buffer containing the input bytes.

outputBytes

The buffer to receive the output bytes.

bytesSize

The size of the input and output buffers.

Back to OMType

OMType::expand

void OMType::expand(OMByte* *inputBytes*, **size_t** *inputBytesSize*, **OMByte*** *outputBytes*, **size_t** *outputBytesSize*, **OMByteOrder** *byteOrder***)**

Expand, by padding with leading zeros, the value decribed by *inputBytes* and *inputBytesSize* into the buffer described by *outputBytes* and *outputBytesSize*. The value is in the *byteOrder* specified. This static function is provided to aid Object Manager clients in providing suitable implementations of the virtual functions in this class. In particular, Object Manager clients may wish to use this function when implementing **internalize** and **externalize**.

Defined in: OMType.cpp

Parameters

inputBytes

The buffer containing the input bytes.

inputBytesSize

The size of the input buffer.

outputBytes

The buffer to receive the output bytes.

outputBytesSize

The size of the output buffer.

byteOrder
The byte order.
Back to OMType

OMType::reorderInteger

void OMType::reorderInteger(OMByte* *bytes*, **size_t** *bytesSize***)**

Reorder (swap) the integer described by *bytes* and *bytesSize*. This static function is provided to aid Object Manager clients in providing suitable implementations of the virtual functions in this class. In particular, Object Manager clients may wish to use this function when implementing **reorder**.

Defined in: OMType.cpp

Parameters

bytes
The buffer containing the bytes to reorder.
bytesSize
The size of the buffer.
Back to OMType

OMUniqueObjectIdentificationType class

OMUniqueObjectIdentificationType **class OMUniqueObjectIdentificationType: public OMType, public OMSingleton**

Type definition for OMUniqueObjectIdentification.

Defined in: OMUniqueObjectIdentType.h

Author

Tim Bingham - tjb - (Avid Technology, Inc.)

OMVariableSizeProperty class

OMVariableSizeProperty **class OMVariableSizeProperty: public OMSimpleProperty**

Variable size simple (data) persistent properties supported by the Object Manager.

Defined in: OMVariableSizeProperty.h

Class Template Arguments

PropertyType
The type of the property. This can be any type.

Author

Tim Bingham - tjb - (Avid Technology, Inc.)

Class Members

Public members.

OMVariableSizeProperty(const OMPropertyId propertyId, const wchar_t* name)

Constructor.

virtual ~OMVariableSizeProperty(void)

Destructor.

void getValue**(PropertyType* value, size_t valueSize) const**

Get the value of this **OMVariableSizeProperty.**

void setValue**(const PropertyType* value, size_t valueSize)**

Set the value of this **OMVariableSizeProperty.** The value is set by copying *valueSize* bytes from the address *value* into the **OMVariableSizeProperty.**

void setElementValues**(const PropertyType* value, size_t elementCount)**

Set the value of this **OMVariableSizeProperty.** The value is set by copying *elementCount* elements from the address *value* into the **OMVariableSizeProperty.**

void getValueAt**(PropertyType* value, const size_t index) const**

Get the value of the item at position *index* in this **OMVariableSizeProperty.** The value is obtained by copying a single item of type PropertyType from this **OMVariableSizeProperty** at position *index*.

void setValueAt**(const PropertyType* value, const size_t index)**

Set the value of the item at position *index* in this **OMVariableSizeProperty.** The value is set by copying a single item of type PropertyType into this **OMVariableSizeProperty** at position *index*.

void appendValue**(const PropertyType* value)**

Set the value of the item at the last position in this **OMVariableSizeProperty.** The **OMVariableSizeProperty** is first increased in size by one item. The value is then set by copying a single item of type PropertyType into this **OMVariableSizeProperty** at the last position.

void prependValue**(const PropertyType* value)**

Set the value of the item at the first position in this **OMVariableSizeProperty.** The **OMVariableSizeProperty** is first increased in size by one item and all existing items are moved up by on position. The value is then set by copying a single item of type PropertyType into this **OMVariableSizeProperty** at the first position.

bool copyToBuffer**(PropertyType* buffer, size_t bufferSize) const**

Get the value of this **OMVariableSizeProperty.** The value is obtained by copying the value from the **OMVariableSizeProperty.** The buffer is at address *buffer* and is *bufferSize* bytes in size. Copying only takes place if the buffer is large enough.

bool copyElementsToBuffer**(PropertyType* buffer, size_t elementCount) const**

Get the value of this **OMVariableSizeProperty.** The value is obtained by copying the value from the **OMVariableSizeProperty.** The buffer is at address *buffer* and is *elementCount* elements in size. Copying only takes place if the buffer is large enough.

virtual void restore**(size_t externalSize)**

Restore this **OMVariableSizeProperty,** the external (persisted) size of the **OMVariableSizeProperty** is *externalSize*.

size_t count**(void) const**

The number of items in this this **OMVariableSizeProperty.**

OMVariableSizeProperty::appendValue

template <class *PropertyType***>**
void OMVariableSizeProperty<*PropertyType***>::appendValue(const PropertyType*** *value***)**

Set the value of the item at the last position in this OMVariableSizeProperty. The OMVariableSizeProperty is first increased in size by one item. The value is then set by copying a single item of type PropertyType into this OMVariableSizeProperty at the last position.

Defined in: OMVariableSizePropertyT.h

Parameters

value
A pointer to the new value that is to be copied into this OMVariableSizeProperty.

Class Template Arguments

PropertyType
The type of the property. This can be any type.

Back to OMVariableSizeProperty

OMVariableSizeProperty::copyElementsToBuffer

template <class *PropertyType***>**
bool OMVariableSizeProperty<*PropertyType***>::copyElementsToBuffer(PropertyType*** *buffer***, size_t** *elementCount***) const**

Get the value of this OMVariableSizeProperty. The value is obtained by copying the value from the OMVariableSizeProperty. The buffer is at address *buffer* and is *elementCount* elements in size. Copying only takes place if the buffer is large enough.

Defined in: OMVariableSizePropertyT.h

Return Value

true if the value was successfully copied **false** otherwise.

Parameters

buffer
A pointer to the buffer.
elementCount
The number of elements in the buffer.

Class Template Arguments

PropertyType
The type of the property. This can be any type.
Back to OMVariableSizeProperty

OMVariableSizeProperty::copyToBuffer

template <class *PropertyType***>**
bool OMVariableSizeProperty<*PropertyType***>::copyToBuffer(PropertyType*** *buffer***, size_t** *bufferSize***) const**

Get the value of this OMVariableSizeProperty. The value is obtained by copying the value from the OMVariableSizeProperty. The buffer is at address *buffer* and is *bufferSize* bytes in size. Copying only takes place if the buffer is large enough.

Defined in: OMVariableSizePropertyT.h

Return Value

true if the value was successfully copied **false** otherwise.

Parameters

buffer
A pointer to the buffer.
bufferSize
The size of the buffer.

Class Template Arguments

PropertyType
The type of the property. This can be any type.
Back to OMVariableSizeProperty

OMVariableSizeProperty::count

template <class *PropertyType***>**
size_t OMVariableSizeProperty<*PropertyType***>::count(void) const**

The number of items in this this OMVariableSizeProperty.

Defined in: OMVariableSizePropertyT.h

Return Value

The number of items in this OMVariableSizeProperty.

Class Template Arguments

PropertyType
The type of the property. This can be any type.
Back to OMVariableSizeProperty

OMVariableSizeProperty::getValue

template <class *PropertyType***>**
void OMVariableSizeProperty<*PropertyType***>::getValue(PropertyType*** *value***, size_t** *valueSize***) const**

Get the value of this OMVariableSizeProperty.

Defined in: OMVariableSizePropertyT.h

Parameters

value
A pointer to a *PropertyType*
valueSize
The size of the *value*

Class Template Arguments

PropertyType
The type of the property. This can be any type.
Back to OMVariableSizeProperty

OMVariableSizeProperty::getValueAt

template <class *PropertyType***>**
void OMVariableSizeProperty<*PropertyType***>::getValueAt(PropertyType*** *value***, const size_t** *index***) const**

Get the value of the item at position *index* in this OMVariableSizeProperty. The value is obtained by copying a single item of type PropertyType from this OMVariableSizeProperty at position *index*.

Defined in: OMVariableSizePropertyT.h

Parameters

value
The location that is to receive the value copied out of this OMVariableSizeProperty.
index
The index of the value to get.

Class Template Arguments

PropertyType

The type of the property. This can be any type.

Back to OMVariableSizeProperty

OMVariableSizeProperty::prependValue

void OMVariableSizeProperty::prependValue(const PropertyType* *value***)**

Set the value of the item at the first position in this OMVariableSizeProperty. The OMVariableSizeProperty is first increased in size by one item and all existing items are moved up by on position. The value is then set by copying a single item of type PropertyType into this OMVariableSizeProperty at the first position.

Defined in: OMVariableSizePropertyT.h

Parameters

value

A pointer to the new value that is to be copied into this OMVariableSizeProperty.

Back to OMVariableSizeProperty

OMVariableSizeProperty::restore

template <class *PropertyType***>**
void OMVariableSizeProperty<*PropertyType***>::restore(size_t** *externalSize***)**

Restore this OMVariableSizeProperty, the external (persisted) size of the OMVariableSizeProperty is *externalSize*.

Defined in: OMVariableSizePropertyT.h

Parameters

externalSize

The external (persisted) size of the OMVariableSizeProperty.

Class Template Arguments

PropertyType

The type of the property. This can be any type.

Back to OMVariableSizeProperty

OMVariableSizeProperty::setElementValues

template <class *PropertyType***>**
void OMVariableSizeProperty<*PropertyType***>::setElementValues(const PropertyType*** *value***, size_t** *elementCount***)**

Set the value of this OMVariableSizeProperty. The value is set by copying *elementCount* elements from the address *value* into the OMVariableSizeProperty.

Defined in: OMVariableSizePropertyT.h

Parameters

value
: A pointer to an array of *PropertyType*s

elementCount
: The number of element in the array *value*

Class Template Arguments

PropertyType
: The type of the property. This can be any type.

Back to OMVariableSizeProperty

OMVariableSizeProperty::setValue

template <class *PropertyType***>**
void OMVariableSizeProperty<*PropertyType***>::setValue(const PropertyType*** *value***, size_t** *valueSize***)**

Set the value of this OMVariableSizeProperty. The value is set by copying *valueSize* bytes from the address *value* into the OMVariableSizeProperty.

Defined in: OMVariableSizePropertyT.h

Parameters

value
: A pointer to an array of *PropertyType*s

valueSize
: The size of the array *value* in bytes

Class Template Arguments

PropertyType
: The type of the property. This can be any type.

Back to OMVariableSizeProperty

OMVariableSizeProperty::setValueAt

template <class *PropertyType***>**
void OMVariableSizeProperty<*PropertyType***>::setValueAt(const PropertyType*** *value***, const size_t** *index***)**

Set the value of the item at position *index* in this OMVariableSizeProperty. The value is set by copying a single item of type PropertyType into this OMVariableSizeProperty at position *index*.

Defined in: OMVariableSizePropertyT.h

Parameters

value
: A pointer to the new value that is to be copied into this OMVariableSizeProperty.

index
: The index of the value to set.

Class Template Arguments

PropertyType
: The type of the property. This can be any type.

Back to OMVariableSizeProperty

OMVector class

OMVector **class OMVector: public OMContainer**

Elastic sequential collections of elements accessible by index. The order of elements is determined externally. Duplicate elements are allowed.

Defined in: OMVector.h

Class Template Arguments

Element
: The type of an **OMVector** element. This type must support operator = and operator ==.

Author

Tim Bingham - tjb - (Avid Technology, Inc.)

Class Members

Public members.

OMVector**()**
: Constructor.

virtual ~OMVector**(void)**
: Destructor.

virtual void insert**(const Element value)**
: Insert *value* into this **OMVector.**

virtual bool containsValue**(const Element value) const**
: Does this **OMVector** contain *value* ?

size_t count**(void) const**

The number of elements in this **OMVector**. **count** returns the actual number of elements in the **OMVector**.

virtual void removeValue**(const Element value)**

Remove *value* from this **OMVector**. In the case of duplicate values, the one with the lowest index is removed.

virtual size_t capacity**(void) const**

The capacity of this **OMVector**. **capacity** returns the potential number of elements in the **OMVector**.

virtual void grow**(size_t capacity)**

Increase the capacity of this **OMVector** so that it can contain at least *capacity* elements without having to be resized.

virtual void shrink**(size_t capacity)**

Free any unused capacity in this **OMVector** while ensuring that it can contain at least *capacity* elements.

virtual bool full**(void) const**

Is this **OMVector** full ?

virtual bool empty**(void) const**

Is this **OMVector** empty ?

void setAt**(const Element value, const size_t index)**

Set the value of the *Element* at position *index* in this **OMVector**. The existing *Element* at *index* is replaced.

void getAt**(Element& value, const size_t index) const**

Get the value of the *Element* at position *index* in this **OMVector**.

Element& getAt**(const size_t index) const**

Get the value of the *Element* at position *index* in this **OMVector**.

Element& valueAt**(const size_t index) const**

The value of the *Element* at position *index* in this **OMVector**.

virtual void insertAt**(const Element value, const size_t index)**

Insert *value* into this **OMVector** at position *index*. Existing values in this **OMVector** at *index* and higher are shifted up one index position.

void append**(const Element value)**

Append the given *Element value* to this **OMVector**. The new element is added after the last element currently in this **OMVector**.

void prepend**(const Element value)**

Prepend the given *Element value* to this **OMVector**. The new element is added before the first element currently in this **OMVector**. Existing values in this **OMVector** are shifted up one index position.

virtual void removeAt**(const size_t index)**

Remove the value from this **OMVector** at position *index*. Existing values in this **OMVector** at *index* + 1 and higher are shifted down on index position.

void removeLast**(void)**

Remove the last (index == count() - 1) element from this **OMVector**.

void removeFirst**(void)**

Remove the first (index == 0) element from this **OMVector**. Existing values in this **OMVector** are shifted down one index position.

virtual void clear**(void)**

Remove all elements from this **OMVector**.

size_t indexOfValue**(const Element value) const**

The index of the element with value *value*. In the case of duplicate values, lowest index is returned.

size_t countValue**(const Element value) const**

The number of elements with value *value*.

Class Members

Private members.
static size_t nextHigherCapacity**(size_t capacity)**
Calculate the next valid capacity higher than *capacity*.

OMVector::append

template <class *Element***>**
void OMVector<*Element***>::append(const Element** *value***)**

Append the given *Element value* to this OMVector. The new element is added after the last element currently in this OMVector.

Defined in: OMVectorT.h

Parameters

value
The value to append.

Class Template Arguments

Element
The type of an OMVector element. This type must support operator = and operator ==.
Back to OMVector

OMVector::capacity

template <class *Element***>**
size_t OMVector<*Element***>::capacity(void) const**

The capacity of this OMVector. **capacity** returns the potential number of elements in the OMVector.

Defined in: OMVectorT.h

Return Value

The capacity of this OMVector.

Class Template Arguments

Element
The type of an OMVector element. This type must support operator = and operator ==.
Back to OMVector

OMVector::clear

template <class *Element***>**
void OMVector<*Element***>::clear(void)**

Remove all elements from this OMVector. from this OMVector. Existing values in this OMVector are shifted down one index position.

Defined in: OMVectorT.h

Class Template Arguments

Element
The type of an OMVector element. This type must support operator = and operator ==.

Back to OMVector

OMVector::containsValue

template <class *Element***>**
bool OMVector<*Element***>::containsValue(const Element** *value***) const**

Does this OMVector contain *value* ?

Defined in: OMVectorT.h

Parameters

value
The Element to search for. A value of type *Element* by value.

Class Template Arguments

Element
The type of an OMVector element. This type must support operator = and operator ==.

Back to OMVector

OMVector::count

template <class *Element***>**
size_t OMVector<*Element***>::count(void) const**

The number of elements in this OMVector. **count** returns the actual number of elements in the OMVector.

Defined in: OMVectorT.h

Return Value

The number of elements in this OMVector.

Class Template Arguments

Element
The type of an OMVector element. This type must support operator = and operator ==.

Back to OMVector

OMVector::countValue

template <class *Element***>**
size_t OMVector<*Element***>::countValue(const Element** *value***) const**

The number of elements with value *value*.

Defined in: OMVectorT.h

Parameters

value
The value for which the index is to be found.

Class Template Arguments

Element
The type of an OMVector element. This type must support operator = and operator ==.

Back to OMVector

OMVector::empty

template <class *Element***>**
bool OMVector<*Element***>::empty(void) const**

Is this OMVector empty ?

Defined in: OMVectorT.h

Return Value

True if this OMVector is empty, false otherwise ?

Class Template Arguments

Element

The type of an OMVector element. This type must support operator = and operator ==.
Back to OMVector

OMVector::full

template <class *Element***>**
bool OMVector<*Element***>::full(void) const**

Is this OMVector full ?

Defined in: OMVectorT.h

Return Value

True if this OMVector is full, false otherwise ?

Class Template Arguments

Element
The type of an OMVector element. This type must support operator = and operator ==.
Back to OMVector

OMVector::getAt

template <class *Element***>**
Element& OMVector<*Element***>::getAt(const size_t** *index***) const**

Get the value of the *Element* at position *index* in this OMVector.

Defined in: OMVectorT.h

Parameters

index
The index.

Class Template Arguments

Element
The type of an OMVector element. This type must support operator = and operator ==.
Back to OMVector

OMVector::getAt

template <class *Element***>**
void OMVector<*Element***>::getAt(Element&** *value***, const size_t** *index***) const**

Get the value of the *Element* at position *index* in this OMVector.

Defined in: OMVectorT.h

Parameters

value
: The value obtained. By reference.

index
: The index.

Class Template Arguments

Element
: The type of an OMVector element. This type must support operator = and operator ==.

Back to OMVector

OMVector::grow

template <class *Element***>**
void OMVector<*Element***>::grow(size_t** *capacity***)**

Increase the capacity of this OMVector so that it can contain at least *capacity* elements without having to be resized.

Defined in: OMVectorT.h

Parameters

capacity
: The new capacity.

Class Template Arguments

Element
: The type of an OMVector element. This type must support operator = and operator ==.

Back to OMVector

OMVector::indexOfValue

template <class *Element***>**
size_t OMVector<*Element***>::indexOfValue(const Element** *value***) const**

The index of the element with value *value*. In the case of duplicate values, lowest index is returned.

Defined in: OMVectorT.h

Parameters

value
The value for which the index is to be found.

Class Template Arguments

Element
The type of an OMVector element. This type must support operator = and operator ==.

Back to OMVector

OMVector::insert

template <class *Element***>**
void OMVector<*Element***>::insert(const Element** *value***)**

Insert *value* into this OMVector.

Defined in: OMVectorT.h

Parameters

value
The Element to insert. A value of type *Element* by value.

Class Template Arguments

Element
The type of an OMVector element. This type must support operator = and operator ==.

Back to OMVector

OMVector::insertAt

template <class *Element***>**
void OMVector<*Element***>::insertAt(const Element** *value***, const size_t** *index***)**

Insert *value* into this OMVector at position *index*. Existing values in this OMVector at *index* and higher are shifted up one index position.

Defined in: OMVectorT.h

Parameters

value
The value to insert.

index
The index.

Class Template Arguments

Element
The type of an OMVector element. This type must support operator = and operator ==.
Back to OMVector

OMVector::nextHigherCapacity

template <class *Element***>**
size_t OMVector<*Element***>::nextHigherCapacity(size_t** *capacity***)**

Calculate the next valid capacity higher than *capacity*.

Defined in: OMVectorT.h

Parameters

capacity
The desired capaciy.

Class Template Arguments

Element
The type of an OMVector element. This type must support operator = and operator ==.
Back to OMVector

OMVector::OMVector

template <class *Element***>**
OMVector<*Element***>::OMVector(void)**

Constructor.

Defined in: OMVectorT.h

Class Template Arguments

Element
The type of an OMVector element. This type must support operator = and operator ==.
Back to OMVector

OMVector::prepend

template <class *Element***>**
void OMVector<*Element***>::prepend(const Element** *value***)**

Prepend the given *Element value* to this OMVector. The new element is added before the first element currently in this OMVector. Existing values in this OMVector are shifted up one index position.

Defined in: OMVectorT.h

Parameters

value
The value to prepend.

Class Template Arguments

Element
The type of an OMVector element. This type must support operator = and operator ==.

Back to OMVector

OMVector::removeAt

template <class *Element***>**
void OMVector<*Element***>::removeAt(const size_t** *index***)**

Remove the value from this OMVector at position *index*. Existing values in this OMVector at *index* + 1 and higher are shifted down on index position.

Defined in: OMVectorT.h

Parameters

index
The index of the value to be removed.

Class Template Arguments

Element
The type of an OMVector element. This type must support operator = and operator ==.

Back to OMVector

OMVector::removeFirst

template <class *Element***>**
void OMVector<*Element***>::removeFirst(void)**

Remove the first (index == 0) element from this OMVector. Existing values in this OMVector are shifted down one index position.

Defined in: OMVectorT.h

Class Template Arguments

Element
The type of an OMVector element. This type must support operator = and operator ==.

Back to OMVector

OMVector::removeLast

template <class *Element***>**
void OMVector<*Element***>::removeLast(void)**

Remove the last (index == count() - 1) element from this OMVector.

Defined in: OMVectorT.h

Class Template Arguments

Element
The type of an OMVector element. This type must support operator = and operator ==.

Back to OMVector

OMVector::removeValue

template <class *Element***>**
void OMVector<*Element***>::removeValue(const Element** *value***)**

Remove *value* from this OMVector. In the case of duplicate values, the one with the lowest index is removed.

Defined in: OMVectorT.h

Parameters

value
The Element to remove. A value of type *Element* by value.

Class Template Arguments

Element
The type of an OMVector element. This type must support operator = and operator ==.

Back to OMVector

OMVector::setAt

template <class *Element***>**
void OMVector<*Element***>::setAt(const Element** *value***, const size_t** *index***)**

Set the value of the *Element* at position *index* in this OMVector. The existing *Element* at *index* is replaced.

Defined in: OMVectorT.h

Parameters

value
The new value.
index
The index.

Class Template Arguments

Element
The type of an OMVector element. This type must support operator = and operator ==.
Back to OMVector

OMVector::shrink

template <class *Element***>**
void OMVector<*Element***>::shrink(size_t** *capacity***)**

Free any unused capacity in this OMVector while ensuring that it can contain at least *capacity* elements.

Defined in: OMVectorT.h

Parameters

capacity
The new capacity.

Class Template Arguments

Element
The type of an OMVector element. This type must support operator = and operator ==.
Back to OMVector

OMVector::valueAt

template <class *Element***>**
Element& OMVector<*Element***>::valueAt(const size_t** *index***) const**

The value of the *Element* at position *index* in this OMVector.

Defined in: OMVectorT.h

Parameters

index
The index.

Class Template Arguments

Element
The type of an OMVector element. This type must support operator = and operator ==.

Back to OMVector

OMVector::~OMVector

template <class *Element***>**
OMVector<*Element***>::~OMVector(void)**

Destructor.

Defined in: OMVectorT.h

Class Template Arguments

Element
The type of an OMVector element. This type must support operator = and operator ==.

Back to OMVector

OMVectorElement class

OMVectorElement **class OMVectorElement**

Pointer elements of non-persistent Object Manager vectors.

Defined in: OMContainerElement.h

Class Template Arguments

ReferencedObject
The type of the referenced object.

Author

Tim Bingham - tjb - (Avid Technology, Inc.)

Class Members

Public members.

OMVectorElement**(void)**

Constructor.

OMVectorElement**(const ReferencedObject* pointer)**

Constructor.

OMVectorElement**(const OMVectorElement&ltReferencedObject>& rhs)**

Copy constructor.

~OMVectorElement**(void)**

Destructor.

OMVectorElement&ltReferencedObject>& operator=**(const OMVectorElement&ltReferencedObject>& rhs)**

Assignment. This operator provides value semantics for OMContainer.

bool operator==**(const OMVectorElement&ltReferencedObject>& rhs) const**

Equality. This operator provides value semantics for OMContainer.

ReferencedObject* getValue**(void) const**

Get the value of this **OMVectorElement.**

ReferencedObject* setValue**(const ReferencedObject* value)**

Set the value of this **OMVectorElement.**

ReferencedObject* pointer**(void) const**

The value of this **OMVectorElement** as a pointer.

OMVectorElement::getValue

template <class *ReferencedObject***>**
ReferencedObject* OMVectorElement<*ReferencedObject***>::getValue(void) const**

Get the value of this OMVectorElement.

Defined in: OMContainerElementT.h

Return Value

A pointer to the *ReferencedObject*.

Class Template Arguments

ReferencedObject

The type of the referenced object.

Back to OMVectorElement

OMVectorElement::OMVectorElement

template <class *ReferencedObject***>**
OMVectorElement<*ReferencedObject***>::OMVectorElement(void)**

Constructor.

Defined in: OMContainerElementT.h

Class Template Arguments

ReferencedObject
The type of the referenced object.

Back to OMVectorElement

OMVectorElement::OMVectorElement

template <class *ReferencedObject***>**
OMVectorElement<*ReferencedObject***>::OMVectorElement(const ReferencedObject*** *pointer***)**

Constructor.

Defined in: OMContainerElementT.h

Parameters

pointer
A pointer to a *ReferencedObject*.

Class Template Arguments

ReferencedObject
The type of the referenced object.

Back to OMVectorElement

OMVectorElement::OMVectorElement

template <class *ReferencedObject***>**
OMVectorElement<*ReferencedObject***>::OMVectorElement(const OMVectorElement&ltReferencedObject>&** *rhs***)**

Copy constructor.

Defined in: OMContainerElementT.h

Parameters

rhs
The OMVectorElement to copy.

Class Template Arguments

ReferencedObject
The type of the referenced object.

Back to OMVectorElement

OMVectorElement::operator=

template <class *ReferencedObject***>**
OMVectorElement&ltReferencedObject>& OMVectorElement<*ReferencedObject***>::operator=(const OMVectorElement&ltReferencedObject>&** *rhs***)**

Assignment. This operator provides value semantics for OMVector.

Defined in: OMContainerElementT.h

Return Value

The OMVectorElement resulting from the assignment.

Parameters

rhs
The OMVectorElement to be assigned.

Class Template Arguments

ReferencedObject
The type of the referenced object.

Back to OMVectorElement

OMVectorElement::operator==

template <class *ReferencedObject***>**
bool OMVectorElement<*ReferencedObject***>::operator==(const OMVectorElement&ltReferencedObject>&** *rhs***)**

Equality. This operator provides value semantics for OMVector.

Defined in: OMContainerElementT.h

Return Value

True if the values are the same, false otherwise.

Parameters

rhs
The OMVectorElement to be compared.

Class Template Arguments

ReferencedObject
The type of the referenced object.
Back to OMVectorElement

OMVectorElement::pointer

template <class *ReferencedObject***>**
ReferencedObject* OMVectorElement<*ReferencedObject***>::pointer(void) const**

The value of this OMVectorElement as a pointer.

Defined in: OMContainerElementT.h

Return Value

A pointer to the *ReferencedObject*, if loaded.

Class Template Arguments

ReferencedObject
The type of the referenced object.
Back to OMVectorElement

OMVectorElement::setValue

template <class *ReferencedObject***>**
ReferencedObject* OMVectorElement<*ReferencedObject***>::setValue(const ReferencedObject*** *value***)**

Set the value of this OMVectorElement.

Defined in: OMContainerElementT.h

Return Value

A pointer to previous *ReferencedObject*, if any.

Parameters

value
A pointer to the new *ReferencedObject*.

Class Template Arguments

ReferencedObject
The type of the referenced object.
Back to OMVectorElement

OMVectorElement::~OMVectorElement

template <class *ReferencedObject***>**
OMVectorElement<*ReferencedObject***>::~OMVectorElement(void)**

Destructor.

Defined in: OMContainerElementT.h

Class Template Arguments

ReferencedObject
The type of the referenced object.

Back to OMVectorElement

OMVectorIterator class

OMVectorIterator **class OMVectorIterator: public OMContainerIterator**

Iterators over OMVectors.

Defined in: OMVectorIterator.h

Class Template Arguments

Element
The type of the contained elements.

Author

Tim Bingham - tjb - (Avid Technology, Inc.)

Class Members

Public members.

OMVectorIterator**(const OMVector&ltElement>& vector, OMIteratorPosition initialPosition)**
Create an **OMVectorIterator** over the given OMVector *vector* and initialize it to the given *initialPosition*. If *initialPosition* is specified as OMBefore then this **OMVectorIterator** is made ready to traverse the associated OMVector in the forward direction (increasing indexes). If *initialPosition* is specified as OMAfter then this **OMVectorIterator** is made ready to traverse the associated OMVector in the reverse direction (decreasing indexes).

virtual ~OMVectorIterator**(void)**
Destroy this **OMVectorIterator.**

virtual void reset**(OMIteratorPosition initialPosition)**
Reset this **OMVectorIterator** to the given *initialPosition*. If *initialPosition* is specified as OMBefore then this **OMVectorIterator** is made ready to traverse the associated OMVector in the

forward direction (increasing indexes). If *initialPosition* is specified as OMAfter then this **OMVectorIterator** is made ready to traverse the associated OMVector in the reverse direction (decreasing indexes).

virtual bool before**(void) const**

Is this **OMVectorIterator** positioned before the first *Element* ?

virtual bool after**(void) const**

Is this **OMVectorIterator** positioned after the last *Element* ?

virtual size_t count**(void) const**

The number of *Elements*s in the associated OMVector.

virtual bool operator++**()**

Advance this **OMVectorIterator** to the next *Element*, if any. If the end of the associated OMVector is not reached then the result is **true** , **valid** becomes **true** and **after** becomes **false** . If the end of the associated OMVector is reached then the result is **false** , **valid** becomes **false** and **after** becomes **true** .

virtual bool operator--**()**

Retreat this **OMVectorIterator** to the previous *Element*, if any. If the beginning of the associated OMVector is not reached then the result is **true** , **valid** becomes **true** and **before** becomes **false** . If the beginning of the associated OMVector is reached then the result is **false** , **valid** becomes **false** and **before** becomes **true** .

virtual Element& value**(void) const**

Return the *Element* in the associated OMVector at the position currently designated by this **OMVectorIterator.**

virtual Element setValue**(Element newElement)**

Set the *Element* in the associated OMVector at the position currently designated by this **OMVectorIterator** to *newElement*. The previous *Element* is returned.

virtual size_t index**(void) const**

Return the index of the *Element* in the associated OMVector at the position currently designated by this **OMVectorIterator**.

OMVectorIterator::after

template <class *Element***>**
bool OMVectorIterator<*Element***>::after(void) const**

Is this OMVectorIterator positioned after the last *Element* ?

Defined in: OMVectorIteratorT.h

Return Value

true if this OMVectorIterator is positioned after the last *Element*, **false** otherwise.

Class Template Arguments

Element

The type of the contained elements.

Back to OMVectorIterator

OMVectorIterator::before

template <class *Element***>**
bool OMVectorIterator<*Element***>::before(void) const**

Is this OMVectorIterator positioned before the first *Element* ?

Defined in: OMVectorIteratorT.h

Return Value

true if this OMVectorIterator is positioned before the first *Element*, **false** otherwise.

Class Template Arguments

Element
The type of the contained elements.

Back to OMVectorIterator

OMVectorIterator::count

template <class *Element***>**
size_t OMVectorIterator<*Element***>::count(void) const**

The number of *Elementss* in the associated OMVector.

Defined in: OMVectorIteratorT.h

Return Value

The number of *Elementss*

Class Template Arguments

Element
The type of the contained elements.

Back to OMVectorIterator

OMVectorIterator::index

template <class *Element***>**
size_t OMVectorIterator<*Element***>::index(void) const**

Return the index of the *Element* in the associated OMVector at the position currently designated by this OMVectorIterator.

Defined in: OMVectorIteratorT.h

Return Value

The index of the current position.

Class Template Arguments

Element
The type of the contained elements.

Back to OMVectorIterator

OMVectorIterator::OMVectorIterator

template <class *Element***>**
OMVectorIterator<*Element***>::OMVectorIterator(const OMVector<Element>& *vector*, OMIteratorPosition *initialPosition*)**

Create an OMVectorIterator over the given OMVector *vector* and initialize it to the given *initialPosition*. If *initialPosition* is specified as OMBefore then this OMVectorIterator is made ready to traverse the associated OMVector in the forward direction (increasing indexes). If *initialPosition* is specified as OMAfter then this OMVectorIterator is made ready to traverse the associated OMVector in the reverse direction (decreasing indexes).

Defined in: OMVectorIteratorT.h

Parameters

vector
The OMVector over which to iterate.

initialPosition
The initial position for this OMVectorIterator.

Class Template Arguments

Element
The type of the contained elements.

Back to OMVectorIterator

OMVectorIterator::operator++

template <class *Element***>**
bool OMVectorIterator<*Element***>::operator++(void)**

Advance this OMVectorIterator to the next *Element*, if any. If the end of the associated OMVector is not reached then the result is **true** , **valid** becomes **true** and **after** becomes **false** . If the end of the associated OMVector is reached then the result is **false** , **valid** becomes **false** and **after** becomes **true** .

Defined in: OMVectorIteratorT.h

Return Value

false if this OMVectorIterator has passed the last *Element*, **true** otherwise.

Class Template Arguments

Element
The type of the contained elements.

Back to OMVectorIterator

OMVectorIterator::operator--

template <class *Element***>**
bool OMVectorIterator<*Element***>::operator--(void)**

Retreat this OMVectorIterator to the previous *Element*, if any. If the beginning of the associated OMVector is not reached then the result is **true** , **valid** becomes **true** and **before** becomes **false** . If the beginning of the associated OMVector is reached then the result is **false** , **valid** becomes **false** and **before** becomes **true** .

Defined in: OMVectorIteratorT.h

Return Value

false if this OMVectorIterator has passed the first *Element*, **true** otherwise.

Class Template Arguments

Element
The type of the contained elements.

Back to OMVectorIterator

OMVectorIterator::reset

template <class *Element***>**
void OMVectorIterator<*Element***>::reset(OMIteratorPosition** *initialPosition***)**

Reset this OMVectorIterator to the given *initialPosition*. If *initialPosition* is specified as OMBefore then this OMVectorIterator is made ready to traverse the associated OMVector in the forward direction (increasing indexes). If *initialPosition* is specified as OMAfter then this OMVectorIterator is made ready to traverse the associated OMVector in the reverse direction (decreasing indexes).

Defined in: OMVectorIteratorT.h

Parameters

initialPosition
The position to which this OMVectorIterator should be reset.

Class Template Arguments

Element
The type of the contained elements.

Back to OMVectorIterator

OMVectorIterator::setValue

template <class *Element***>**
Element OMVectorIterator<*Element***>::setValue(Element** *newElement***)**

Set the *Element* in the associated OMVector at the position currently designated by this OMVectorIterator to *newElement*. The previous *Element* is returned.

Defined in: OMVectorIteratorT.h

Return Value

The previous *Element*.

Parameters

newElement
The new *Element*.

Class Template Arguments

Element
The type of the contained elements.

Back to OMVectorIterator

OMVectorIterator::value

template <class *Element***>**
Element& OMVectorIterator<*Element***>::value(void) const**

Return the *Element* in the associated OMVector at the position currently designated by this OMVectorIterator.

Defined in: OMVectorIteratorT.h

Return Value

The *Element* at the current position.

Class Template Arguments

Element
The type of the contained elements.
Back to OMVectorIterator

OMVectorIterator::~OMVectorIterator

template <class *Element***>**
OMVectorIterator<*Element***>::~OMVectorIterator(void)**

Destroy this OMVectorIterator.

Defined in: OMVectorIteratorT.h

Class Template Arguments

Element
The type of the contained elements.
Back to OMVectorIterator

OMWeakObjectReference class

OMWeakObjectReference **class OMWeakObjectReference: public OMObjectReference**

Persistent weak references to persistent objects.

Defined in: OMObjectReference.h

Author

Tim Bingham - tjb - (Avid Technology, Inc.)

Class Members

Public members.
OMWeakObjectReference**(void)**
Constructor.
OMWeakObjectReference**(OMProperty* property)**
Constructor.
OMWeakObjectReference**(OMProperty* property, OMUniqueObjectIdentification identification, OMPropertyTag targetTag)**
Constructor.
OMWeakObjectReference**(const OMWeakObjectReference&)**
Copy constructor.
virtual ~OMWeakObjectReference**(void)**
Destructor.
virtual bool isVoid**(void) const**
Is this **OMWeakObjectReference** void ?
OMWeakObjectReference& operator=**(const OMWeakObjectReference& rhs)**

Assignment. This operator provides value semantics for OMContainer. This operator does not provide assignment of object references.

bool operator==**(const OMWeakObjectReference& rhs) const**

Equality. This operator provides value semantics for OMContainer. This operator does not provide equality of object references.

virtual void save**(void) const**

Save this **OMWeakObjectReference.**

virtual void close**(void)**

Close this **OMWeakObjectReference.**

virtual void detach**(void)**

Detach this **OMWeakObjectReference.**

virtual void restore**(void)**

Restore this **OMWeakObjectReference.**

virtual OMStorable* getValue**(void) const**

Get the value of this **OMWeakObjectReference.** The value is a pointer to the referenced OMStorable**.**

virtual OMStorable* setValue**(const OMUniqueObjectIdentification& identification, const OMStorable* value)**

Set the value of this **OMWeakObjectReference.** The value is a pointer to the referenced OMStorable**.**

OMWeakObjectReference::close

void OMWeakObjectReference::close(void)

Close this OMWeakObjectReference.

Defined in: OMObjectReference.cpp

Back to OMWeakObjectReference

OMWeakObjectReference::detach

void OMWeakObjectReference::detach(void)

Detach this OMWeakObjectReference.

Defined in: OMObjectReference.cpp

Back to OMWeakObjectReference

OMWeakObjectReference::getValue

OMStorable* OMWeakObjectReference::getValue(void) const

Get the value of this OMWeakObjectReference. The value is a pointer to the referenced OMStorable.

Defined in: OMObjectReference.cpp

Return Value

A pointer to the referenced OMStorable.

Back to OMWeakObjectReference

OMWeakObjectReference::isVoid

bool OMWeakObjectReference::isVoid(void) const

Is this OMWeakObjectReference void ?

Defined in: OMObjectReference.cpp

Return Value

True if this OMWeakObjectReference is void, false otherwise.

Back to OMWeakObjectReference

OMWeakObjectReference::OMWeakObjectReference

OMWeakObjectReference::OMWeakObjectReference(void)

Constructor.

Defined in: OMObjectReference.cpp

Back to OMWeakObjectReference

OMWeakObjectReference::OMWeakObjectReference

OMWeakObjectReference::OMWeakObjectReference(OMProperty* *property*, **OMUniqueObjectIdentification** *identification*, **OMPropertyTag** *targetTag***)**

Constructor.

Defined in: OMObjectReference.cpp

Parameters

property

The OMProperty that contains this OMWeakObjectReference.

identification

The unique key of this OMWeakObjectReference.

targetTag

A tag identifying the OMStrongReferenceSetProperty in which the target resides.

Back to OMWeakObjectReference

OMWeakObjectReference::OMWeakObjectReference

OMWeakObjectReference::OMWeakObjectReference(const OMWeakObjectReference& *rhs***)**

Copy constructor.

Defined in: OMObjectReference.cpp

Parameters

rhs

The OMWeakObjectReference to copy.

Back to OMWeakObjectReference

OMWeakObjectReference::OMWeakObjectReference

OMWeakObjectReference::OMWeakObjectReference(OMProperty* *property***)**

Constructor.

Defined in: OMObjectReference.cpp

Parameters

property

The OMProperty that contains this OMWeakObjectReference.

Back to OMWeakObjectReference

OMWeakObjectReference::operator=

OMWeakObjectReference& OMWeakObjectReference::operator=(const OMWeakObjectReference& *rhs***)**

Assignment. This operator provides value semantics for OMContainer. This operator does not provide assignment of object references.

Defined in: OMObjectReference.cpp

Return Value

The OMWeakObjectReference resulting from the assignment.

Parameters

rhs
The OMWeakObjectReference to be assigned.

Back to OMWeakObjectReference

OMWeakObjectReference::operator==

bool OMWeakObjectReference::operator==(const OMWeakObjectReference& *rhs*) const

Equality.

Defined in: OMObjectReference.cpp

Return Value

True if the values are the same, false otherwise.

Parameters

rhs
The OMWeakObjectReference to be compared.

Back to OMWeakObjectReference

OMWeakObjectReference::restore

void OMWeakObjectReference::restore(void)

Restore this OMWeakObjectReference.

Defined in: OMObjectReference.cpp

Back to OMWeakObjectReference

OMWeakObjectReference::save

void OMWeakObjectReference::save(void) const

Save this OMWeakObjectReference.

Defined in: OMObjectReference.cpp

Back to OMWeakObjectReference

OMWeakObjectReference::setValue

OMStorable* OMWeakObjectReference::setValue(const OMUniqueObjectIdentification& *identification*, **const OMStorable*** *value***)**

Set the value of this OMWeakObjectReference. The value is a pointer to the referenced OMStorable.

Defined in: OMObjectReference.cpp

Return Value

A pointer to previous OMStorable, if any.

Parameters

identification
TBS
value
A pointer to the new OMStorable.

Back to OMWeakObjectReference

OMWeakObjectReference::~OMWeakObjectReference

OMWeakObjectReference::~OMWeakObjectReference(void)

Destructor.

Defined in: OMObjectReference.cpp

Back to OMWeakObjectReference

OMWeakReference class

OMWeakReference **class OMWeakReference: public OMReferenceProperty**

Persistent weak reference (pointer to shared object) properties supported by the Object Manager.

Defined in: OMWeakReference.h

Author

Tim Bingham - tjb - (Avid Technology, Inc.)

Class Members

Public members.

OMWeakReference**(const OMPropertyId propertyId, const wchar_t* name)**

Constructor.

~OMWeakReference**(void)**

Destructor.

OMWeakReference::OMWeakReference

OMWeakReference::OMWeakReference(void)

Constructor.

Defined in: OMWeakReference.cpp

Back to OMWeakReference

OMWeakReference::~OMWeakReference

OMWeakReference::~OMWeakReference(void)

Destructor.

Defined in: OMWeakReference.cpp

Back to OMWeakReference

OMWeakReferenceProperty class

OMWeakReferenceProperty **class OMWeakReferenceProperty: public OMWeakReference**

Persistent weak reference (pointer to shared object) properties supported by the Object Manager.

Defined in: OMWeakRefProperty.h

Class Template Arguments

ReferencedObject

The type of the referenced (pointed to) object. This type must be a descendant of OMStorable.

Author

Tim Bingham - tjb - (Avid Technology, Inc.)

Class Members

Public members.

OMWeakReferenceProperty**(const OMPropertyId propertyId, const wchar_t* name, const wchar_t* targetName, const OMPropertyId keyPropertyId)**

Constructor.

OMWeakReferenceProperty**(const OMPropertyId propertyId, const wchar_t* name, const OMPropertyId keyPropertyId, const OMPropertyId* targetPropertyPath)**

Constructor.

virtual ~OMWeakReferenceProperty(void)

Destructor.

virtual void getValue**(ReferencedObject*& object) const**

Get the value of this **OMWeakReferenceProperty.**

virtual ReferencedObject* setValue**(const ReferencedObject* object)**

set the value of this **OMWeakReferenceProperty.**

virtual ReferencedObject* clearValue**(void)**

Clear the value of this **OMWeakReferenceProperty.**

OMWeakReferenceProperty&ltReferencedObject>& operator=

Assignment operator.

ReferencedObject* operator->**(void)**

Dereference operator.

const ReferencedObject* operator->**(void) const**

Dereference operator.

operator ReferencedObject*() const

Type conversion. Convert an **OMWeakReferenceProperty** into a pointer to the referenced (pointed to) *ReferencedObject*.

virtual void save**(void) const**

Save this **OMWeakReferenceProperty.**

virtual void close**(void)**

close this **OMWeakReferenceProperty.**

virtual void restore**(size_t externalSize)**

Restore this **OMWeakReferenceProperty,** the external (persisted) size of the **OMWeakReferenceProperty** is *externalSize*.

virtual bool isVoid**(void) const**

Is this **OMWeakReferenceProperty** void ?

virtual void getBits**(OMByte* bits, size_t size) const**

Get the raw bits of this **OMWeakReferenceProperty.** The raw bits are copied to the buffer at address *bits* which is *size* bytes in size.

virtual void setBits**(const OMByte* bits, size_t size)**

Set the raw bits of this **OMWeakReferenceProperty.** The raw bits are copied from the buffer at address *bits* which is *size* bytes in size.

virtual OMObject* getObject**(void) const**

Get the value of this **OMWeakReferenceProperty.**

virtual OMObject* setObject**(const OMObject* object)**

set the value of this **OMWeakReferenceProperty.**

virtual OMStrongReferenceSet* targetSet(void) const

The OMStrongReferenceSet in which the object referenced by this **OMWeakReferenceProperty** must reside.

OMWeakReferenceProperty::clearValue

template <class *ReferencedObject***>**

ReferencedObject* OMWeakReferenceProperty<*ReferencedObject***>::clearValue(void)**

Clear the value of this OMWeakReferenceProperty.

Defined in: OMWeakRefPropertyT.h

Return Value

A pointer to the old *ReferencedObject*. If lazy loading is enabled and the referenced object was never loaded the value returned is 0.

Class Template Arguments

ReferencedObject
: The type of the referenced (contained) object. This type must be a descendant of OMStorable.

Back to OMWeakReferenceProperty

OMWeakReferenceProperty::close

template <class *ReferencedObject***>**
void OMWeakReferenceProperty<*ReferencedObject***>::close(void)**

Close this OMWeakReferenceProperty.

Defined in: OMWeakRefPropertyT.h

Class Template Arguments

ReferencedObject
: The type of the referenced (contained) object. This type must be a descendant of OMStorable.

Back to OMWeakReferenceProperty

OMWeakReferenceProperty::getBits

template <class *ReferencedObject***>**
void OMWeakReferenceProperty<*ReferencedObject***>::getBits(OMByte*** *bits***, size_t** *ANAME***) const**

Get the raw bits of this OMWeakReferenceProperty. The raw bits are copied to the buffer at address *bits* which is *size* bytes in size.

Defined in: OMWeakRefPropertyT.h

Parameters

bits
: The address of the buffer into which the raw bits are copied.

ANAME

The size of the buffer.

Class Template Arguments

ReferencedObject
The type of the referenced object. This type must be a descendant of OMStorable.

Back to OMWeakReferenceProperty

OMWeakReferenceProperty::getObject

template <class *ReferencedObject***>**
OMObject* OMWeakReferenceProperty<*ReferencedObject***>::getObject(void) const**

Get the value of this OMWeakReferenceProperty.

Defined in: OMWeakRefPropertyT.h

Return Value

A pointer to an OMObject.

Class Template Arguments

ReferencedObject
The type of the referenced (pointed to) object. This type must be a descendant of OMStorable.

Back to OMWeakReferenceProperty

OMWeakReferenceProperty::getValue

template <class *ReferencedObject***>**
void OMWeakReferenceProperty<*ReferencedObject***>::getValue(ReferencedObject*& *object***) const**

Get the value of this OMWeakReferenceProperty.

Defined in: OMWeakRefPropertyT.h

Parameters

object
A pointer to a *ReferencedObject* by reference.

Class Template Arguments

ReferencedObject

The type of the referenced (pointed to) object. This type must be a descendant of OMStorable.

Back to OMWeakReferenceProperty

OMWeakReferenceProperty::isVoid

template <class *ReferencedObject***>**
bool OMWeakReferenceProperty<*ReferencedObject***>::isVoid(void) const**

Is this OMWeakReferenceProperty void ?

Defined in: OMWeakRefPropertyT.h

Return Value

True if this OMWeakReferenceProperty is void, false otherwise

Class Template Arguments

ReferencedObject
The type of the referenced (contained) object. This type must be a descendant of OMStorable.

Back to OMWeakReferenceProperty

OMWeakReferenceProperty::OMWeakReferenceProperty

OMWeakReferenceProperty::OMWeakReferenceProperty(const OMPropertyId *propertyId***, const wchar_t*** *name***, const OMPropertyId** *keyPropertyId***, const OMPropertyId*** *targetPropertyPath***)**

Constructor.

Defined in: OMWeakRefPropertyT.h

Parameters

propertyId
The property id.

name
The name of this OMWeakReferenceProperty.

keyPropertyId
The id of the property by which the *ReferencedObject* is uniquely identified (the key).

targetPropertyPath
The name (as a list of pids) of the the OMProperty instance (a set property) in which the object referenced by this OMWeakReferenceProperty resides.

Back to OMWeakReferenceProperty

OMWeakReferenceProperty::OMWeakReferenceProperty

OMWeakReferenceProperty::OMWeakReferenceProperty(const OMPropertyId *propertyId*, **const wchar_t*** *name*, **const wchar_t*** *targetName*, **const OMPropertyId** *keyPropertyId***)**

Constructor.

Defined in: OMWeakRefPropertyT.h

Parameters

propertyId
: The property id.

name
: The name of this OMWeakReferenceProperty.

targetName
: The name (as a string) of the the OMProperty instance (a set property) in which the object referenced by this OMWeakReferenceProperty resides.

keyPropertyId
: The id of the property by which the *ReferencedObject* is uniquely identified (the key).

Back to OMWeakReferenceProperty

OMWeakReferenceProperty::operator ReferencedObject*

OMWeakReferenceProperty::operator ReferencedObject* (void) const

Type conversion. Convert an OMWeakReferenceProperty into a pointer to the referenced (pointed to) *ReferencedObject*.

Defined in: OMWeakRefPropertyT.h

Return Value

The result of the conversion as a value of type pointer to *ReferencedObject*.

Back to OMWeakReferenceProperty

OMWeakReferenceProperty::operator->

template <class *ReferencedObject***>**
ReferencedObject* OMWeakReferenceProperty<*ReferencedObject***>::operator->(void)**

Dereference operator.

Defined in: OMWeakRefPropertyT.h

Return Value

A pointer to a *ReferencedObject* by value.

Class Template Arguments

ReferencedObject
: The type of the referenced (contained) object. This type must be a descendant of OMStorable

Back to OMWeakReferenceProperty

OMWeakReferenceProperty::operator=

template <class *ReferencedObject***>**
OMWeakReferenceProperty&ltReferencedObject>&
OMWeakReferenceProperty<*ReferencedObject***>::operator=(const ReferencedObject*** *value***)**

Assignment operator.

Defined in: OMWeakRefPropertyT.h

Return Value

The result of the assignment.

Parameters

value
: A pointer to a *ReferencedObject* by value.

Class Template Arguments

ReferencedObject
: The type of the referenced (pointed to) object. This type must be a descendant of OMStorable.

Back to OMWeakReferenceProperty

OMWeakReferenceProperty::restore

template <class *ReferencedObject***>**
void OMWeakReferenceProperty<*ReferencedObject***>::restore(size_t** *externalSize***)**

Restore this OMWeakReferenceProperty, the external (persisted) size of the OMWeakReferenceProperty is *externalSize*.

Defined in: OMWeakRefPropertyT.h

Parameters

externalSize
The external (persisted) size of the OMWeakReferenceProperty.

Class Template Arguments

ReferencedObject
The type of the referenced (pointed to) object. This type must be a descendant of OMStorable.

Back to OMWeakReferenceProperty

OMWeakReferenceProperty::save

template <class *ReferencedObject***>**
void OMWeakReferenceProperty<*ReferencedObject***>::save(void) const**

Save this OMWeakReferenceProperty.

Defined in: OMWeakRefPropertyT.h

Class Template Arguments

ReferencedObject
The type of the referenced (pointed to) object. This type must be a descendant of OMStorable.

Back to OMWeakReferenceProperty

OMWeakReferenceProperty::setBits

template <class *ReferencedObject***>**
void OMWeakReferenceProperty<*ReferencedObject***>::setBits(const OMByte*** *bits***, size_t** *ANAME***)**

Set the raw bits of this OMWeakReferenceProperty. The raw bits are copied from the buffer at address *bits* which is *size* bytes in size.

Defined in: OMWeakRefPropertyT.h

Parameters

bits
The address of the buffer into which the raw bits are copied.

ANAME
The size of the buffer.

Class Template Arguments

ReferencedObject

The type of the referenced object. This type must be a descendant of OMStorable.

Back to OMWeakReferenceProperty

OMWeakReferenceProperty::setObject

template <class *ReferencedObject***>**
OMObject* OMWeakReferenceProperty<*ReferencedObject***>::setObject(const OMObject*** *object***)**

Set the value of this OMWeakReferenceProperty.

Defined in: OMWeakRefPropertyT.h

Return Value

A pointer to the old OMObject. If lazy loading is enabled and the referenced object was never loaded the value returned is 0.

Parameters

object
A pointer to the new OMObject.

Class Template Arguments

ReferencedObject
The type of the referenced (contained) object. This type must be a descendant of OMStorable.

Back to OMWeakReferenceProperty

OMWeakReferenceProperty::setValue

template <class *ReferencedObject***>**
ReferencedObject* OMWeakReferenceProperty<*ReferencedObject***>::setValue(const ReferencedObject*** *object***)**

Set the value of this OMWeakReferenceProperty.

Defined in: OMWeakRefPropertyT.h

Return Value

A pointer to the old *ReferencedObject*. If lazy loading is enabled and the referenced object was never loaded the value returned is 0.

Parameters

object
A pointer to the new *ReferencedObject*.

Class Template Arguments

ReferencedObject
The type of the referenced (contained) object. This type must be a descendant of OMStorable.

Back to OMWeakReferenceProperty

OMWeakReferenceSet class

OMWeakReferenceSet **class OMWeakReferenceSet: public OMReferenceSetProperty**

Persistent sets of uniquely identified weakly referenced (non-contained) objects supported by the Object Manager. Objects are accessible by unique identifier (the key). The objects are not ordered. Duplicates objects are not allowed.

Defined in: OMWeakReferenceSet.h

Author

Tim Bingham - tjb - (Avid Technology, Inc.)

Class Members

Public members.
OMWeakReferenceSet**(const OMPropertyId propertyId, const wchar_t* name)**
Constructor.
virtual ~OMWeakReferenceSet**(void)**
Destructor.

OMWeakReferenceSet::OMWeakReferenceSet

OMWeakReferenceSet::OMWeakReferenceSet(const OMPropertyId *propertyId***, const wchar_t*** *name***)**

Constructor.

Defined in: OMWeakReferenceSet.cpp

Parameters

propertyId
The property id.
name
The name of this OMWeakReferenceSet.

Back to OMWeakReferenceSet

OMWeakReferenceSet::~OMWeakReferenceSet

OMWeakReferenceSet::~OMWeakReferenceSet(void)

Destructor.

Defined in: OMWeakReferenceSet.cpp

Back to OMWeakReferenceSet

OMWeakReferenceSetElement class

OMWeakReferenceSetElement **class OMWeakReferenceSetElement: public OMContainerElement**

Elements of Object Manager reference sets.

Defined in: OMContainerElement.h

Author

Tim Bingham - tjb - (Avid Technology, Inc.)

Class Members

Public members.

OMWeakReferenceSetElement**(void)**
: Constructor.

OMWeakReferenceSetElement**(OMProperty* property, OMUniqueObjectIdentification identification, OMPropertyTag targetTag)**
: Constructor.

OMWeakReferenceSetElement**(const OMWeakReferenceSetElement& rhs)**
: Copy constructor.

~OMWeakReferenceSetElement**(void)**
: Destructor.

OMWeakReferenceSetElement& operator=**(const OMWeakReferenceSetElement& rhs)**
: Assignment. This operator provides value semantics for OMSet. This operator does not provide assignment of object references.

bool operator==**(const OMWeakReferenceSetElement& rhs) const**
: Equality. This operator provides value semantics for OMSet. This operator does not provide equality of object references.

OMStorable* setValue**(const OMUniqueObjectIdentification& identification, const OMStorable* value)**
: Set the value of this **OMWeakReferenceSetElement.**

OMUniqueObjectIdentification identification**(void) const**
: The unique key of this **OMWeakReferenceSetElement.**

OMWeakReferenceSetElement::identification

OMUniqueObjectIdentification OMWeakReferenceSetElement::identification(void)

The unique key of this OMWeakReferenceSetElement.

Defined in: OMContainerElement.cpp

Return Value

The unique key of this OMWeakReferenceSetElement.

Back to OMWeakReferenceSetElement

OMWeakReferenceSetElement::OMWeakReferenceSetElement

OMWeakReferenceSetElement::OMWeakReferenceSetElement(const OMWeakReferenceSetElement& *rhs***)**

Copy constructor.

Defined in: OMContainerElement.cpp

Parameters

rhs
The OMWeakReferenceSetElement to copy.

Back to OMWeakReferenceSetElement

OMWeakReferenceSetElement::OMWeakReferenceSetElement

OMWeakReferenceSetElement::OMWeakReferenceSetElement(void)

Constructor.

Defined in: OMContainerElement.cpp

Back to OMWeakReferenceSetElement

OMWeakReferenceSetElement::OMWeakReferenceSetElement

OMWeakReferenceSetElement::OMWeakReferenceSetElement(OMProperty* *property***, OMUniqueObjectIdentification** *identification***, OMPropertyTag** *targetTag***)**

Constructor.

Defined in: OMContainerElement.cpp

Parameters

property
The OMProperty (a set property) that contains this OMWeakReferenceSetElement.
identification
The unique key of this OMWeakReferenceSetElement.
targetTag
A tag identifying the OMStrongReferenceSetProperty in which the target resides.
Back to OMWeakReferenceSetElement

OMWeakReferenceSetElement::operator=

OMWeakReferenceSetElement& OMWeakReferenceSetElement::operator=(const OMWeakReferenceSetElement& *rhs*)

Assignment. This operator provides value semantics for OMSet. This operator does not provide assignment of object references.

Defined in: OMContainerElement.cpp

Return Value

The OMWeakReferenceSetElement resulting from the assignment.

Parameters

rhs
The OMWeakReferenceSetElement to be assigned.
Back to OMWeakReferenceSetElement

OMWeakReferenceSetElement::operator==

bool OMWeakReferenceSetElement::operator==(const OMWeakReferenceSetElement& *rhs*)

Equality. This operator provides value semantics for OMSet. This operator does not provide equality of object references.

Defined in: OMContainerElement.cpp

Return Value

True if the values are the same, false otherwise.

Parameters

rhs
The OMWeakReferenceSetElement to be compared.
Back to OMWeakReferenceSetElement

OMWeakReferenceSetElement::setValue

OMStorable* OMWeakReferenceSetElement::setValue(const OMUniqueObjectIdentification& *identification*, **const OMStorable*** *value***)**

Set the value of this OMWeakReferenceSetElement.

Defined in: OMContainerElement.cpp

Return Value

A pointer to previous OMStorable, if any.

Parameters

identification
: TBS

value
: A pointer to the new OMStorable.

Back to OMWeakReferenceSetElement

OMWeakReferenceSetElement::~OMWeakReferenceSetElement

OMWeakReferenceSetElement::~OMWeakReferenceSetElement(void)

Destructor.

Defined in: OMContainerElement.cpp

Back to OMWeakReferenceSetElement

OMWeakReferenceSetIterator class

OMWeakReferenceSetIterator **class OMWeakReferenceSetIterator: public OMReferenceContainerIterator**

Iterators over OMWeakReferenceSetProperty s.

Defined in: OMWeakReferenceSetIter.h

Class Template Arguments

ReferencedObject
: The type of the contained objects.

Author

Class Members

Public members.

OMWeakReferenceSetIterator**(const OMWeakReferenceSetProperty&ltReferencedObject>& set, OMIteratorPosition initialPosition = OMBefore)**

Create an **OMWeakReferenceSetIterator** over the given OMWeakReferenceSetProperty *set* and initialize it to the given *initialPosition*. If *initialPosition* is specified as OMBefore then this **OMWeakReferenceSetIterator** is made ready to traverse the associated OMWeakReferenceSetProperty in the forward direction (increasing *Key*s). If *initialPosition* is specified as OMAfter then this **OMWeakReferenceSetIterator** is made ready to traverse the associated OMWeakReferenceSetProperty in the reverse direction (decreasing *Key*s).

virtual ~OMWeakReferenceSetIterator**(void)**

Destroy this **OMWeakReferenceSetIterator.**

virtual OMReferenceContainerIterator* copy**(void) const**

Create a copy of this **OMWeakReferenceSetIterator.**

virtual void reset**(OMIteratorPosition initialPosition = OMBefore)**

Reset this **OMWeakReferenceSetIterator** to the given *initialPosition*. If *initialPosition* is specified as OMBefore then this **OMWeakReferenceSetIterator** is made ready to traverse the associated OMWeakReferenceSetProperty in the forward direction (increasing *Key*s). If *initialPosition* is specified as OMAfter then this **OMWeakReferenceSetIterator** is made ready to traverse the associated OMWeakReferenceSetProperty in the reverse direction (decreasing *Key*s).

virtual bool before**(void) const**

Is this **OMWeakReferenceSetIterator** positioned before the first *ReferencedObject* ?

virtual bool after**(void) const**

Is this **OMWeakReferenceSetIterator** positioned after the last *ReferencedObject* ?

virtual bool valid**(void) const**

Is this **OMWeakReferenceSetIterator** validly positioned on a *ReferencedObject* ?

virtual size_t count**(void) const**

The number of *ReferencedObject*s in the associated OMWeakReferenceSetProperty.

virtual bool operator++**()**

Advance this **OMWeakReferenceSetIterator** to the next *ReferencedObject*, if any. If the end of the associated OMWeakReferenceSetProperty is not reached then the result is **true** , **valid** becomes **true** and **after** becomes **false** . If the end of the associated OMWeakReferenceSetProperty is reached then the result is **false** , **valid** becomes **false** and **after** becomes **true** .

virtual bool operator--**()**

Retreat this **OMWeakReferenceSetIterator** to the previous *ReferencedObject*, if any. If the beginning of the associated OMWeakReferenceSetProperty is not reached then the result is **true** , **valid** becomes **true** and **before** becomes **false** . If the beginning of the associated OMWeakReferenceSetProperty is reached then the result is **false** , **valid** becomes **false** and **before** becomes **true** .

virtual ReferencedObject* value**(void) const**

Return the *ReferencedObject* in the associated OMWeakReferenceSetProperty at the position currently designated by this **OMWeakReferenceSetIterator.**

virtual ReferencedObject* setValue**(const ReferencedObject* newObject)**

Set the *ReferencedObject* in the associated OMWeakReferenceSetProperty at the position currently designated by this **OMWeakReferenceSetIterator** to *newObject*. The previous

ReferencedObject, if any, is returned. To preserve the ordering of *Key*s, the *Key* of *newObject* must be the same as that of the existing *ReferencedObject*.

virtual ReferencedObject* clearValue**(void)**

Set the *ReferencedObject* in the associated OMWeakReferenceSetProperty at the position currently designated by this **OMWeakReferenceSetIterator** to 0. The previous *ReferencedObject*, if any, is returned.

OMUniqueObjectIdentification identification**(void) const**

Return the *Key* of the *ReferencedObject* in the associated OMWeakReferenceSetProperty at the position currently designated by this **OMWeakReferenceSetIterator**.

virtual OMObject* currentObject**(void) const**

Return the *OMObject* in the associated reference container property at the position currently designated by this **OMWeakReferenceSetIterator**.

OMWeakReferenceSetIterator**(const SetIterator& iter)**

Create an **OMWeakReferenceSetIterator** given an underlying OMSetIterator.

OMWeakReferenceSetIterator::after

template <class *ReferencedObject***>**
bool OMWeakReferenceSetIterator<*ReferencedObject***>::after(void) const**

Is this OMWeakReferenceSetIterator positioned after the last *ReferencedObject* ?

Defined in: OMWeakReferenceSetIterT.h

Return Value

true if this OMWeakReferenceSetIterator is positioned after the last *ReferencedObject*, **false** otherwise.

Class Template Arguments

ReferencedObject

The type of the contained objects.

Back to OMWeakReferenceSetIterator

OMWeakReferenceSetIterator::before

template <class *ReferencedObject***>**
bool OMWeakReferenceSetIterator<*ReferencedObject***>::before(void) const**

Is this OMWeakReferenceSetIterator positioned before the first *ReferencedObject* ?

Defined in: OMWeakReferenceSetIterT.h

Return Value

true if this OMWeakReferenceSetIterator is positioned before the first *ReferencedObject*, **false** otherwise.

Class Template Arguments

ReferencedObject
The type of the contained objects.

Back to OMWeakReferenceSetIterator

OMWeakReferenceSetIterator::clearValue

template <class *ReferencedObject***>**
ReferencedObject* OMWeakReferenceSetIterator<*ReferencedObject***>::clearValue(void)**

Set the *ReferencedObject* in the associated OMWeakReferenceSetProperty at the position currently designated by this OMWeakReferenceSetIterator to 0. The previous *ReferencedObject*, if any, is returned.

Defined in: OMWeakReferenceSetIterT.h

Return Value

The previous *ReferencedObject* if any, otherwise 0.

Class Template Arguments

ReferencedObject
The type of the contained objects.

Back to OMWeakReferenceSetIterator

OMWeakReferenceSetIterator::copy

template <class *ReferencedObject***>**
OMReferenceContainerIterator* OMWeakReferenceSetIterator<*ReferencedObject***>::copy(void) const**

Create a copy of this OMWeakReferenceSetIterator.

Defined in: OMWeakReferenceSetIterT.h

Return Value

The new OMWeakReferenceSetIterator.

Class Template Arguments

ReferencedObject
The type of the contained objects.

Back to OMWeakReferenceSetIterator

OMWeakReferenceSetIterator::count

template <class *ReferencedObject***>**
size_t OMWeakReferenceSetIterator<*ReferencedObject***>::count(void) const**

The number of *ReferencedObject*s in the associated OMWeakReferenceSetProperty.

Defined in: OMWeakReferenceSetIterT.h

Return Value

The number of *ReferencedObject*s

Class Template Arguments

ReferencedObject
The type of the contained objects.

Back to OMWeakReferenceSetIterator

OMWeakReferenceSetIterator::currentObject

template <class *ReferencedObject***>**
OMObject* OMWeakReferenceSetIterator<*ReferencedObject***>::currentObject(void) const**

Return the OMObject in the associated OMWeakReferenceSetProperty at the position currently designated by this OMWeakReferenceSetIterator.

Defined in: OMWeakReferenceSetIterT.h

Return Value

The OMObject at the current position.

Class Template Arguments

ReferencedObject
The type of the contained objects.

Back to OMWeakReferenceSetIterator

OMWeakReferenceSetIterator::identification

template <class *ReferencedObject***>**
OMUniqueObjectIdentification OMWeakReferenceSetIterator<*ReferencedObject***>::identification(void) const**

Return the *Key* of the *ReferencedObject* in the associated OMWeakReferenceSetProperty at the position currently designated by this OMWeakReferenceSetIterator.

Defined in: OMWeakReferenceSetIterT.h

Return Value

The *Key* at the current position.

Class Template Arguments

ReferencedObject
The type of the contained objects.

Back to OMWeakReferenceSetIterator

OMWeakReferenceSetIterator::OMWeakReferenceSetIterator

template <class *ReferencedObject***>**
OMWeakReferenceSetIterator<*ReferencedObject***>::OMWeakReferenceSetIterator(const OMWeakReferenceSetProperty&ltReferencedObject>&** *set***, OMIteratorPosition** *initialPosition***)**

Create an OMWeakReferenceSetIterator over the given OMWeakReferenceSetProperty *set* and initialize it to the given *initialPosition*. If *initialPosition* is specified as OMBefore then this OMWeakReferenceSetIterator is made ready to traverse the associated OMWeakReferenceSetProperty in the forward direction (increasing *Keys*). If *initialPosition* is specified as OMAfter then this OMWeakReferenceSetIterator is made ready to traverse the associated OMWeakReferenceSetProperty in the reverse direction (decreasing *Keys*).

Defined in: OMWeakReferenceSetIterT.h

Parameters

set
The OMWeakReferenceSet over which to iterate.

initialPosition
The initial position for this OMWeakReferenceSetIterator.

Class Template Arguments

ReferencedObject
The type of the contained objects.

Back to OMWeakReferenceSetIterator

OMWeakReferenceSetIterator::OMWeakReferenceSetIterator

template <class *ReferencedObject***>**
OMWeakReferenceSetIterator<*ReferencedObject***>::OMWeakReferenceSetIterator(const SetIterator&** *iter***)**

Create an OMWeakReferenceSetIterator given an underlying OMSetIterator.

Defined in: OMWeakReferenceSetIterT.h

Parameters

iter
The underlying OMSetIterator.

Class Template Arguments

ReferencedObject
The type of the contained objects.

Back to OMWeakReferenceSetIterator

OMWeakReferenceSetIterator::operator++

template <class *ReferencedObject***>**
bool OMWeakReferenceSetIterator<*ReferencedObject***>::operator++(void)**

Advance this OMWeakReferenceSetIterator to the next *ReferencedObject*, if any. If the end of the associated OMWeakReferenceSetProperty is not reached then the result is **true** , **valid** becomes **true** and **after** becomes **false** . If the end of the associated OMWeakReferenceSetProperty is reached then the result is **false** , **valid** becomes **false** and **after** becomes **true** .

Defined in: OMWeakReferenceSetIterT.h

Return Value

false if this OMWeakReferenceSetIterator has passed the last *ReferencedObject*, **true** otherwise.

Class Template Arguments

ReferencedObject
The type of the contained objects.

Back to OMWeakReferenceSetIterator

OMWeakReferenceSetIterator::operator--

template <class *ReferencedObject***>**
bool OMWeakReferenceSetIterator<*ReferencedObject***>::operator--(void)**

Retreat this OMWeakReferenceSetIterator to the previous *ReferencedObject*, if any. If the beginning of the associated OMWeakReferenceSetProperty is not reached then the result is **true** , **valid** becomes **true** and **before** becomes **false** . If the beginning of the associated OMWeakReferenceSetProperty is reached then the result is **false** , **valid** becomes **false** and **before** becomes **true** .

Defined in: OMWeakReferenceSetIterT.h

Return Value

false if this OMWeakReferenceSetIterator has passed the first *ReferencedObject*, **true** otherwise.

Class Template Arguments

ReferencedObject
The type of the contained objects.

Back to OMWeakReferenceSetIterator

OMWeakReferenceSetIterator::reset

template <class *ReferencedObject***>**
void OMWeakReferenceSetIterator<*ReferencedObject***>::reset(OMIteratorPosition** *initialPosition***)**

Reset this OMWeakReferenceSetIterator to the given *initialPosition*. If *initialPosition* is specified as OMBefore then this OMWeakReferenceSetIterator is made ready to traverse the associated OMWeakReferenceSetProperty in the forward direction (increasing *Keys*). If *initialPosition* is specified as OMAfter then this OMWeakReferenceSetIterator is made ready to traverse the associated OMWeakReferenceSetProperty in the reverse direction (decreasing *Keys*).

Defined in: OMWeakReferenceSetIterT.h

Parameters

initialPosition
The position to which this OMWeakReferenceSetIterator should be reset.

Class Template Arguments

ReferencedObject
The type of the contained objects.

Back to OMWeakReferenceSetIterator

OMWeakReferenceSetIterator::setValue

template <class *ReferencedObject***>**
ReferencedObject* OMWeakReferenceSetIterator<*ReferencedObject***>::setValue(const ReferencedObject*** *newObject***)**

Set the *ReferencedObject* in the associated OMWeakReferenceSetProperty at the position currently designated by this OMWeakReferenceSetIterator to *newObject*. The previous *ReferencedObject*, if any, is returned. To preserve the ordering of *Keys*, the *Key* of *newObject* must be the same as that of the existing *ReferencedObject*.

Defined in: OMWeakReferenceSetIterT.h

Return Value

The previous *ReferencedObject* if any, otherwise 0.

Parameters

newObject
The new *ReferencedObject*.

Class Template Arguments

ReferencedObject
The type of the contained objects.

Back to OMWeakReferenceSetIterator

OMWeakReferenceSetIterator::valid

template <class *ReferencedObject***>**
bool OMWeakReferenceSetIterator<*ReferencedObject***>::valid(void) const**

Is this OMWeakReferenceSetIterator validly positioned on a *ReferencedObject* ?

Defined in: OMWeakReferenceSetIterT.h

Return Value

true if this OMWeakReferenceSetIterator is positioned on a *ReferencedObject*, **false** otherwise.

Class Template Arguments

ReferencedObject
The type of the contained objects.

Back to OMWeakReferenceSetIterator

OMWeakReferenceSetIterator::value

template <class *ReferencedObject***>**
ReferencedObject* OMWeakReferenceSetIterator<*ReferencedObject***>::value(void) const**

Return the *ReferencedObject* in the associated OMWeakReferenceSetProperty at the position currently designated by this OMWeakReferenceSetIterator.

Defined in: OMWeakReferenceSetIterT.h

Return Value

The *ReferencedObject* at the current position.

Class Template Arguments

ReferencedObject
The type of the contained objects.

Back to OMWeakReferenceSetIterator

OMWeakReferenceSetIterator::~OMWeakReferenceSetIterator

template <class *ReferencedObject***>**
OMWeakReferenceSetIterator<*ReferencedObject***>::~OMWeakReferenceSetIterator(void)**

Destroy this OMWeakReferenceSetIterator.

Defined in: OMWeakReferenceSetIterT.h

Class Template Arguments

ReferencedObject
The type of the contained objects.

Back to OMWeakReferenceSetIterator

OMWeakReferenceSetProperty class

OMWeakReferenceSetProperty **class OMWeakReferenceSetProperty: public OMWeakReferenceSet**

Persistent sets of uniquely identified weakly referenced (non-contained) objects supported by the Object Manager. Objects are accessible by unique identifier (the key). The objects are not ordered. Duplicates objects are not allowed.

Defined in: OMWeakRefSetProperty.h

Class Template Arguments

ReferencedObject
The type of the referenced (contained) object. This type must be a descendant of OMStorable and of **OMUnique** .

Author

Tim Bingham - tjb - (Avid Technology, Inc.)

Class Members

Public members.
OMWeakReferenceSetProperty**(const OMPropertyId propertyId, const wchar_t* name, const wchar_t* targetName, const OMPropertyId keyPropertyId)**
Constructor.
OMWeakReferenceSetProperty**(const OMPropertyId propertyId, const wchar_t* name, const OMPropertyId keyPropertyId, const OMPropertyId* targetPropertyPath)**
Constructor.
virtual ~OMWeakReferenceSetProperty**(void)**
Destructor.
virtual void save**(void) const**

Save this **OMWeakReferenceSetProperty.**

virtual void close**(void)**

Close this **OMWeakReferenceSetProperty.**

virtual void detach**(void)**

Detach this **OMWeakReferenceSetProperty.**

virtual void restore**(size_t externalSize)**

Restore this **OMWeakReferenceSetProperty**, the external (persisted) size of the **OMWeakReferenceSetProperty** is *externalSize*.

size_t count**(void) const**

The number of *ReferencedObject*s in this **OMWeakReferenceSetProperty.**

void insert**(const ReferencedObject* object)**

Insert *object* into this **OMWeakReferenceSetProperty.**

bool ensurePresent**(const ReferencedObject* object)**

If it is not already present, insert *object* into this **OMWeakReferenceSetProperty** and return true, otherwise return false.

void appendValue**(const ReferencedObject* object)**

Append the given *ReferencedObject object* to this **OMWeakReferenceSetProperty.**

ReferencedObject* remove**(const OMUniqueObjectIdentification& identification)**

Remove the *ReferencedObject* identified by *identification* from this **OMWeakReferenceSetProperty.**

bool ensureAbsent**(const OMUniqueObjectIdentification& identification)**

If it is present, remove the *ReferencedObject* identified by *identification* from this **OMWeakReferenceSetProperty** and return true, otherwise return false.

void removeValue**(const ReferencedObject* object)**

Remove *object* from this **OMWeakReferenceSetProperty.**

bool ensureAbsent**(const ReferencedObject* object)**

If it is present, remove *object* from this **OMWeakReferenceSetProperty** and return true, otherwise return false.

bool containsValue**(const ReferencedObject* object) const**

Does this **OMWeakReferenceSetProperty** contain *object* ?

virtual bool contains**(const OMUniqueObjectIdentification& identification) const**

Does this **OMWeakReferenceSetProperty** contain a *ReferencedObject* identified by *identification*?

ReferencedObject* value**(const OMUniqueObjectIdentification& identification) const**

The *ReferencedObject* in this **OMWeakReferenceSetProperty** identified by *identification*.

virtual bool find**(const OMUniqueObjectIdentification& identification, ReferencedObject*& object) const**

Find the *ReferencedObject* in this **OMWeakReferenceSetProperty** identified by *identification*. If the object is found it is returned in *object* and the result is true. If the element is not found the result is false.

virtual bool isVoid**(void) const**

Is this **OMWeakReferenceSetProperty** void ?

virtual void removeProperty**(void)**

Remove this optional **OMWeakReferenceSetProperty.**

virtual size_t bitsSize**(void) const**

The size of the raw bits of this **OMWeakReferenceSetProperty.** The size is given in bytes.

virtual void getBits**(OMByte* bits, size_t size) const**

Get the raw bits of this **OMWeakReferenceSetProperty.** The raw bits are copied to the buffer at address *bits* which is *size* bytes in size.

virtual void setBits**(const OMByte* bits, size_t size)**

Set the raw bits of this **OMWeakReferenceSetProperty.** The raw bits are copied from the buffer at address *bits* which is *size* bytes in size.

virtual void insertObject**(const OMObject* object)**

Insert *object* into this **OMWeakReferenceSetProperty.**

virtual bool containsObject**(const OMObject* object) const**

Does this **OMWeakReferenceSetProperty** contain *object* ?

virtual void removeObject**(const OMObject* object)**

Remove *object* from this **OMWeakReferenceSetProperty.**

virtual void removeAllObjects**(void)**

Remove all objects from this **OMWeakReferenceSetProperty.**

virtual OMReferenceContainerIterator* createIterator**(void) const**

Create an OMReferenceContainerIterator over this **OMWeakReferenceSetProperty.**

virtual OMObject* remove**(void* identification)**

Remove the OMObject identified by *identification* from this **OMWeakReferenceSetProperty.**

virtual bool contains**(void* identification) const**

Does this **OMWeakReferenceSetProperty** contain an OMObject identified by *identification* ?

virtual bool findObject**(void* identification, OMObject*& object) const**

Find the OMObject in this **OMWeakReferenceSetProperty** identified by *identification*. If the object is found it is returned in *object* and the result is **true** . If the object is not found the result is **false** .

virtual OMStrongReferenceSet* targetSet(void) const

The OMStrongReferenceSet in which the objects referenced by this **OMWeakReferenceSetProperty** must reside.

OMWeakReferenceSetProperty::

OMWeakReferenceSetProperty::(*OMWeakReferenceSetProperty*, **const OMPropertyId** *propertyId*, **const wchar_t*** *name*, **const wchar_t*** *targetName***)**

Constructor.

Defined in: OMWeakRefSetPropertyT.h

Parameters

OMWeakReferenceSetProperty

The property id.

propertyId

The name of this OMWeakReferenceSetProperty.

name

The name (as a string) of the the OMProperty instance (a set property) in which the objects referenced by the elements of this OMWeakReferenceSetProperty reside.

targetName

The id of the property by which the *ReferencedObject*s are uniquely identified (the key).

Back to OMWeakReferenceSetProperty

OMWeakReferenceSetProperty::appendValue

template <class *ReferencedObject***>**

void OMWeakReferenceSetProperty<_ReferencedObject_**>::appendValue(const ReferencedObject*** *object***)**

Append the given *ReferencedObject object* to this OMWeakReferenceSetProperty.

Defined in: OMWeakRefSetPropertyT.h

Parameters

object
: A pointer to a *ReferencedObject*.

Class Template Arguments

ReferencedObject
: The type of the referenced (contained) object. This type must be a descendant of OMStorable.

Back to OMWeakReferenceSetProperty

OMWeakReferenceSetProperty::bitsSize

template <class *ReferencedObject***>**
size_t OMWeakReferenceSetProperty<*ReferencedObject***>::bitsSize(void) const**

The size of the raw bits of this OMWeakReferenceSetProperty. The size is given in bytes.

Defined in: OMWeakRefSetPropertyT.h

Return Value

The size of the raw bits of this OMWeakReferenceSetProperty in bytes.

Class Template Arguments

ReferencedObject
: The type of the referenced (contained) object. This type must be a descendant of OMStorable and **OMUnique** .

Back to OMWeakReferenceSetProperty

OMWeakReferenceSetProperty::close

template <class *ReferencedObject***>**
void OMWeakReferenceSetProperty<*ReferencedObject***>::close(void)**

Close this OMWeakReferenceSetProperty.

Defined in: OMWeakRefSetPropertyT.h

Class Template Arguments

ReferencedObject
The type of the referenced (contained) object. This type must be a descendant of OMStorable and **OMUnique** .

Back to OMWeakReferenceSetProperty

OMWeakReferenceSetProperty::contains

template <class *ReferencedObject***>**
bool OMWeakReferenceSetProperty<*ReferencedObject***>::contains(const OMUniqueObjectIdentification&** *identification***)**

Does this OMWeakReferenceSetProperty contain a *ReferencedObject* identified by *identification*?

Defined in: OMWeakRefSetPropertyT.h

Return Value

True if the object is found, false otherwise.

Parameters

identification
The unique identification of the desired object, the search key.

Class Template Arguments

ReferencedObject
The type of the referenced (contained) object. This type must be a descendant of OMStorable.

Back to OMWeakReferenceSetProperty

OMWeakReferenceSetProperty::contains

template <class *ReferencedObject***>**
bool OMWeakReferenceSetProperty<*ReferencedObject***>::contains(void*** *identification***) const**

Does this OMWeakReferenceSetProperty contain an OMObject identified by *identification* ?

Defined in: OMWeakRefSetPropertyT.h

Return Value

True if the object was found, false otherwise.

Parameters

identification
The unique identification of the object for which to search.

Class Template Arguments

ReferencedObject
The type of the referenced (contained) object. This type must be a descendant of OMStorable and **OMUnique** .

Back to OMWeakReferenceSetProperty

OMWeakReferenceSetProperty::containsObject

template <class *ReferencedObject***>**
bool OMWeakReferenceSetProperty<*ReferencedObject***>::containsObject(const OMObject*** *object***) const**

Does this OMWeakReferenceSetProperty contain *object* ?

Defined in: OMWeakRefSetPropertyT.h

Return Value

True if *object* is present, false otherwise.

Parameters

object
The OMObject for which to search.

Class Template Arguments

ReferencedObject
The type of the referenced (contained) object. This type must be a descendant of OMStorable and **OMUnique** .

Back to OMWeakReferenceSetProperty

OMWeakReferenceSetProperty::containsValue

template <class *ReferencedObject***>**
bool OMWeakReferenceSetProperty<*ReferencedObject***>::containsValue(const ReferencedObject*** *object***)**

Does this OMWeakReferenceSetProperty contain *object* ?

Defined in: OMWeakRefSetPropertyT.h

Parameters

object
: A pointer to a *ReferencedObject*.

Class Template Arguments

ReferencedObject
: The type of the referenced (contained) object. This type must be a descendant of OMStorable.

Back to OMWeakReferenceSetProperty

OMWeakReferenceSetProperty::count

size_t OMWeakReferenceSetProperty::count(void) const

The number of *ReferencedObject*s in this OMWeakReferenceSetProperty.

Defined in: OMWeakRefSetPropertyT.h

Back to OMWeakReferenceSetProperty

OMWeakReferenceSetProperty::createIterator

template <class *ReferencedObject***>**
OMReferenceContainerIterator* OMWeakReferenceSetProperty<*ReferencedObject***>::createIterator(void) const**

Create an OMReferenceContainerIterator over this OMWeakReferenceSetProperty.

Defined in: OMWeakRefSetPropertyT.h

Return Value

An OMReferenceContainerIterator over this OMWeakReferenceSetProperty.

Class Template Arguments

ReferencedObject
: The type of the referenced (contained) object. This type must be a descendant of OMStorable and **OMUnique** .

Back to OMWeakReferenceSetProperty

OMWeakReferenceSetProperty::detach

template <class *ReferencedObject***>**
void OMWeakReferenceSetProperty<*ReferencedObject***>::detach(void)**

Detach this OMWeakReferenceSetProperty.

Defined in: OMWeakRefSetPropertyT.h

Class Template Arguments

ReferencedObject
The type of the referenced (contained) object. This type must be a descendant of OMStorable and **OMUnique** .

Back to OMWeakReferenceSetProperty

OMWeakReferenceSetProperty::ensureAbsent

template <class *ReferencedObject***>**
bool OMWeakReferenceSetProperty<*ReferencedObject***>::ensureAbsent(const ReferencedObject*** *object***)**

If it is present, remove *object* from this OMWeakReferenceSetProperty and return true, otherwise return false.

Defined in: OMWeakRefSetPropertyT.h

Return Value

True if the object was removed, false if it was already absent.

Parameters

object
The object to remove.

Class Template Arguments

ReferencedObject
The type of the referenced (contained) object. This type must be a descendant of OMStorable.

Back to OMWeakReferenceSetProperty

OMWeakReferenceSetProperty::ensureAbsent

template <class *ReferencedObject***>**
bool OMWeakReferenceSetProperty<*ReferencedObject***>::ensureAbsent(const OMUniqueObjectIdentification&** *identification***)**

If it is present, remove the *ReferencedObject* identified by *identification* from this OMWeakReferenceSetProperty and return true, otherwise return false.

Defined in: OMWeakRefSetPropertyT.h

Return Value

True if the object was removed, false if it was already absent.

Parameters

identification
The object to remove.

Class Template Arguments

ReferencedObject
The type of the referenced (contained) object. This type must be a descendant of OMStorable.

Back to OMWeakReferenceSetProperty

OMWeakReferenceSetProperty::ensurePresent

template <class *ReferencedObject***>**
bool OMWeakReferenceSetProperty<*ReferencedObject***>::ensurePresent(const ReferencedObject*** *object***)**

If it is not already present, insert *object* into this OMWeakReferenceSetProperty and return true, otherwise return false.

Defined in: OMWeakRefSetPropertyT.h

Return Value

True if the object was inserted, false if it was already present.

Parameters

object
The object to insert.

Class Template Arguments

ReferencedObject
The type of the referenced (contained) object. This type must be a descendant of OMStorable.

Back to OMWeakReferenceSetProperty

OMWeakReferenceSetProperty::find

template <class *ReferencedObject***>**
bool OMWeakReferenceSetProperty<*ReferencedObject***>::find(const OMUniqueObjectIdentification&** *identification***, ReferencedObject*&** *object***) const**

Find the *ReferencedObject* in this OMWeakReferenceSetProperty identified by *identification*. If the object is found it is returned in *object* and the result is true. If the element is not found the result is false.

Defined in: OMWeakRefSetPropertyT.h

Return Value

True if the object is found, false otherwise.

Parameters

identification
: The unique identification of the desired object, the search key.

object
: A pointer to a *ReferencedObject* by reference.

Class Template Arguments

ReferencedObject
: The type of the referenced (contained) object. This type must be a descendant of OMStorable.

Back to OMWeakReferenceSetProperty

OMWeakReferenceSetProperty::findObject

template <class *ReferencedObject***>**
bool OMWeakReferenceSetProperty<*ReferencedObject***>::findObject(void*** *identification*, **OMObject*&** *object***) const**

Find the OMObject in this OMWeakReferenceSetProperty identified by *identification*. If the object is found it is returned in *object* and the result is **true** . If the object is not found the result is **false** .

Defined in: OMWeakRefSetPropertyT.h

Return Value

True if the object was found, false otherwise.

Parameters

identification
: The unique identification of the object for which to search.

object
: The object.

Class Template Arguments

ReferencedObject

The type of the referenced (contained) object. This type must be a descendant of OMStorable and **OMUnique** .

Back to OMWeakReferenceSetProperty

OMWeakReferenceSetProperty::getBits

template <class *ReferencedObject***>**
void OMWeakReferenceSetProperty<*ReferencedObject***>::getBits(OMByte*** *bits***, size_t** *ANAME***) const**

Get the raw bits of this OMWeakReferenceSetProperty. The raw bits are copied to the buffer at address *bits* which is *size* bytes in size.

Defined in: OMWeakRefSetPropertyT.h

Parameters

bits
The address of the buffer into which the raw bits are copied.

ANAME
The size of the buffer.

Class Template Arguments

ReferencedObject
The type of the referenced (contained) object. This type must be a descendant of OMStorable and **OMUnique** .

Back to OMWeakReferenceSetProperty

OMWeakReferenceSetProperty::insert

template <class *ReferencedObject***>**
void OMWeakReferenceSetProperty<*ReferencedObject***>::insert(const ReferencedObject*** *object***)**

Insert *object* into this OMWeakReferenceSetProperty.

Defined in: OMWeakRefSetPropertyT.h

Parameters

object
The object to insert.

Class Template Arguments

ReferencedObject

The type of the referenced (contained) object. This type must be a descendant of OMStorable.

Back to OMWeakReferenceSetProperty

OMWeakReferenceSetProperty::insertObject

template <class *ReferencedObject***>**
void OMWeakReferenceSetProperty<*ReferencedObject***>::insertObject(const OMObject*** *object***)**

Insert *object* into this OMWeakReferenceSetProperty.

Defined in: OMWeakRefSetPropertyT.h

Parameters

object
The OMObject to insert.

Class Template Arguments

ReferencedObject
The type of the referenced (contained) object. This type must be a descendant of OMStorable and **OMUnique** .

Back to OMWeakReferenceSetProperty

OMWeakReferenceSetProperty::isVoid

template <class *ReferencedObject***>**
bool OMWeakReferenceSetProperty<*ReferencedObject***>::isVoid(void) const**

Is this OMWeakReferenceSetProperty void ?

Defined in: OMWeakRefSetPropertyT.h

Return Value

True if this OMWeakReferenceSetProperty is void, false otherwise.

Class Template Arguments

ReferencedObject
The type of the referenced (contained) object. This type must be a descendant of OMStorable and **OMUnique** .

Back to OMWeakReferenceSetProperty

OMWeakReferenceSetProperty::OMWeakReferenceSetProperty

OMWeakReferenceSetProperty::OMWeakReferenceSetProperty(const OMPropertyId *propertyId*, **const wchar_t*** *name*, **const OMPropertyId** *keyPropertyId*, **const OMPropertyId*** *targetPropertyPath***)**

Constructor.

Defined in: OMWeakRefSetPropertyT.h

Parameters

propertyId
: The property id.

name
: The name of this OMWeakReferenceSetProperty.

keyPropertyId
: The id of the property by which the *ReferencedObject*s are uniquely identified (the key).

targetPropertyPath
: The name (as a list of pids) of the the OMProperty instance (a set property) in which the objects referenced by the elements of this OMWeakReferenceSetProperty reside.

Back to OMWeakReferenceSetProperty

OMWeakReferenceSetProperty::remove

template <class *ReferencedObject***>**
ReferencedObject* OMWeakReferenceSetProperty<*ReferencedObject***>::remove(const OMUniqueObjectIdentification&** *identification***)**

Remove the *ReferencedObject* identified by *identification* from this OMWeakReferenceSetProperty.

Defined in: OMWeakRefSetPropertyT.h

Return Value

A pointer to the removed *ReferencedObject*. If lazy loading is enabled and the referenced object was never loaded the value returned is 0.

Parameters

identification
: The unique identification of the object to be removed, the search key.

Class Template Arguments

ReferencedObject
: The type of the referenced (contained) object. This type must be a descendant of OMStorable.

Back to OMWeakReferenceSetProperty

OMWeakReferenceSetProperty::remove

template <class *ReferencedObject***>**
OMObject* OMWeakReferenceSetProperty<*ReferencedObject***>::remove(void*** *identification***)**

Remove the OMObject identified by *identification* from this OMWeakReferenceSetProperty.

Defined in: OMWeakRefSetPropertyT.h

Return Value

The object that was removed.

Parameters

identification
The unique identification of the object to remove.

Class Template Arguments

ReferencedObject
The type of the referenced (contained) object. This type must be a descendant of OMStorable and **OMUnique** .

Back to OMWeakReferenceSetProperty

OMWeakReferenceSetProperty::removeAllObjects

template <class *ReferencedObject***>**
void OMWeakReferenceSetProperty<*ReferencedObject***>::removeAllObjects(void)**

Remove all objects from this OMWeakReferenceSetProperty.

Defined in: OMWeakRefSetPropertyT.h

Class Template Arguments

ReferencedObject
The type of the referenced (contained) object. This type must be a descendant of OMStorable and **OMUnique** .

Back to OMWeakReferenceSetProperty

OMWeakReferenceSetProperty::removeObject

template <class *ReferencedObject***>**
void OMWeakReferenceSetProperty<*ReferencedObject***>::removeObject(const OMObject*** *object***)**

Remove *object* from this OMWeakReferenceSetProperty.

Defined in: OMWeakRefSetPropertyT.h

Parameters

object
The OMObject to remove.

Class Template Arguments

ReferencedObject
The type of the referenced (contained) object. This type must be a descendant of OMStorable and **OMUnique** .

Back to OMWeakReferenceSetProperty

OMWeakReferenceSetProperty::removeProperty

template <class *ReferencedObject***>**
void OMWeakReferenceSetProperty<*ReferencedObject***>::removeProperty(void)**

Remove this optional OMWeakReferenceSetProperty.

Defined in: OMWeakRefSetPropertyT.h

Class Template Arguments

ReferencedObject
The type of the referenced (contained) object. This type must be a descendant of OMStorable and **OMUnique** .

Back to OMWeakReferenceSetProperty

OMWeakReferenceSetProperty::removeValue

template <class *ReferencedObject***>**
void OMWeakReferenceSetProperty<*ReferencedObject***>::removeValue(const ReferencedObject*** *object***)**

Remove *object* from this OMWeakReferenceSetProperty.

Defined in: OMWeakRefSetPropertyT.h

Parameters

object
: A pointer to a *ReferencedObject*.

Class Template Arguments

ReferencedObject
: The type of the referenced (contained) object. This type must be a descendant of OMStorable.

Back to OMWeakReferenceSetProperty

OMWeakReferenceSetProperty::restore

template <class *ReferencedObject***>**
void OMWeakReferenceSetProperty<*ReferencedObject***>::restore(size_t** *externalSize***)**

Restore this OMWeakReferenceSetProperty, the external (persisted) size of the OMWeakReferenceSetProperty is *externalSize*.

Defined in: OMWeakRefSetPropertyT.h

Parameters

externalSize
: The external (persisted) size of the OMWeakReferenceSetProperty.

Class Template Arguments

ReferencedObject
: The type of the referenced (contained) object. This type must be a descendant of OMStorable and **OMUnique** .

Back to OMWeakReferenceSetProperty

OMWeakReferenceSetProperty::save

template <class *ReferencedObject***>**
void OMWeakReferenceSetProperty<*ReferencedObject***>::save(void) const**

Save this OMWeakReferenceSetProperty.

Defined in: OMWeakRefSetPropertyT.h

Class Template Arguments

ReferencedObject
: The type of the referenced (contained) object. This type must be a descendant of OMStorable and **OMUnique** .

Back to OMWeakReferenceSetProperty

OMWeakReferenceSetProperty::setBits

template <class *ReferencedObject***>**
void OMWeakReferenceSetProperty<*ReferencedObject***>::setBits(const OMByte*** *bits***, size_t** *size***)**

Set the raw bits of this OMWeakReferenceSetProperty. The raw bits are copied from the buffer at address *bits* which is *size* bytes in size.

Defined in: OMWeakRefSetPropertyT.h

Parameters

bits
: The address of the buffer from which the raw bits are copied.

size
: The size of the buffer.

Class Template Arguments

ReferencedObject
: The type of the referenced (contained) object. This type must be a descendant of OMStorable and **OMUnique** .

Back to OMWeakReferenceSetProperty

OMWeakReferenceSetProperty::value

template <class *ReferencedObject***>**
ReferencedObject* OMWeakReferenceSetProperty<*ReferencedObject***>::value(const OMUniqueObjectIdentification&** *identification***) const**

The *ReferencedObject* in this OMWeakReferenceSetProperty identified by *identification*.

Defined in: OMWeakRefSetPropertyT.h

Return Value

A pointer to the *ReferencedObject*.

Parameters

identification
: The unique identification of the desired object, the search key.

Class Template Arguments

ReferencedObject
The type of the referenced (contained) object. This type must be a descendant of OMStorable.

Back to OMWeakReferenceSetProperty

OMWeakReferenceSetProperty::~OMWeakReferenceSetProperty

OMWeakReferenceSetProperty::~OMWeakReferenceSetProperty(void)

Destructor.

Defined in: OMWeakRefSetPropertyT.h

Back to OMWeakReferenceSetProperty

OMWeakReferenceVector class

OMWeakReferenceVector **class OMWeakReferenceVector: public OMReferenceVectorProperty**

Persistent elastic sequential collections of uniquely identified weakly referenced (non-contained) objects supported by the Object Manager. Objects are accessible by index. The order of objects is determined externally. Duplicate objects are allowed.

Defined in: OMWeakReferenceVector.h

Author

Tim Bingham - tjb - (Avid Technology, Inc.)

Class Members

Public members.
OMWeakReferenceVector**(const OMPropertyId propertyId, const wchar_t* name)**
Constructor.
virtual ~OMWeakReferenceVector**(void)**
Destructor.
virtual void grow(const size_t capacity)
Increase the capacity of this **OMWeakReferenceVector** so that it can contain at least *capacity OMObject*s without having to be resized.

OMWeakReferenceVector::OMWeakReferenceVector

OMWeakReferenceVector::OMWeakReferenceVector(const OMPropertyId *propertyId*, **const wchar_t*** *name***)**

Constructor.

Defined in: OMWeakReferenceVector.cpp

Parameters

propertyId
The property id.
name
The name of this OMWeakReferenceVector.
Back to OMWeakReferenceVector

OMWeakReferenceVector::~OMWeakReferenceVector

OMWeakReferenceVector::~OMWeakReferenceVector(void)

Destructor.

Defined in: OMWeakReferenceVector.cpp

Back to OMWeakReferenceVector

OMWeakReferenceVectorElement class

OMWeakReferenceVectorElement **class OMWeakReferenceVectorElement: public OMContainerElement**

Elements of Object Manager reference vectors.

Defined in: OMContainerElement.h

Author

Tim Bingham - tjb - (Avid Technology, Inc.)

Class Members

Public members.
OMWeakReferenceVectorElement**(void)**
Constructor.
OMWeakReferenceVectorElement**(OMProperty* property, OMUniqueObjectIdentification identification, OMPropertyTag targetTag)**
Constructor.
OMWeakReferenceVectorElement**(const OMWeakReferenceVectorElement& rhs)**
Copy constructor.
~OMWeakReferenceVectorElement**(void)**
Destructor.
OMWeakReferenceVectorElement& operator=**(const OMWeakReferenceVectorElement& rhs)**
Assignment. This operator provides value semantics for OMVector. This operator does not provide assignment of object references.
bool operator==**(const OMWeakReferenceVectorElement& rhs) const**

Equality. This operator provides value semantics for OMVector. This operator does not provide equality of object references.

OMStorable* setValue**(const OMUniqueObjectIdentification& identification, const OMStorable* value)**

Set the value of this **OMWeakReferenceVectorElement.**

OMUniqueObjectIdentification identification**(void) const**

The unique key of this **OMWeakReferenceVectorElement.**

OMWeakReferenceVectorElement::identification

OMUniqueObjectIdentification OMWeakReferenceVectorElement::identification(void)

The unique key of this OMWeakReferenceVectorElement.

Defined in: OMContainerElement.cpp

Return Value

The unique key of this OMWeakReferenceVectorElement.

Back to OMWeakReferenceVectorElement

OMWeakReferenceVectorElement::OMWeakReferenceVectorElement

OMWeakReferenceVectorElement::OMWeakReferenceVectorElement(const OMWeakReferenceVectorElement& *rhs***)**

Copy constructor.

Defined in: OMContainerElement.cpp

Parameters

rhs

The OMWeakReferenceVectorElement to copy.

Back to OMWeakReferenceVectorElement

OMWeakReferenceVectorElement::OMWeakReferenceVectorElement

OMWeakReferenceVectorElement::OMWeakReferenceVectorElement(OMProperty* *property***, OMUniqueObjectIdentification** *identification***, OMPropertyTag** *targetTag***)**

Constructor.

Defined in: OMContainerElement.cpp

Parameters

property
The OMProperty (a set property) that contains this OMWeakReferenceVectorElement.
identification
The unique key of this OMWeakReferenceVectorElement.
targetTag
A tag identifying the OMStrongReferenceVectorProperty in which the target resides.
Back to OMWeakReferenceVectorElement

OMWeakReferenceVectorElement::OMWeakReferenceVectorElement

OMWeakReferenceVectorElement::OMWeakReferenceVectorElement(void)

Constructor.

Defined in: OMContainerElement.cpp

Back to OMWeakReferenceVectorElement

OMWeakReferenceVectorElement::operator=

OMWeakReferenceVectorElement& OMWeakReferenceVectorElement::operator=(const OMWeakReferenceVectorElement& *rhs*)

Assignment. This operator provides value semantics for OMVector. This operator does not provide assignment of object references.

Defined in: OMContainerElement.cpp

Return Value

The OMWeakReferenceVectorElement resulting from the assignment.

Parameters

rhs
The OMWeakReferenceVectorElement to be assigned.
Back to OMWeakReferenceVectorElement

OMWeakReferenceVectorElement::operator==

bool OMWeakReferenceVectorElement::operator==(const OMWeakReferenceVectorElement& *rhs*)

Equality. This operator provides value semantics for OMVector. This operator does not provide equality of object references.

Defined in: OMContainerElement.cpp

Return Value

True if the values are the same, false otherwise.

Parameters

rhs
The OMWeakReferenceVectorElement to be compared.

Back to OMWeakReferenceVectorElement

OMWeakReferenceVectorElement::setValue

OMStorable* OMWeakReferenceVectorElement::setValue(const OMUniqueObjectIdentification& *identification*, **const OMStorable*** *value***)**

Set the value of this OMWeakReferenceVectorElement.

Defined in: OMContainerElement.cpp

Return Value

A pointer to previous OMStorable, if any.

Parameters

identification
TBS

value
A pointer to the new OMStorable.

Back to OMWeakReferenceVectorElement

OMWeakReferenceVectorElement::~OMWeakReferenceVectorElement

OMWeakReferenceVectorElement::~OMWeakReferenceVectorElement(void)

Destructor.

Defined in: OMContainerElement.cpp

Back to OMWeakReferenceVectorElement

OMWeakReferenceVectorIterator class

OMWeakReferenceVectorIterator **class OMWeakReferenceVectorIterator: public OMReferenceContainerIterator**

Iterators over OMWeakReferenceVectorPropertys.

Defined in: OMWeakReferenceVectorIter.h

Class Template Arguments

ReferencedObject
The type of the contained objects.

Author

Tim Bingham - tjb - (Avid Technology, Inc.)

Class Members

Public members.

OMWeakReferenceVectorIterator**(const OMWeakReferenceVectorProperty&ltReferencedObject>& vector, OMIteratorPosition initialPosition = OMBefore)**
Create an **OMWeakReferenceVectorIterator** over the given OMWeakReferenceVectorProperty *vector* and initialize it to the given *initialPosition*. If *initialPosition* is specified as OMBefore then this **OMWeakReferenceVectorIterator** is made ready to traverse the associated OMWeakReferenceVectorProperty in the forward direction (increasing indexes). If *initialPosition* is specified as OMAfter then this **OMWeakReferenceVectorIterator** is made ready to traverse the associated OMWeakReferenceVectorProperty in the reverse direction (decreasing indexes).

virtual OMReferenceContainerIterator* copy**(void) const**
Create a copy of this **OMWeakReferenceVectorIterator**.

virtual ~OMWeakReferenceVectorIterator(void)
Destroy this **OMWeakReferenceVectorIterator**.

virtual void reset**(OMIteratorPosition initialPosition = OMBefore)**
Reset this **OMWeakReferenceVectorIterator** to the given *initialPosition*. If *initialPosition* is specified as OMBefore then this **OMWeakReferenceVectorIterator** is made ready to traverse the associated OMWeakReferenceVectorProperty in the forward direction (increasing indexes). If *initialPosition* is specified as OMAfter then this **OMWeakReferenceVectorIterator** is made ready to traverse the associated OMWeakReferenceVectorProperty in the reverse direction (decreasing indexes).

virtual bool before**(void) const**
Is this **OMWeakReferenceVectorIterator** positioned before the first *ReferencedObject* ?

virtual bool after**(void) const**
Is this **OMWeakReferenceVectorIterator** positioned after the last *ReferencedObject* ?

virtual bool valid**(void) const**
Is this **OMWeakReferenceVectorIterator** validly positioned on a *ReferencedObject* ?

virtual size_t count**(void) const**
The number of *ReferencedObject*s in the associated OMWeakReferenceVectorProperty.

virtual bool operator++**()**
Advance this **OMWeakReferenceVectorIterator** to the next *ReferencedObject*, if any. If the end of the associated OMWeakReferenceVectorProperty is not reached then the result is **true** , **valid** becomes **true** and **after** becomes **false** . If the end of the associated OMWeakReferenceVectorProperty is reached then the result is **false** , **valid** becomes **false** and **after** becomes **true** .

virtual bool operator--**()**

Retreat this **OMWeakReferenceVectorIterator** to the previous *ReferencedObject*, if any. If the beginning of the associated OMWeakReferenceVectorProperty is not reached then the result is **true** , **valid** becomes **true** and **before** becomes **false** . If the beginning of the associated OMWeakReferenceVectorProperty is reached then the result is **false** , **valid** becomes **false** and **before** becomes **true** .

virtual ReferencedObject* value**(void) const**

Return the *ReferencedObject* in the associated OMWeakReferenceVectorProperty at the position currently designated by this **OMWeakReferenceVectorIterator.**

virtual ReferencedObject* setValue**(const ReferencedObject* newObject)**

Set the *ReferencedObject* in the associated OMWeakReferenceVectorProperty at the position currently designated by this **OMWeakReferenceVectorIterator** to *newObject*. The previous *ReferencedObject*, if any, is returned.

virtual ReferencedObject* clearValue**(void)**

Set the *ReferencedObject* in the associated OMWeakReferenceVectorProperty at the position currently designated by this **OMWeakReferenceVectorIterator** to 0. The previous *ReferencedObject*, if any, is returned.

virtual size_t index**(void) const**

Return the index of the *ReferencedObject* in the associated OMWeakReferenceVectorProperty at the position currently designated by this **OMWeakReferenceVectorIterator.**

OMUniqueObjectIdentification identification**(void) const**

Return the *Key* of the *ReferencedObject* in the associated OMWeakReferenceVectorProperty at the position currently designated by this **OMWeakReferenceVectorIterator.**

virtual OMObject* currentObject**(void) const**

Return the *OMObject* in the associated reference container property at the position currently designated by this **OMWeakReferenceVectorIterator.**

OMWeakReferenceVectorIterator**(const VectorIterator& iter)**

Create an **OMWeakReferenceVectorIterator** given an underlying OMVectorIterator.

OMWeakReferenceVectorIterator::

template <class *ReferencedObject***>**
OMWeakReferenceVectorIterator<*ReferencedObject***>::(void)**

Destroy this OMWeakReferenceVectorIterator.

Defined in: OMWeakReferenceVectorIterT.h

Class Template Arguments

ReferencedObject

The type of the contained objects.

Back to OMWeakReferenceVectorIterator

OMWeakReferenceVectorIterator::after

template <class *ReferencedObject***>**
bool OMWeakReferenceVectorIterator<*ReferencedObject***>::after(void) const**

Is this OMWeakReferenceVectorIterator positioned after the last *ReferencedObject* ?

Defined in: OMWeakReferenceVectorIterT.h

Return Value

true if this OMWeakReferenceVectorIterator is positioned after the last *ReferencedObject*, **false** otherwise.

Class Template Arguments

ReferencedObject
The type of the contained objects.

Back to OMWeakReferenceVectorIterator

OMWeakReferenceVectorIterator::before

template <class *ReferencedObject***>**
bool OMWeakReferenceVectorIterator<*ReferencedObject***>::before(void) const**

Is this OMWeakReferenceVectorIterator positioned before the first *ReferencedObject* ?

Defined in: OMWeakReferenceVectorIterT.h

Return Value

true if this OMWeakReferenceVectorIterator is positioned before the first *ReferencedObject*, **false** otherwise.

Class Template Arguments

ReferencedObject
The type of the contained objects.

Back to OMWeakReferenceVectorIterator

OMWeakReferenceVectorIterator::clearValue

template <class *ReferencedObject***>**
ReferencedObject* OMWeakReferenceVectorIterator<*ReferencedObject***>::clearValue(void)**

Set the *ReferencedObject* in the associated OMWeakReferenceVectorProperty at the position currently designated by this OMWeakReferenceVectorIterator to 0. The previous *ReferencedObject*, if any, is returned.

Defined in: OMWeakReferenceVectorIterT.h

Return Value

The previous *ReferencedObject* if any, otherwise 0.

Class Template Arguments

ReferencedObject
The type of the contained objects.

Back to OMWeakReferenceVectorIterator

OMWeakReferenceVectorIterator::copy

template <class *ReferencedObject***>**
OMReferenceContainerIterator* OMWeakReferenceVectorIterator<*ReferencedObject***>::copy(void) const**

Create a copy of this OMWeakReferenceVectorIterator.

Defined in: OMWeakReferenceVectorIterT.h

Return Value

The new OMWeakReferenceVectorIterator.

Class Template Arguments

ReferencedObject
The type of the contained objects.

Back to OMWeakReferenceVectorIterator

OMWeakReferenceVectorIterator::count

template <class *ReferencedObject***>**
size_t OMWeakReferenceVectorIterator<*ReferencedObject***>::count(void) const**

The number of *ReferencedObject*s in the associated OMWeakReferenceVectorProperty.

Defined in: OMWeakReferenceVectorIterT.h

Return Value

The number of *ReferencedObject*s

Class Template Arguments

ReferencedObject
The type of the contained objects.

Back to OMWeakReferenceVectorIterator

OMWeakReferenceVectorIterator::currentObject

template <class *ReferencedObject***>**
OMObject* OMWeakReferenceVectorIterator<*ReferencedObject***>::currentObject(void) const**

Return the OMObject in the associated OMWeakReferenceVectorProperty at the position currently designated by this OMWeakReferenceVectorIterator.

Defined in: OMWeakReferenceVectorIterT.h

Return Value

The OMObject at the current position.

Class Template Arguments

ReferencedObject
The type of the contained objects.

Back to OMWeakReferenceVectorIterator

OMWeakReferenceVectorIterator::identification

template <class *ReferencedObject***>**
OMUniqueObjectIdentification OMWeakReferenceVectorIterator<*ReferencedObject***>::identification(void) const**

Return the *Key* of the *ReferencedObject* in the associated OMWeakReferenceVectorProperty at the position currently designated by this OMWeakReferenceVectorIterator.

Defined in: OMWeakReferenceVectorIterT.h

Return Value

The *Key* at the current position.

Class Template Arguments

ReferencedObject
The type of the contained objects.

Back to OMWeakReferenceVectorIterator

OMWeakReferenceVectorIterator::index

template <class *Element***>**
size_t OMWeakReferenceVectorIterator<*Element***>::index(void) const**

Return the index of the *ReferencedObject* in the associated OMWeakReferenceVectorProperty at the position currently designated by this OMWeakReferenceVectorIterator.

Defined in: OMWeakReferenceVectorIterT.h

Return Value

The index of the current position.

Class Template Arguments

Element
The type of the contained elements.

Back to OMWeakReferenceVectorIterator

OMWeakReferenceVectorIterator::OMWeakReferenceVectorIterator

template <class *ReferencedObject***>**
OMWeakReferenceVectorIterator<*ReferencedObject***>::OMWeakReferenceVectorIterator(const VectorIterator&** *iter***)**

Create an OMWeakReferenceVectorIterator given an underlying OMVectorIterator.

Defined in: OMWeakReferenceVectorIterT.h

Parameters

iter
The underlying OMVectorIterator.

Class Template Arguments

ReferencedObject
The type of the contained objects.

Back to OMWeakReferenceVectorIterator

OMWeakReferenceVectorIterator::OMWeakReferenceVectorIterator

template <class *ReferencedObject***>**
OMWeakReferenceVectorIterator<*ReferencedObject***>::OMWeakReferenceVectorIterator(const OMWeakReferenceVectorProperty&ltReferencedObject>&** *vector***, OMIteratorPosition** *initialPosition***)**

Create an OMWeakReferenceVectorIterator over the given OMWeakReferenceVectorProperty *vector* and initialize it to the given *initialPosition*. If *initialPosition* is specified as OMBefore then this OMWeakReferenceVectorIterator is made ready to traverse the associated OMWeakReferenceVectorProperty in the forward direction (increasing indexes). If *initialPosition* is specified as OMAfter then this OMWeakReferenceVectorIterator is made ready to traverse the associated OMWeakReferenceVectorProperty in the reverse direction (decreasing indexes).

Defined in: OMWeakReferenceVectorIterT.h

Parameters

vector
The OMWeakReferenceVector over which to iterate.
initialPosition
The initial position for this OMWeakReferenceVectorIterator.

Class Template Arguments

ReferencedObject
The type of the contained objects.
Back to OMWeakReferenceVectorIterator

OMWeakReferenceVectorIterator::operator++

template <class *ReferencedObject***>**
bool OMWeakReferenceVectorIterator<*ReferencedObject***>::operator++(void)**

Advance this OMWeakReferenceVectorIterator to the next *ReferencedObject*, if any. If the end of the associated OMWeakReferenceVectorProperty is not reached then the result is **true** , **valid** becomes **true** and **after** becomes **false** . If the end of the associated OMWeakReferenceVectorProperty is reached then the result is **false** , **valid** becomes **false** and **after** becomes **true** .

Defined in: OMWeakReferenceVectorIterT.h

Return Value

false if this OMWeakReferenceVectorIterator has passed the last *ReferencedObject*, **true** otherwise.

Class Template Arguments

ReferencedObject
The type of the contained objects.
Back to OMWeakReferenceVectorIterator

OMWeakReferenceVectorIterator::operator--

template <class *ReferencedObject***>**
bool OMWeakReferenceVectorIterator<*ReferencedObject***>::operator--(void)**

Retreat this OMWeakReferenceVectorIterator to the previous *ReferencedObject*, if any. If the beginning of the associated OMWeakReferenceVectorProperty is not reached then the result is **true** , **valid** becomes **true** and **before** becomes **false** . If the beginning of the associated OMWeakReferenceVectorProperty is reached then the result is **false** , **valid** becomes **false** and **before** becomes **true** .

Defined in: OMWeakReferenceVectorIterT.h

Return Value

false if this OMWeakReferenceVectorIterator has passed the first *ReferencedObject*, **true** otherwise.

Class Template Arguments

ReferencedObject
The type of the contained objects.

Back to OMWeakReferenceVectorIterator

OMWeakReferenceVectorIterator::reset

template <class *ReferencedObject***>**
void OMWeakReferenceVectorIterator<*ReferencedObject***>::reset(OMIteratorPosition** *initialPosition***)**

Reset this OMWeakReferenceVectorIterator to the given *initialPosition*. If *initialPosition* is specified as OMBefore then this OMWeakReferenceVectorIterator is made ready to traverse the associated OMWeakReferenceVectorProperty in the forward direction (increasing indexes). If *initialPosition* is specified as OMAfter then this OMWeakReferenceVectorIterator is made ready to traverse the associated OMWeakReferenceVectorProperty in the reverse direction (decreasing indexes).

Defined in: OMWeakReferenceVectorIterT.h

Parameters

initialPosition
The position to which this OMWeakReferenceVectorIterator should be reset.

Class Template Arguments

ReferencedObject
The type of the contained objects.

Back to OMWeakReferenceVectorIterator

OMWeakReferenceVectorIterator::setValue

template <class *ReferencedObject***>**
ReferencedObject* OMWeakReferenceVectorIterator<*ReferencedObject***>::setValue(const ReferencedObject*** *newObject***)**

Set the *ReferencedObject* in the associated OMWeakReferenceVectorProperty at the position currently designated by this OMWeakReferenceVectorIterator to *newObject*. The previous *ReferencedObject*, if any, is returned.

Defined in: OMWeakReferenceVectorIterT.h

Return Value

The previous *ReferencedObject* if any, otherwise 0.

Parameters

newObject
The new *ReferencedObject*.

Class Template Arguments

ReferencedObject
The type of the contained objects.

Back to OMWeakReferenceVectorIterator

OMWeakReferenceVectorIterator::valid

template <class *ReferencedObject***>**
bool OMWeakReferenceVectorIterator<*ReferencedObject***>::valid(void) const**

Is this OMWeakReferenceVectorIterator validly positioned on a *ReferencedObject* ?

Defined in: OMWeakReferenceVectorIterT.h

Return Value

true if this OMWeakReferenceVectorIterator is positioned on a *ReferencedObject*, **false** otherwise.

Class Template Arguments

ReferencedObject
The type of the contained objects.

Back to OMWeakReferenceVectorIterator

OMWeakReferenceVectorIterator::value

template <class *ReferencedObject***>**
ReferencedObject* OMWeakReferenceVectorIterator<*ReferencedObject***>::value(void) const**

Return the *ReferencedObject* in the associated OMWeakReferenceVectorProperty at the position currently designated by this OMWeakReferenceVectorIterator.

Defined in: OMWeakReferenceVectorIterT.h

Return Value

The *ReferencedObject* at the current position.

Class Template Arguments

ReferencedObject
The type of the contained objects.

Back to OMWeakReferenceVectorIterator

OMWeakReferenceVectorProperty class

OMWeakReferenceVectorProperty **class OMWeakReferenceVectorProperty: public OMWeakReferenceVector**

Persistent elastic sequential collections of uniquely identified weakly referenced (non-contained) objects supported by the Object Manager. Objects are accessible by index. The order of objects is determined externally. Duplicate objects are allowed.

Defined in: OMWeakRefVectorProperty.h

Class Template Arguments

ReferencedObject
The type of the referenced (contained) object. This type must be a descendant of OMStorable.

Author

Tim Bingham - tjb - (Avid Technology, Inc.)

Class Members

Public members.

OMWeakReferenceVectorProperty**(const OMPropertyId propertyId, const wchar_t* name, const wchar_t* targetName, const OMPropertyId keyPropertyId)**
Constructor.

OMWeakReferenceVectorProperty**(const OMPropertyId propertyId, const wchar_t* name, const OMPropertyId keyPropertyId, const OMPropertyId* targetPropertyPath)**
Constructor.

virtual ~OMWeakReferenceVectorProperty(void)
Destructor.

virtual void save**(void) const**
Save this **OMWeakReferenceVectorProperty.**

virtual void close**(void)**
Close this **OMWeakReferenceVectorProperty.**

virtual void detach**(void)**
Detach this **OMWeakReferenceVectorProperty.**

virtual void restore**(size_t externalSize)**
Restore this **OMWeakReferenceVectorProperty**, the external (persisted) size of the **OMWeakReferenceVectorProperty** is *externalSize*.

size_t count**(void) const**
The number of *ReferencedObject*s in this **OMWeakReferenceVectorProperty.**

ReferencedObject* setValueAt**(const ReferencedObject* object, const size_t index)**
Set the value of this **OMWeakReferenceVectorProperty** at position *index* to *object*.

ReferencedObject* clearValueAt**(const size_t index)**

Set the value of this **OMWeakReferenceVectorProperty** at position *index* to 0.

ReferencedObject* valueAt**(const size_t index) const**

The value of this **OMWeakReferenceVectorProperty** at position *index*.

void getValueAt**(ReferencedObject*& object, const size_t index) const**

Get the value of this **OMWeakReferenceVectorProperty** at position *index* into *object*.

bool find**(const size_t index, ReferencedObject*& object) const**

If *index* is valid, get the value of this **OMWeakReferenceVectorProperty** at position *index* into *object* and return true, otherwise return false.

void appendValue**(const ReferencedObject* object)**

Append the given *ReferencedObject object* to this **OMWeakReferenceVectorProperty.**

void prependValue**(const ReferencedObject* object)**

Prepend the given *ReferencedObject object* to this **OMWeakReferenceVectorProperty.**

void insert**(const ReferencedObject* object)**

Insert *object* into this **OMWeakReferenceVectorProperty**. This function is redefined from OMContainerProperty as **appendValue.**

void insertAt**(const ReferencedObject* object, const size_t index)**

Insert *object* into this **OMWeakReferenceVectorProperty** at position *index*. Existing objects at *index* and higher are shifted up one index position.

bool containsValue**(const ReferencedObject* object) const**

Does this **OMWeakReferenceVectorProperty** contain *object* ?

void removeValue**(const ReferencedObject* object)**

Remove *object* from this **OMWeakReferenceVectorProperty.**

ReferencedObject* removeAt**(const size_t index)**

Remove the object from this **OMWeakReferenceVectorProperty** at position *index*. Existing objects in this **OMWeakReferenceVectorProperty** at *index* + 1 and higher are shifted down one index position.

ReferencedObject* removeLast**(void)**

Remove the last (index == count() - 1) object from this **OMWeakReferenceVectorProperty.**

ReferencedObject* removeFirst**(void)**

Remove the first (index == 0) object from this **OMWeakReferenceVectorProperty.** Existing objects in this **OMWeakReferenceVectorProperty** are shifted down one index position.

size_t indexOfValue**(const ReferencedObject* object) const**

The index of the *ReferencedObject** *object*.

size_t countOfValue**(const ReferencedObject* object) const**

The number of occurrences of *object* in this **OMWeakReferenceVectorProperty.**

bool containsIndex**(const size_t index) const**

Does this **OMWeakReferenceVectorProperty** contain *index* ? Is *index* valid ?

bool findIndex**(const ReferencedObject* object, size_t& index) const**

If this OMWeakReferenceProperty contains *object* then place its index in *index* and return true, otherwise return false.

void grow**(const size_t capacity)**

Increase the capacity of this **OMWeakReferenceVectorProperty** so that it can contain at least *capacity ReferencedObject*s without having to be resized.

virtual bool isVoid**(void) const**

Is this **OMWeakReferenceVectorProperty** void ?

virtual void removeProperty**(void)**

Remove this optional **OMWeakReferenceVectorProperty.**

virtual size_t bitsSize**(void) const**

The size of the raw bits of this **OMWeakReferenceVectorProperty.** The size is given in bytes.

virtual void getBits**(OMByte* bits, size_t size) const**

Get the raw bits of this **OMWeakReferenceVectorProperty**. The raw bits are copied to the buffer at address *bits* which is *size* bytes in size.

virtual void setBits**(const OMByte* bits, size_t size)**

Set the raw bits of this **OMWeakReferenceVectorProperty**. The raw bits are copied from the buffer at address *bits* which is *size* bytes in size.

virtual void insertObject**(const OMObject* object)**

Insert *object* into this **OMWeakReferenceVectorProperty.**

virtual bool containsObject**(const OMObject* object) const**

Does this **OMWeakReferenceVectorProperty** contain *object* ?

virtual void removeObject**(const OMObject* object)**

Remove *object* from this **OMWeakReferenceVectorProperty.**

virtual void removeAllObjects**(void)**

Remove all objects from this **OMWeakReferenceVectorProperty.**

virtual OMReferenceContainerIterator* createIterator**(void) const**

Create an OMReferenceContainerIterator over this **OMWeakReferenceVectorProperty.**

virtual OMObject* setObjectAt**(const OMObject* object, const size_t index)**

Set the value of this **OMWeakReferenceVectorProperty** at position *index* to *object*.

virtual OMObject* getObjectAt**(const size_t index) const**

The value of this **OMWeakReferenceVectorProperty** at position *index*.

virtual void appendObject**(const OMObject* object)**

Append the given *OMObject object* to this **OMWeakReferenceVectorProperty.**

virtual void prependObject**(const OMObject* object)**

Prepend the given *OMObject object* to this **OMWeakReferenceVectorProperty.**

virtual OMObject* removeObjectAt**(const size_t index)**

Remove the object from this **OMWeakReferenceVectorProperty** at position *index*. Existing objects in this **OMWeakReferenceVectorProperty** at *index* + 1 and higher are shifted down one index position.

virtual void insertObjectAt**(const OMObject* object, const size_t index)**

Insert *object* into this **OMWeakReferenceVectorProperty** at position *index*. Existing objects at *index* and higher are shifted up one index position.

virtual OMStrongReferenceSet* targetSet(void) const

The OMStrongReferenceSet in which the objects referenced by this **OMWeakReferenceVectorProperty** must reside.

OMWeakReferenceVectorProperty::

OMWeakReferenceVectorProperty::(void)

Destructor.

Defined in: OMWeakRefVectorPropertyT.h

Back to OMWeakReferenceVectorProperty

OMWeakReferenceVectorProperty::

OMWeakReferenceVectorProperty::(*OMWeakReferenceVectorProperty*, **const OMPropertyId** *propertyId*, **const wchar_t*** *name*, **const wchar_t*** *targetName***)**

Constructor.

Defined in: OMWeakRefVectorPropertyT.h

Parameters

OMWeakReferenceVectorProperty
: The property id.

propertyId
: The name of this OMWeakReferenceVectorProperty.

name
: The name (as a string) of the the OMProperty instance (a set property) in which the objects referenced by the elements of this OMWeakReferenceVectorProperty reside.

targetName
: The id of the property by which the *ReferencedObject*s are uniquely identified (the key).

Back to OMWeakReferenceVectorProperty

OMWeakReferenceVectorProperty::appendObject

template <class *ReferencedObject***>**
void OMWeakReferenceVectorProperty<*ReferencedObject***>::appendObject(const OMObject*** *object***)**

Append the given *OMObject object* to this OMWeakReferenceVectorProperty.

Defined in: OMWeakRefVectorPropertyT.h

Parameters

object
: The object to append.

Class Template Arguments

ReferencedObject
: The type of the referenced (contained) object. This type must be a descendant of OMStorable.

Back to OMWeakReferenceVectorProperty

OMWeakReferenceVectorProperty::appendValue

template <class *ReferencedObject***>**
void OMWeakReferenceVectorProperty<*ReferencedObject***>::appendValue(const ReferencedObject*** *object***)**

Append the given *ReferencedObject object* to this OMWeakReferenceVectorProperty.

Defined in: OMWeakRefVectorPropertyT.h

Parameters

object
A pointer to a *ReferencedObject*.

Class Template Arguments

ReferencedObject
The type of the referenced (contained) object. This type must be a descendant of OMStorable.

Back to OMWeakReferenceVectorProperty

OMWeakReferenceVectorProperty::bitsSize

template <class *ReferencedObject***>**
size_t OMWeakReferenceVectorProperty<*ReferencedObject***>::bitsSize(void) const**

The size of the raw bits of this OMWeakReferenceVectorProperty. The size is given in bytes.

Defined in: OMWeakRefVectorPropertyT.h

Return Value

The size of the raw bits of this OMWeakReferenceVectorProperty in bytes.

Class Template Arguments

ReferencedObject
The type of the referenced (contained) object. This type must be a descendant of OMStorable.

Back to OMWeakReferenceVectorProperty

OMWeakReferenceVectorProperty::clearValueAt

template <class *ReferencedObject***>**
ReferencedObject* OMWeakReferenceVectorProperty<*ReferencedObject***>::clearValueAt(const size_t** *index***)**

Set the value of this OMWeakReferenceVectorProperty at position *index* to 0.

Defined in: OMWeakRefVectorPropertyT.h

Return Value

A pointer to the old *ReferencedObject*. If lazy loading is enabled and the referenced object was never loaded the value returned is 0.

Parameters

index
The position to clear.

Class Template Arguments

ReferencedObject
The type of the referenced (contained) object. This type must be a descendant of OMStorable.

Back to OMWeakReferenceVectorProperty

OMWeakReferenceVectorProperty::close

template <class *ReferencedObject***>**
void OMWeakReferenceVectorProperty<*ReferencedObject***>::close(void)**

Close this OMWeakReferenceVectorProperty.

Defined in: OMWeakRefVectorPropertyT.h

Class Template Arguments

ReferencedObject
The type of the referenced (contained) object. This type must be a descendant of OMStorable.

Back to OMWeakReferenceVectorProperty

OMWeakReferenceVectorProperty::containsIndex

template <class *ReferencedObject***>**
bool OMWeakReferenceVectorProperty<*ReferencedObject***>::containsIndex(const size_t** *index***) const**

Does this OMWeakReferenceVectorProperty contain *index* ? Is *index* valid ?

Defined in: OMWeakRefVectorPropertyT.h

Return Value

True if the index is valid, false otherwise.

Parameters

index
The index.

Class Template Arguments

ReferencedObject
The type of the referenced (contained) object. This type must be a descendant of OMStorable.

Back to OMWeakReferenceVectorProperty

OMWeakReferenceVectorProperty::containsObject

template <class *ReferencedObject***>**
bool OMWeakReferenceVectorProperty<*ReferencedObject***>::containsObject(const OMObject*** *object***) const**

Does this OMWeakReferenceVectorProperty contain *object* ?

Defined in: OMWeakRefVectorPropertyT.h

Return Value

True if *object* is present, false otherwise.

Parameters

object
The OMObject for which to search.

Class Template Arguments

ReferencedObject
The type of the referenced (contained) object. This type must be a descendant of OMStorable and **OMUnique** .

Back to OMWeakReferenceVectorProperty

OMWeakReferenceVectorProperty::containsValue

template <class *ReferencedObject***>**
bool OMWeakReferenceVectorProperty<*ReferencedObject***>::containsValue(const ReferencedObject*** *object***) const**

Does this OMWeakReferenceVectorProperty contain *object* ?

Defined in: OMWeakRefVectorPropertyT.h

Parameters

object
: A pointer to a *ReferencedObject*.

Class Template Arguments

ReferencedObject
: The type of the referenced (contained) object. This type must be a descendant of OMStorable.

Back to OMWeakReferenceVectorProperty

OMWeakReferenceVectorProperty::count

size_t OMWeakReferenceVectorProperty::count(void) const

The number of *ReferencedObject*s in this OMWeakReferenceVectorProperty.

Defined in: OMWeakRefVectorPropertyT.h

Back to OMWeakReferenceVectorProperty

OMWeakReferenceVectorProperty::countOfValue

template <class *ReferencedObject***>**
size_t OMWeakReferenceVectorProperty<*ReferencedObject***>::countOfValue(const ReferencedObject*** *object***) const**

The number of occurrences of *object* in this OMWeakReferenceVectorProperty.

Defined in: OMWeakRefVectorPropertyT.h

Return Value

The number of occurrences.

Parameters

object
: The object to count.

Class Template Arguments

ReferencedObject

The type of the referenced (contained) object. This type must be a descendant of OMStorable.

Back to OMWeakReferenceVectorProperty

OMWeakReferenceVectorProperty::createIterator

template <class *ReferencedObject***>**
OMReferenceContainerIterator*
OMWeakReferenceVectorProperty<*ReferencedObject***>::createIterator(void) const**

Create an OMReferenceContainerIterator over this OMWeakReferenceVectorProperty.

Defined in: OMWeakRefVectorPropertyT.h

Return Value

An OMReferenceContainerIterator over this OMWeakReferenceVectorProperty.

Class Template Arguments

ReferencedObject
The type of the referenced (contained) object. This type must be a descendant of OMStorable and **OMUnique** .

Back to OMWeakReferenceVectorProperty

OMWeakReferenceVectorProperty::detach

template <class *ReferencedObject***>**
void OMWeakReferenceVectorProperty<*ReferencedObject***>::detach(void)**

Detach this OMWeakReferenceVectorProperty.

Defined in: OMWeakRefVectorPropertyT.h

Class Template Arguments

ReferencedObject
The type of the referenced (contained) object. This type must be a descendant of OMStorable.

Back to OMWeakReferenceVectorProperty

OMWeakReferenceVectorProperty::find

template <class *ReferencedObject***>**
bool OMWeakReferenceVectorProperty<*ReferencedObject***>::find(const size_t** *index***, ReferencedObject*&** *object***) const**

If *index* is valid, get the value of this OMWeakReferenceVectorProperty at position *index* into *object* and return true, otherwise return false.

Defined in: OMWeakRefVectorPropertyT.h

Return Value

True if *index* is valid, false otherwise.

Parameters

index
: The position from which to get the *ReferencedObject*.

object
: A pointer to a *ReferencedObject*.

Class Template Arguments

ReferencedObject
: The type of the referenced (contained) object. This type must be a descendant of OMStorable.

Back to OMWeakReferenceVectorProperty

OMWeakReferenceVectorProperty::findIndex

template <class *ReferencedObject***>**
bool OMWeakReferenceVectorProperty<*ReferencedObject***>::findIndex(const ReferencedObject*** *object***, size_t&** *index***) const**

If this OMWeakReferenceProperty contains *object* then place its index in *index* and return true, otherwise return false.

Defined in: OMWeakRefVectorPropertyT.h

Return Value

True if the object was found, false otherwise.

Parameters

object
: The object for which to search.

index
: The index of the object.

Class Template Arguments

ReferencedObject
: The type of the referenced (contained) object. This type must be a descendant of OMStorable.

Back to OMWeakReferenceVectorProperty

OMWeakReferenceVectorProperty::getBits

template <class *ReferencedObject***>**
void OMWeakReferenceVectorProperty<*ReferencedObject***>::getBits(OMByte*** *bits***, size_t** *size***) const**

Get the raw bits of this OMWeakReferenceVectorProperty. The raw bits are copied to the buffer at address *bits* which is *size* bytes in size.

Defined in: OMWeakRefVectorPropertyT.h

Parameters

bits
: The address of the buffer into which the raw bits are copied.

size
: The size of the buffer.

Class Template Arguments

ReferencedObject
: The type of the referenced (contained) object. This type must be a descendant of OMStorable.

Back to OMWeakReferenceVectorProperty

OMWeakReferenceVectorProperty::getObjectAt

template <class *ReferencedObject***>**
OMObject* OMWeakReferenceVectorProperty<*ReferencedObject***>::getObjectAt(const size_t** *index***) const**

The value of this OMWeakReferenceVectorProperty at position *index*.

Defined in: OMWeakRefVectorPropertyT.h

Return Value

The object.

Parameters

index
The index.

Class Template Arguments

ReferencedObject
The type of the referenced (contained) object. This type must be a descendant of OMStorable.

Back to OMWeakReferenceVectorProperty

OMWeakReferenceVectorProperty::getValueAt

template <class *ReferencedObject***>**
void OMWeakReferenceVectorProperty<*ReferencedObject***>::getValueAt(ReferencedObject*& *object*, const size_t** *index***) const**

Get the value of this OMWeakReferenceVectorProperty at position *index* into *object*.

Defined in: OMWeakRefVectorPropertyT.h

Parameters

object
A pointer to a *ReferencedObject* by reference.

index
The position from which to get the *ReferencedObject*.

Class Template Arguments

ReferencedObject
The type of the referenced (contained) object. This type must be a descendant of OMStorable.

Back to OMWeakReferenceVectorProperty

OMWeakReferenceVectorProperty::grow

void OMWeakReferenceVectorProperty::grow(const size_t *capacity***)**

Increase the capacity of this OMWeakReferenceVectorProperty so that it can contain at least *capacity* *ReferencedObject*s without having to be resized.

Defined in: OMWeakRefVectorPropertyT.h

Parameters

capacity

The desired capacity.

Back to OMWeakReferenceVectorProperty

OMWeakReferenceVectorProperty::indexOfValue

template <class *ReferencedObject***>**
size_t OMWeakReferenceVectorProperty<*ReferencedObject***>::indexOfValue(const ReferencedObject*** *object***) const**

The index of the *ReferencedObject** *object*.

Defined in: OMWeakRefVectorPropertyT.h

Return Value

The index.

Parameters

object
: A pointer to the *ReferencedObject* to find.

Class Template Arguments

ReferencedObject
: The type of the referenced (contained) object. This type must be a descendant of OMStorable.

Back to OMWeakReferenceVectorProperty

OMWeakReferenceVectorProperty::insert

template <class *ReferencedObject***>**
void OMWeakReferenceVectorProperty<*ReferencedObject***>::insert(const ReferencedObject*** *object***)**

Insert *object* into this OMWeakReferenceVectorProperty. This function is redefined from OMContainerProperty as **appendValue**.

Defined in: OMWeakRefVectorPropertyT.h

Parameters

object
: A pointer to a *ReferencedObject*.

Class Template Arguments

ReferencedObject
: The type of the referenced (contained) object. This type must be a descendant of OMStorable.

Back to OMWeakReferenceVectorProperty

OMWeakReferenceVectorProperty::insertAt

template <class *ReferencedObject***>**
void OMWeakReferenceVectorProperty<*ReferencedObject***>::insertAt(const ReferencedObject*** *object***, const size_t** *index***)**

Insert *value* into this OMWeakReferenceVectorProperty at position *index*. Existing values at *index* and higher are shifted up one index position.

Defined in: OMWeakRefVectorPropertyT.h

Parameters

object
: A pointer to a *ReferencedObject*.

index
: The position at which to insert the *ReferencedObject*.

Class Template Arguments

ReferencedObject
: The type of the referenced (contained) object. This type must be a descendant of OMStorable.

Back to OMWeakReferenceVectorProperty

OMWeakReferenceVectorProperty::insertObject

template <class *ReferencedObject***>**
void OMWeakReferenceVectorProperty<*ReferencedObject***>::insertObject(const OMObject*** *object***)**

Insert *object* into this OMWeakReferenceVectorProperty.

Defined in: OMWeakRefVectorPropertyT.h

Parameters

object
: The OMObject to insert.

Class Template Arguments

ReferencedObject
: The type of the referenced (contained) object. This type must be a descendant of OMStorable and **OMUnique** .

Back to OMWeakReferenceVectorProperty

OMWeakReferenceVectorProperty::insertObjectAt

template <class *ReferencedObject***>**
void OMWeakReferenceVectorProperty<*ReferencedObject***>::insertObjectAt(const OMObject*** *object***, const size_t** *index***)**

Insert *object* into this OMWeakReferenceVectorProperty at position *index*. Existing objects at *index* and higher are shifted up one index position.

Defined in: OMWeakRefVectorPropertyT.h

Parameters

object
: The object to insert.

index
: The index at which to insert the object.

Class Template Arguments

ReferencedObject
: The type of the referenced (contained) object. This type must be a descendant of OMStorable.

Back to OMWeakReferenceVectorProperty

OMWeakReferenceVectorProperty::isVoid

template <class *ReferencedObject***>**
bool OMWeakReferenceVectorProperty<*ReferencedObject***>::isVoid(void) const**

Is this OMWeakReferenceVectorProperty void ?

Defined in: OMWeakRefVectorPropertyT.h

Return Value

True if this OMWeakReferenceVectorProperty is void, false otherwise.

Class Template Arguments

ReferencedObject

The type of the referenced (contained) object. This type must be a descendant of OMStorable.

Back to OMWeakReferenceVectorProperty

OMWeakReferenceVectorProperty::OMWeakReferenceVectorProperty

OMWeakReferenceVectorProperty::OMWeakReferenceVectorProperty(const OMPropertyId *propertyId*, **const wchar_t*** *name*, **const OMPropertyId** *keyPropertyId*, **const OMPropertyId*** *targetPropertyPath***)**

Constructor.

Defined in: OMWeakRefVectorPropertyT.h

Parameters

propertyId
The property id.

name
The name of this OMWeakReferenceVectorProperty.

keyPropertyId
The name (as a string) of the the OMProperty instance (a set property) in which the objects referenced by the elements of this OMWeakReferenceVectorProperty reside.

targetPropertyPath
The id of the property by which the *ReferencedObject*s are uniquely identified (the key).

Back to OMWeakReferenceVectorProperty

OMWeakReferenceVectorProperty::prependObject

template <class *ReferencedObject***>**
void OMWeakReferenceVectorProperty<*ReferencedObject***>::prependObject(const OMObject*** *object***)**

Prepend the given *OMObject object* to this OMWeakReferenceVectorProperty.

Defined in: OMWeakRefVectorPropertyT.h

Parameters

object
The object to prepend.

Class Template Arguments

ReferencedObject
The type of the referenced (contained) object. This type must be a descendant of OMStorable.

Back to OMWeakReferenceVectorProperty

OMWeakReferenceVectorProperty::prependValue

template <class *ReferencedObject***>**
void OMWeakReferenceVectorProperty<*ReferencedObject***>::prependValue(const ReferencedObject*** *object***)**

Prepend the given *ReferencedObject object* to this OMWeakReferenceVectorProperty.

Defined in: OMWeakRefVectorPropertyT.h

Parameters

object
: A pointer to a *ReferencedObject*.

Class Template Arguments

ReferencedObject
: The type of the referenced (contained) object. This type must be a descendant of OMStorable.

Back to OMWeakReferenceVectorProperty

OMWeakReferenceVectorProperty::removeAllObjects

template <class *ReferencedObject***>**
void OMWeakReferenceVectorProperty<*ReferencedObject***>::removeAllObjects(void)**

Remove all objects from this OMWeakReferenceVectorProperty.

Defined in: OMWeakRefVectorPropertyT.h

Class Template Arguments

ReferencedObject
: The type of the referenced (contained) object. This type must be a descendant of OMStorable.

Back to OMWeakReferenceVectorProperty

OMWeakReferenceVectorProperty::removeAt

template <class *ReferencedObject***>**
ReferencedObject* OMWeakReferenceVectorProperty<*ReferencedObject***>::removeAt(const size_t** *index***)**

Remove the object from this OMWeakReferenceVectorProperty at position *index*. Existing objects in this OMWeakReferenceVectorProperty at *index* + 1 and higher are shifted down one index position.

Defined in: OMWeakRefVectorPropertyT.h

Return Value

A pointer to the removed *ReferencedObject*. If lazy loading is enabled and the referenced object was never loaded the value returned is 0.

Parameters

index
: The position from which to remove the *ReferencedObject*.

Class Template Arguments

ReferencedObject
: The type of the referenced (contained) object. This type must be a descendant of OMStorable.

Back to OMWeakReferenceVectorProperty

OMWeakReferenceVectorProperty::removeFirst

template <class *ReferencedObject***>**
ReferencedObject* OMWeakReferenceVectorProperty<*ReferencedObject***>::removeFirst(void)**

Remove the first (index == 0) object from this OMWeakReferenceVectorProperty. Existing objects in this OMWeakReferenceVectorProperty are shifted down one index position.

Defined in: OMWeakRefVectorPropertyT.h

Return Value

A pointer to the removed *ReferencedObject*. If lazy loading is enabled and the referenced object was never loaded the value returned is 0.

Class Template Arguments

ReferencedObject
: The type of the referenced (contained) object. This type must be a descendant of OMStorable.

Back to OMWeakReferenceVectorProperty

OMWeakReferenceVectorProperty::removeLast

template <class *ReferencedObject***>**
ReferencedObject* OMWeakReferenceVectorProperty<*ReferencedObject***>::removeLast(void)**

Remove the last (index == count() - 1) object from this OMWeakReferenceVectorProperty.

Defined in: OMWeakRefVectorPropertyT.h

Return Value

A pointer to the removed *ReferencedObject*. If lazy loading is enabled and the referenced object was never loaded the value returned is 0.

Class Template Arguments

ReferencedObject
: The type of the referenced (contained) object. This type must be a descendant of OMStorable.

Back to OMWeakReferenceVectorProperty

OMWeakReferenceVectorProperty::removeObject

template <class *ReferencedObject***>**
void OMWeakReferenceVectorProperty<*ReferencedObject***>::removeObject(const OMObject*** *object***)**

Remove *object* from this OMWeakReferenceVectorProperty.

Defined in: OMWeakRefVectorPropertyT.h

Parameters

object
: The OMObject to remove.

Class Template Arguments

ReferencedObject
: The type of the referenced (contained) object. This type must be a descendant of OMStorable and **OMUnique** .

Back to OMWeakReferenceVectorProperty

OMWeakReferenceVectorProperty::removeObjectAt

template <class *ReferencedObject***>**
OMObject* OMWeakReferenceVectorProperty<*ReferencedObject***>::removeObjectAt(const size_t** *index***)**

Remove the object from this OMWeakReferenceVectorProperty at position *index*. Existing objects in this OMWeakReferenceVectorProperty at *index* + 1 and higher are shifted down one index position.

Defined in: OMWeakRefVectorPropertyT.h

Parameters

index
: The index of the object to remove.

Class Template Arguments

ReferencedObject
: The type of the referenced (contained) object. This type must be a descendant of OMStorable.

Back to OMWeakReferenceVectorProperty

OMWeakReferenceVectorProperty::removeProperty

template <class *ReferencedObject***>**
void OMWeakReferenceVectorProperty<*ReferencedObject***>::removeProperty(void)**

Remove this optional OMWeakReferenceVectorProperty.

Defined in: OMWeakRefVectorPropertyT.h

Class Template Arguments

ReferencedObject
: The type of the referenced (contained) object. This type must be a descendant of OMStorable.

Back to OMWeakReferenceVectorProperty

OMWeakReferenceVectorProperty::removeValue

template <class *ReferencedObject***>**
void OMWeakReferenceVectorProperty<*ReferencedObject***>::removeValue(const ReferencedObject*** *object***)**

Remove *object* from this OMWeakReferenceVectorProperty.

Defined in: OMWeakRefVectorPropertyT.h

Parameters

object
: A pointer to a *ReferencedObject*.

Class Template Arguments

ReferencedObject
: The type of the referenced (contained) object. This type must be a descendant of OMStorable.

Back to OMWeakReferenceVectorProperty

OMWeakReferenceVectorProperty::restore

template <class *ReferencedObject***>**
void OMWeakReferenceVectorProperty<*ReferencedObject***>::restore(size_t** *externalSize***)**

Restore this OMWeakReferenceVectorProperty, the external (persisted) size of the OMWeakReferenceVectorProperty is *externalSize*.

Defined in: OMWeakRefVectorPropertyT.h

Parameters

externalSize
: The external (persisted) size of the OMWeakReferenceVectorProperty.

Class Template Arguments

ReferencedObject
: The type of the referenced (contained) object. This type must be a descendant of OMStorable.

Back to OMWeakReferenceVectorProperty

OMWeakReferenceVectorProperty::save

template <class *ReferencedObject***>**
void OMWeakReferenceVectorProperty<*ReferencedObject***>::save(void) const**

Save this OMWeakReferenceVectorProperty.

Defined in: OMWeakRefVectorPropertyT.h

Class Template Arguments

ReferencedObject
: The type of the referenced (contained) object. This type must be a descendant of OMStorable.

Back to OMWeakReferenceVectorProperty

OMWeakReferenceVectorProperty::setBits

template <class *ReferencedObject***>**
void OMWeakReferenceVectorProperty<*ReferencedObject***>::setBits(const OMByte*** *bits***, size_t** *size***)**

Set the raw bits of this OMWeakReferenceVectorProperty. The raw bits are copied from the buffer at address *bits* which is *size* bytes in size.

Defined in: OMWeakRefVectorPropertyT.h

Parameters

bits
: The address of the buffer from which the raw bits are copied.

size
: The size of the buffer.

Class Template Arguments

ReferencedObject
: The type of the referenced (contained) object. This type must be a descendant of OMStorable.

Back to OMWeakReferenceVectorProperty

OMWeakReferenceVectorProperty::setObjectAt

template <class *ReferencedObject***>**
OMObject* OMWeakReferenceVectorProperty<*ReferencedObject***>::setObjectAt(const OMObject*** *object***, const size_t** *index***)**

Set the value of this OMWeakReferenceVectorProperty at position *index* to *object*.

Defined in: OMWeakRefVectorPropertyT.h

Return Value

The old object.

Parameters

object
: The new object.

index
: The index.

Class Template Arguments

ReferencedObject
: The type of the referenced (contained) object. This type must be a descendant of OMStorable.

Back to OMWeakReferenceVectorProperty

OMWeakReferenceVectorProperty::setValueAt

template <class *ReferencedObject***>**
ReferencedObject* OMWeakReferenceVectorProperty<*ReferencedObject***>::setValueAt(const ReferencedObject*** *object***, const size_t** *index***)**

Set the value of this OMWeakReferenceVectorProperty at position *index* to *object*.

Defined in: OMWeakRefVectorPropertyT.h

Return Value

A pointer to the old *ReferencedObject*. If lazy loading is enabled and the referenced object was never loaded the value returned is 0.

Parameters

object
: A pointer to the new *ReferencedObject*.

index
: The position at which to insert the *ReferencedObject*.

Class Template Arguments

ReferencedObject
: The type of the referenced (contained) object. This type must be a descendant of OMStorable.

Back to OMWeakReferenceVectorProperty

OMWeakReferenceVectorProperty::valueAt

template <class *ReferencedObject***>**
ReferencedObject* OMWeakReferenceVectorProperty<*ReferencedObject***>::valueAt(const size_t** *index***) const**

The value of this OMWeakReferenceVectorProperty at position *index*.

Defined in: OMWeakRefVectorPropertyT.h

Return Value

A pointer to the *ReferencedObject*.

Parameters

index
The position from which to get the *ReferencedObject*.

Class Template Arguments

ReferencedObject
The type of the referenced (contained) object. This type must be a descendant of OMStorable.

Back to OMWeakReferenceVectorProperty

OMWideStringProperty class

OMWideStringProperty **class OMWideStringProperty: public OMCharacterStringProperty**

Persistent wide character strings supported by the Object Manager.

Defined in: OMWideStringProperty.h

Author

Tim Bingham - tjb - (Avid Technology, Inc.)

Class Members

Public members.
OMWideStringProperty**(const OMPropertyId propertyId, const wchar_t* name)**
Constructor.
virtual ~OMWideStringProperty**(void)**
Destructor.
OMWideStringProperty& operator=**(const wchar_t* value)**
Assignment operator.

OMWideStringProperty::OMWideStringProperty

OMWideStringProperty::OMWideStringProperty(const OMPropertyId *propertyId*, **const wchar_t*** *name***)**

Constructor.

Defined in: OMWideStringProperty.cpp

Parameters

propertyId

The property id.

name

The name of this OMWideStringProperty.

Back to OMWideStringProperty

OMWideStringProperty::operator=

OMWideStringProperty& OMWideStringProperty::operator=(const wchar_t* *value***)**

Assignment operator.

Defined in: OMWideStringProperty.cpp

Return Value

The OMWideStringProperty resulting from the assignment.

Parameters

value

The new value for this OMWideStringProperty.

Back to OMWideStringProperty

OMWideStringProperty::~OMWideStringProperty

OMWideStringProperty::~OMWideStringProperty(void)

Destructor.

Defined in: OMWideStringProperty.cpp

Back to OMWideStringProperty

OMXMLStoredObject class

OMXMLStoredObject **class OMXMLStoredObject**

In-memory representation of an object persisted in an eXtensible Markup Language (XML) text file.

Defined in: OMXMLStoredObject.h

Author

Tim Bingham - tjb - (Avid Technology, Inc.)

Class Members

Static members.

static OMXMLStoredObject* openRead**(OMRawStorage* rawStorage)**

Open the root **OMXMLStoredObject** in the raw storage *rawStorage* for reading only.

static OMXMLStoredObject* openModify**(OMRawStorage* rawStorage)**

Open the root **OMXMLStoredObject** in the raw storage *rawStorage* for modification.

static OMXMLStoredObject* createWrite**(OMRawStorage* rawStorage, const OMByteOrder byteOrder)**

Create a new root **OMXMLStoredObject** in the raw storage *rawStorage*. The byte order of the newly created root is given by *byteOrder*.

static OMXMLStoredObject* createModify**(OMRawStorage* rawStorage, const OMByteOrder byteOrder)**

Create a new root **OMXMLStoredObject** in the raw storage *rawStorage*. The byte order of the newly created root is given by *byteOrder*.

static bool isRecognized**(const wchar_t* fileName, OMFileSignature& signature)**

Is the file named *fileName* a recognized file ? If so, the result is true, and the signature is returned in *signature*.

static bool isRecognized**(OMRawStorage* rawStorage, OMFileSignature& signature)**

Does *rawStorage* contain a recognized file ? If so, the result is true, and the signature is returned in *signature*.

static bool isRecognized**(const OMFileSignature& signature)**

Is *signature* recognized ?

Class Members

Public members.

virtual ~OMXMLStoredObject**(void)**

Destructor.

virtual OMStoredObject* create**(const wchar_t* name)**

Create a new **OMXMLStoredObject**, named *name*, contained by this **OMXMLStoredObject.**

Developer Notes

The name argument to this member function doesn't make sense for all derived instances of OMStoredObject.

virtual OMStoredObject* open**(const wchar_t* name)**

Open an exsiting **OMXMLStoredObject**, named *name*, contained by this **OMXMLStoredObject.**

Developer Notes

The name argument to this member function doesn't make sense for all derived instances of OMStoredObject.

virtual void close**(void)**

Close this **OMXMLStoredObject.**

virtual OMByteOrder byteOrder**(void) const**

The byte order of this **OMXMLStoredObject.**

Developer Notes

This member function doesn't make sense for all derived instances of OMStoredObject.

virtual void save**(const OMStoredObjectIdentification& id)**

Save the **OMStoredObjectIdentification** *id* in this **OMXMLStoredObject.**

virtual void save**(const OMPropertySet& properties)**

Save the OMPropertySet *properties* in this **OMXMLStoredObject.**

virtual void save**(const OMSimpleProperty& property)**

Save the OMSimpleProperty *property* in this **OMXMLStoredObject.**

virtual void save**(const OMStrongReference& singleton)**

Save the OMStrongReference *singleton* in this **OMXMLStoredObject.**

virtual void save**(const OMStrongReferenceVector& vector)**

Save the OMStrongReferenceVector *vector* in this **OMXMLStoredObject.**

virtual void save**(const OMStrongReferenceSet& set)**

Save the OMStrongReferenceSet *set* in this **OMXMLStoredObject.**

virtual void save**(const OMWeakReference& singleton)**

Save the OMWeakReference *singleton* in this **OMXMLStoredObject.**

virtual void save**(const OMWeakReferenceVector& vector)**

Save the OMWeakReferenceVector *vector* in this **OMXMLStoredObject.**

virtual void save**(const OMWeakReferenceSet& set)**

Save the OMWeakReferenceSet *set* in this **OMXMLStoredObject.**

virtual void save**(const OMPropertyTable* table)**

Save the OMPropertyTable *table* in this **OMXMLStoredObject.**

Developer Notes

Does this member function make sense for all derived instances of OMStoredObject ?

virtual void save**(const OMDataStream& stream)**

Save the OMDataStream *stream* in this **OMXMLStoredObject.**

virtual void restore**(OMStoredObjectIdentification& id)**

Restore the **OMStoredObjectIdentification** of this **OMXMLStoredObject** into *id*.

virtual void restore**(OMPropertySet& properties)**

Restore the OMPropertySet *properties* into this **OMXMLStoredObject.**

virtual void restore**(OMSimpleProperty& property, size_t externalSize)**

Restore the OMSimpleProperty *property* into this **OMXMLStoredObject.**

Developer Notes

The externalSize argument to this member function doesn't make sense for all derived instances of OMStoredObject.

virtual void restore**(OMStrongReference& singleton, size_t externalSize)**

Restore the OMStrongReference *singleton* into this **OMXMLStoredObject.**

virtual void restore**(OMStrongReferenceVector& vector, size_t externalSize)**

Restore the OMStrongReferenceVector *vector* into this **OMXMLStoredObject.**

virtual void restore**(OMStrongReferenceSet& set, size_t externalSize)**

Restore the OMStrongReferenceSet *set* into this **OMXMLStoredObject.**

virtual void restore**(OMWeakReference& singleton, size_t externalSize)**

Restore the OMWeakReference *singleton* into this **OMXMLStoredObject.**

virtual void restore**(OMWeakReferenceVector& vector, size_t externalSize)**

Restore the OMWeakReferenceVector *vector* into this **OMXMLStoredObject.**

virtual void restore**(OMWeakReferenceSet& set, size_t externalSize)**

Restore the OMWeakReferenceSet *set* into this **OMXMLStoredObject.**

virtual void restore**(OMPropertyTable*& table)**

Restore the OMPropertyTable in this **OMXMLStoredObject.**

Developer Notes

Does this member function make sense for all derived instances of OMStoredObject ?

virtual void restore**(OMDataStream& stream, size_t externalSize)**

Restore the OMDataStream *stream* into this **OMXMLStoredObject.**

virtual OMStoredStream* openStoredStream**(const OMDataStream& property)**

Open the OMStoredStream representing the property *stream* contained within this **OMXMLStoredObject.**

virtual OMStoredStream* createStoredStream**(const OMDataStream& property)**

Create an OMStoredStream representing the property *stream* contained within this **OMXMLStoredObject.**

Class Members

Private members.

OMXMLStoredObject**(OMRawStorage* s, const OMByteOrder byteOrder)**

Constructor.

OMXMLStoredObject::byteOrder

OMByteOrder OMXMLStoredObject::byteOrder(void) const

The byte order of this OMXMLStoredObject.

Defined in: OMXMLStoredObject.cpp

Return Value

The byte order.

Back to OMXMLStoredObject

OMXMLStoredObject::close

void OMXMLStoredObject::close(void)

Close this OMXMLStoredObject.

Defined in: OMXMLStoredObject.cpp

Back to OMXMLStoredObject

OMXMLStoredObject::create

OMStoredObject* OMXMLStoredObject::create(const wchar_t* { *TRACE***)**

Create a new OMXMLStoredObject, named *name*, contained by this OMXMLStoredObject.

Defined in: OMXMLStoredObject.cpp

Return Value

A new OMXMLStoredObject contained by this OMXMLStoredObject. name */)

Parameters

TRACE
The name to be used for the new OMXMLStoredObject.

Back to OMXMLStoredObject

OMXMLStoredObject::createModify

OMXMLStoredObject* OMXMLStoredObject::createModify(OMRawStorage* *rawStorage***, const OMByteOrder** *byteOrder***)**

Create a new root OMXMLStoredObject in the raw storage *rawStorage*. The byte order of the newly created root is given by *byteOrder*.

Defined in: OMXMLStoredObject.cpp

Return Value

An OMXMLStoredObject representing the root object.

Parameters

rawStorage
The raw storage in which to create the file.

byteOrder
The desired byte ordering for the new file.

Back to OMXMLStoredObject

OMXMLStoredObject::createStoredStream

OMStoredStream* OMXMLStoredObject::createStoredStream(const OMDataStream& *property***)**

Create an OMStoredStream representing the property *stream* contained within this OMXMLStoredObject.

Defined in: OMXMLStoredObject.cpp

Return Value

The newly created OMStoredStream.

Parameters

property
The OMDataStream to be created.
Back to OMXMLStoredObject

OMXMLStoredObject::createWrite

OMXMLStoredObject* OMXMLStoredObject::createWrite(OMRawStorage* *rawStorage***, const OMByteOrder** *byteOrder***)**

Create a new root OMXMLStoredObject in the raw storage *rawStorage*. The byte order of the newly created root is given by *byteOrder*.

Defined in: OMXMLStoredObject.cpp

Return Value

An OMXMLStoredObject representing the root object.

Parameters

rawStorage
The raw storage in which to create the file.
byteOrder
The desired byte ordering for the new file.
Back to OMXMLStoredObject

OMXMLStoredObject::isRecognized

bool OMXMLStoredObject::isRecognized(const OMFileSignature& *signature***)**

Is *signature* recognized ? If so, the result is true.

Defined in: OMXMLStoredObject.cpp

Return Value

True if the signature is recognized, false otherwise.

Parameters

signature
The signature to check.
Back to OMXMLStoredObject

OMXMLStoredObject::isRecognized

bool OMXMLStoredObject::isRecognized(OMRawStorage* *ANAME*, *rawStorage***)**

Does *rawStorage* contain a recognized file ? If so, the result is true, and the signature is returned in *signature*.

Defined in: OMXMLStoredObject.cpp

Return Value

True if the OMRawStorage contains a recognized file, false otherwise.

Parameters

ANAME
: The OMRawStorage to check.

rawStorage
: If recognized, the file signature.

Back to OMXMLStoredObject

OMXMLStoredObject::isRecognized

bool OMXMLStoredObject::isRecognized(const wchar_t* *ANAME*, *fileName***)**

Is the file named *fileName* a recognized file ? If so, the result is true, and the signature is returned in *signature*.

Defined in: OMXMLStoredObject.cpp

Return Value

True if the file is recognized, false otherwise.

Parameters

ANAME
: The name of the file to check.

fileName
: If recognized, the file signature.

Back to OMXMLStoredObject

OMXMLStoredObject::OMXMLStoredObject

OMXMLStoredObject::OMXMLStoredObject(OMRawStorage* *s*, **const OMByteOrder** *byteOrder***)**

Constructor.

Defined in: OMXMLStoredObject.cpp

Parameters

s
The OMRawStorage on which this OMXMLStoredObject resides.
byteOrder
TBS
Back to OMXMLStoredObject

OMXMLStoredObject::open

OMStoredObject* OMXMLStoredObject::open(const wchar_t* { *TRACE***)**

Open an exsiting OMXMLStoredObject, named *name*, contained by this OMXMLStoredObject.

Defined in: OMXMLStoredObject.cpp

Return Value

The existing OMXMLStoredObject contained by this OMXMLStoredObject. name */)

Parameters

TRACE
The name of the existing OMXMLStoredObject.
Back to OMXMLStoredObject

OMXMLStoredObject::openModify

OMXMLStoredObject* OMXMLStoredObject::openModify(OMRawStorage* *ANAME***)**

Open the root OMXMLStoredObject in the raw storage

Defined in: OMXMLStoredObject.cpp

Return Value

An OMXMLStoredObject representing the root object. *rawStorage* for modification.

Parameters

ANAME
The raw storage in which to open the file.
Back to OMXMLStoredObject

OMXMLStoredObject::openRead

OMXMLStoredObject* OMXMLStoredObject::openRead(OMRawStorage* *ANAME***)**

Open the root OMXMLStoredObject in the raw storage *rawStorage* for reading only.

Defined in: OMXMLStoredObject.cpp

Return Value

An OMXMLStoredObject representing the root object.

Parameters

ANAME
The raw storage in which to open the file.

Back to OMXMLStoredObject

OMXMLStoredObject::openStoredStream

OMStoredStream* OMXMLStoredObject::openStoredStream(const OMDataStream& { *TRACE***)**

Open the OMStoredStream representing the property *stream* contained within this OMXMLStoredObject.

Defined in: OMXMLStoredObject.cpp

Return Value

The newly created OMStoredStream.

Parameters

TRACE
The OMDataStream to be opened.

Back to OMXMLStoredObject

OMXMLStoredObject::restore

void OMXMLStoredObject::restore(OMWeakReferenceSet& size_t { *TRACE***, OMXMLStoredObject::** *restore***)**

Restore the OMWeakReferenceSet *set* into this OMXMLStoredObject.

Defined in: OMXMLStoredObject.cpp

Parameters

TRACE
The newly restored OMWeakReferenceSet.

restore
The external size. set */,

Back to OMXMLStoredObject

OMXMLStoredObject::restore

void OMXMLStoredObject::restore(OMDataStream& size_t { *TRACE*, **OMXMLStoredObject::** *restore***)**

Restore the OMDataStream *stream* into this OMXMLStoredObject.

Defined in: OMXMLStoredObject.cpp

Parameters

TRACE
The newly restored OMDataStream.
restore
The external size. stream */,
Back to OMXMLStoredObject

OMXMLStoredObject::restore

void OMXMLStoredObject::restore(OMWeakReferenceVector& size_t { *TRACE*, **OMXMLStoredObject::** *restore***)**

Restore the OMWeakReferenceVector *vector* into this OMXMLStoredObject.

Defined in: OMXMLStoredObject.cpp

Parameters

TRACE
The newly restored OMWeakReferenceVector.
restore
The external size. vector */,
Back to OMXMLStoredObject

OMXMLStoredObject::restore

void OMXMLStoredObject::restore(*TRACE***)**

Restore the OMPropertyTable in this OMXMLStoredObject.

Defined in: OMXMLStoredObject.cpp

Parameters

TRACE

The newly restored OMPropertyTable. table */)

Back to OMXMLStoredObject

OMXMLStoredObject::restore

void OMXMLStoredObject::restore(OMWeakReference& size_t { *TRACE*, OMXMLStoredObject:: *restore*)

Restore the OMWeakReference *singleton* into this OMXMLStoredObject.

Defined in: OMXMLStoredObject.cpp

Parameters

TRACE
The newly restored OMWeakReference.

restore
The external size. singleton */,

Back to OMXMLStoredObject

OMXMLStoredObject::restore

void OMXMLStoredObject::restore(OMStrongReferenceSet& size_t { *TRACE*, OMXMLStoredObject:: *restore*)

Restore the OMStrongReferenceSet *set* into this OMXMLStoredObject.

Defined in: OMXMLStoredObject.cpp

Parameters

TRACE
The newly restored OMStrongReferenceSet.

restore
The external size. set */,

Back to OMXMLStoredObject

OMXMLStoredObject::restore

void OMXMLStoredObject::restore(*TRACE*)

Restore the OMPropertySet *properties* into this OMXMLStoredObject.

Defined in: OMXMLStoredObject.cpp

Parameters

TRACE
The newly restored OMPropertySet. properties */)
Back to OMXMLStoredObject

OMXMLStoredObject::restore

void OMXMLStoredObject::restore(OMStrongReferenceVector& size_t { *TRACE*, OMXMLStoredObject:: *restore*)

Restore the OMStrongReferenceVector *vector* into this OMXMLStoredObject.

Defined in: OMXMLStoredObject.cpp

Parameters

TRACE
The newly restored OMStrongReferenceVector.
restore
The external size. vector */,
Back to OMXMLStoredObject

OMXMLStoredObject::restore

void OMXMLStoredObject::restore(OMStrongReference& size_t { *TRACE*, OMXMLStoredObject:: *restore*)

Restore the OMStrongReference *singleton* into this OMXMLStoredObject.

Defined in: OMXMLStoredObject.cpp

Parameters

TRACE
The newly restored OMStrongReference.
restore
The external size. singleton */,
Back to OMXMLStoredObject

OMXMLStoredObject::restore

void OMXMLStoredObject::restore(OMSimpleProperty& size_t { *TRACE*, OMXMLStoredObject:: *restore*)

Restore the OMSimpleProperty *property* into this OMXMLStoredObject.

Defined in: OMXMLStoredObject.cpp

Parameters

TRACE
The newly restored OMSimpleProperty
restore
The external size. property */,
Back to OMXMLStoredObject

OMXMLStoredObject::restore

void OMXMLStoredObject::restore(*TRACE***)**

Restore the **OMStoredObjectIdentification** of this OMXMLStoredObject into *id*.

Defined in: OMXMLStoredObject.cpp

Parameters

TRACE
The newly restored **OMStoredObjectIdentification.** id */)
Back to OMXMLStoredObject

OMXMLStoredObject::save

void OMXMLStoredObject::save(const OMStrongReferenceSet& *set***)**

Save the OMStrongReferenceSet *set* in this OMXMLStoredObject.

Defined in: OMXMLStoredObject.cpp

Parameters

set
The OMStrongReference to save.
Back to OMXMLStoredObject

OMXMLStoredObject::save

void OMXMLStoredObject::save(const OMPropertySet& *properties***)**

Save the OMPropertySet *properties* in this OMXMLStoredObject.

Defined in: OMXMLStoredObject.cpp

Parameters

properties
The OMPropertySet to save.
Back to OMXMLStoredObject

OMXMLStoredObject::save

void OMXMLStoredObject::save(const OMSimpleProperty& *property*)

Save the OMSimpleProperty *property* in this OMXMLStoredObject.

Defined in: OMXMLStoredObject.cpp

Parameters

property
The OMSimpleProperty to save.
Back to OMXMLStoredObject

OMXMLStoredObject::save

void OMXMLStoredObject::save(const OMStrongReference& *singleton*)

Save the OMStrongReference *singleton* in this OMXMLStoredObject.

Defined in: OMXMLStoredObject.cpp

Parameters

singleton
The OMStrongReference to save.
Back to OMXMLStoredObject

OMXMLStoredObject::save

void OMXMLStoredObject::save(const OMStrongReferenceVector& *vector*)

Save the OMStrongReferenceVector *vector* in this OMXMLStoredObject.

Defined in: OMXMLStoredObject.cpp

Parameters

vector
The OMStrongReferenceVector to save.
Back to OMXMLStoredObject

OMXMLStoredObject::save

void OMXMLStoredObject::save(const OMStoredObjectIdentification& *id***)**

Save the **OMStoredObjectIdentification** *id* in this OMXMLStoredObject.

Defined in: OMXMLStoredObject.cpp

Parameters

id
The **OMStoredObjectIdentification** to save.

Back to OMXMLStoredObject

OMXMLStoredObject::save

void OMXMLStoredObject::save(const OMWeakReferenceVector& *vector***)**

Save the OMWeakReferenceVector *vector* in this OMXMLStoredObject.

Defined in: OMXMLStoredObject.cpp

Parameters

vector
The OMWeakReferenceVector to save.

Back to OMXMLStoredObject

OMXMLStoredObject::save

void OMXMLStoredObject::save(const OMWeakReferenceSet& *set***)**

Save the OMWeakReferenceSet *set* in this OMXMLStoredObject.

Defined in: OMXMLStoredObject.cpp

Parameters

set
The OMWeakReferenceSet to save.

Back to OMXMLStoredObject

OMXMLStoredObject::save

void OMXMLStoredObject::save(*TRACE***)**

Save the OMPropertyTable *table* in this OMXMLStoredObject.

Defined in: OMXMLStoredObject.cpp

Parameters

TRACE
The OMPropertyTable to save. table */)

Back to OMXMLStoredObject

OMXMLStoredObject::save

void OMXMLStoredObject::save(const OMDataStream& *stream***)**

Save the OMDataStream *stream* in this OMXMLStoredObject.

Defined in: OMXMLStoredObject.cpp

Parameters

stream
The OMDataStream to save.

Back to OMXMLStoredObject

OMXMLStoredObject::save

void OMXMLStoredObject::save(const OMWeakReference& *singleton***)**

Save the OMWeakReference *singleton* in this OMXMLStoredObject.

Defined in: OMXMLStoredObject.cpp

Parameters

singleton
The OMWeakReference to save.

Back to OMXMLStoredObject

OMXMLStoredObject::~OMXMLStoredObject

OMXMLStoredObject::~OMXMLStoredObject(void)

Destructor.

Defined in: OMXMLStoredObject.cpp

Back to OMXMLStoredObject

OMXMLStoredStream class

OMXMLStoredStream **class OMXMLStoredStream: public OMStoredStream**

Implementation of OMStoredStream for XML.

Defined in: OMXMLStoredStream.h

Author

Tim Bingham - tjb - (Avid Technology, Inc.)

Class Members

Public members.

OMXMLStoredStream(OMRawStorage* store)
: Constructor.

~OMXMLStoredStream(void)
: Destructor.

virtual void read(void* data, size_t size) const
: Read *size* bytes from this OMStoredStream into the buffer at address *data*.

virtual void read(OMByte* data, const OMUInt32 bytes, OMUInt32& bytesRead) const
: Attempt to read *bytes* bytes from this OMStoredStream into the buffer at address *data*. The actual number of bytes read is returned in *bytesRead*.

virtual void write(void* data, size_t size)
: Write *size* bytes from the buffer at address *data* to this OMStoredStream.

virtual void write(const OMByte* data, const OMUInt32 bytes, OMUInt32& bytesWritten)
: Attempt to write *bytes* bytes from the buffer at address *data* to this OMStoredStream. The actual number of bytes written is returned in *bytesWritten*.

virtual OMUInt64 size(void) const
: The size of this OMStoredStream in bytes.

virtual void setSize(const OMUInt64 newSize)
: Set the size of this OMStoredStream to *bytes*.

virtual OMUInt64 position(void) const
: The current position for **read()** and **write()**, as an offset in bytes from the begining of this OMStoredStream.

virtual void setPosition(const OMUInt64 offset)
: Set the current position for **read()** and **write()**, as an offset in bytes from the begining of this OMStoredStream.

virtual void close(void)
: Close this OMStoredStream.

Class Members

Private members.

POSTCONDITION

define POSTCONDITION(*name*, *expression***)**

Assert (when enabled with OM_ENABLE_DEBUG) that the postcondition described by *name* and *expression* is true. An invocation of this macro must be preceeded by an invocation of the TRACE macro.

Defined in: OMAssertions.h

Parameters

name
: The name of the postcondition. The postcondition name is a description of the postcondition that makes sense from the internal point of view (that of someone reading the source text). The name comprises a portion of the message that is printed if the postcondition is violated. The message that is printed makes sense from the external point of view.

expression
: The postcondition expression. The expression should be free of side effects.

PRECONDITION

define PRECONDITION(*name*, *expression***)**

Assert (when enabled with OM_ENABLE_DEBUG) that the precondition described by *name* and *expression* is true. An invocation of this macro must be preceeded by an invocation of the TRACE macro.

Defined in: OMAssertions.h

Parameters

name
: The name of the precondition. The precondition name is a description of the precondition that makes sense from the internal point of view (that of someone reading the source text). The name comprises a portion of the message that is printed if the precondition is violated. The message that is printed makes sense from the external point of view.

expression
: The precondition expression. The expression should be free of side effects.

---

reportAssertionViolation

void reportAssertionViolation(char* *assertionKind*, **char*** *assertionName*, **char*** *expressionString*, **char*** *routineName*, **char*** *fileName*, **OMUInt32** *lineNumber***)**

Report an assertion violation. Used to implement the PRECONDITION, POSTCONDITION and ASSERT macros.

Defined in: OMAssertions.h

Parameters

assertionKind
The kind of assertion - PRECONDITION, POSTCONDITION or ASSERT.
assertionName
The name of the assertion. The assertion name is a description of the assertion that makes sense from the internal point of view (that of someone reading the source text). The name comprises a portion of the message that is printed by this routine. The message that is printed makes sense from the external point of view.
expressionString
The expression, as a text string, that was found not to be true.
routineName
The name of the routine in which the assertion violation occurred.
fileName
The name of the source file in which the assertion violation occurred.
lineNumber
The line number at which the assertion violation occurred.

SAVE

define SAVE(*name*, *type***)**

Save the value of a variable on entry to a routine for later retrieval in the postcondition with OLD.

Defined in: OMAssertions.h

Parameters

name
The name of the variable to save.
type
The type of the variable.

SAVE_EXPRESSION

define SAVE_EXPRESSION(*name*, *expression*, *type***)**

Save the value of an expression on entry to a routine for later retrieval in the postcondition with OLD.

Defined in: OMAssertions.h

Parameters

name

The name of the saved expression.

expression

The expression to save.

type

The type of the expression.

---

saveString

char* saveString(const char* *string***)**

Save a character string.

Defined in: OMUtilities.h

Return Value

The saved character string.

Parameters

string

The character string to save.

---

saveWideString

wchar_t* saveWideString(const wchar_t* *string***)**

Save a wide character string. Same as saveString() but for wide characters.

Defined in: OMUtilities.h

Return Value

The saved wide character string.

Parameters

string

The wide character string to save.

---

squeezeWideString

size_t squeezeWideString(const wchar_t* *clearName***, size_t** *clearNameSize***, wchar_t*** *squeezedName***, size_t** *squeezedNameSize***)**

Squeeze a string to fit within a given size, omitting characters from the center if necessary.

Defined in: OMUtilities.h

Return Value

The size, in characters, of the resulting string.

Parameters

clearName
: The string to be squeezed.

clearNameSize
: The length of the string to be squeezed.

squeezedName
: The resulting string.

squeezedNameSize
: The size, in characters, of the result buffer.

---

stringSize

size_t stringSize(OMUInt32 *i*)

The number of characters needed to represent *i* as a hexadecimal string without leading zeros.

Defined in: OMUtilities.h

Return Value

The number of characters.

Parameters

i
: A non-zero unsigned integer.

---

toWideString

void toWideString(OMUInt32 *i*, wchar_t* *result*, size_t *resultSize*)

Convert *i* hexadecimal string without leading zeros.

Defined in: OMUtilities.h

Parameters

i

A non-zero unsigned integer.

result

The resulting string.

resultSize

The size, in characters, of the result buffer.

TRACE

define TRACE(*routine***)**

Print routine tracing information (when enabled with OM_ENABLE_DEBUG and OM_ENABLE_TRACE). The routine name provided is used by other assertions.

Defined in: OMAssertions.h

Parameters

routine

The routine name. For the most explicit output, names of member functions should be prefixed with the class name, as in className::functionName.

---

trace

void trace(const char* *routineName***)**

Output routine tracing information.

Defined in: OMAssertions.h

Parameters

routineName

The name of the routine.

---

validOMString

bool validOMString(const OMCharacter* *string***)**

Is the given OMCharacter string valid ? Use **validOMString** in expressions passed to the assertion macros PRECONDITION, POSTCONDITION and ASSERT.

Defined in: OMAssertions.h

Return Value

True if the OMCharacter string is valid, false otherwise.

Parameters

string
The OMCharacter string to check for validity.

---

validString

bool validString(const char* *string*)

Is the given string valid ? Use **validString** in expressions passed to the assertion macros PRECONDITION, POSTCONDITION and ASSERT.

Defined in: OMAssertions.h

Return Value

True if the string is valid, false otherwise.

Parameters

string
The string to check for validity.

---

validWideString

bool validWideString(const wchar_t* *string*)

Is the given wchar_t string valid ? Use **validWideString** in expressions passed to the assertion macros PRECONDITION, POSTCONDITION and ASSERT.

Defined in: OMAssertions.h

Return Value

True if the wchar_t string is valid, false otherwise.

Parameters

string
The wchar_t string to check for validity.

---

wfopen

FILE* wfopen(const wchar_t* *fileName*, const wchar_t* *mode*)

Open a named file. Just like ANSI fopen() except for wchar_t* file names and modes.

Defined in: OMUtilities.h

Return Value

An ANSI FILE*

Parameters

fileName
: The file name.

mode
: The mode.

OMFile::OMAccessMode

```
enum OMFile {

        readOnlyMode,

        writeOnlyMode,

        modifyMode,

};
```

File access modes.

Defined in: OMFile.h

Members

readOnlyMode
: The file may be read but may not be written.

writeOnlyMode
: The file may be written but may not be read.

modifyMode
: The file may be read and/or written.

OMFile::OMFileEncoding

```
enum OMFile {

        MSSBinaryEncoding,

        KLVBinaryEncoding,

        XMLTextEncoding,

};
```

Supported file encodings.

Defined in: OMFile.h

Members

MSSBinaryEncoding
: Microsoft Structured Storage (binary).

KLVBinaryEncoding
: SMPTE KLV (binary).

XMLTextEncoding
: XML (text).

OMFile::OMLoadMode

```
enum OMFile {

        eagerLoad,

        lazyLoad,

};
```

Lazy loading modes (degrees of indolence).

Defined in: OMFile.h

Members

eagerLoad
: Objects are loaded when the root object is loaded.

lazyLoad
: Objects are loaded on demand.

OMIteratorPosition

```
enum OMIteratorPosition {

        OMBefore,

        OMAfter,

};
```

Iterator initial position

Defined in: OMContainerIterator.h

Members

OMBefore

Position the iterator before the first element

OMAfter

Position the iterator after the last element