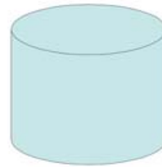
Open Service Interface Definition

Repository



Document Release: 2.0
OSID 2.0

Summary

The Repository Open Service Interface Definition covers storing and retrieving digital content, referred to as Assets, as well as information about the Assets. Assets, examples of which include: documents, course material, assessment item, images, video, audio, etc, reside in Repositories which have names and descriptions and which support a specific set of Asset Types. Repositories are themselves organized by the RepositoryManager that keeps track of repositories and supports certain operations such as searching for Assets across repositories. Associated with each Asset Type is a RecordStructure that defines the format of information comprising the Asset or information describing the Asset. An Asset can have content as well as Records, which are data in the format defined by the Asset's RecordStructure. Assets may contain other Assets.

Service Definition

An examination of an Open Service Interface Definition (OSID) begins with the Manager. All Managers[1] provide the way to create the objects that implement the principal interfaces in the Service.

org.osid.repository.RepositoryManager

The **RepositoryManager** provides for the creation of Repositories. This can be thought of as a catalog in that the Repositories can be created as well as deleted and the set of Repositories can be returned. The **Repository** is the principal interface in the Service. Other objects such as Assets and Records are man-

[1] org.osid.OsidManager defines the interface extended by the Managers in each OSID. Here we will be discussing org.osid.repository.RepositoryManager. Refer to OsidManager for more information.

Copyright © 2004 Massachusetts Institute of Technology

aged elsewhere. The RepositoryManager includes the method **createRepository()** which accepts a name, a description and a repositoryType which are set in the created object. This method returns a new object that implements the Repository interface.[2] The created object is given a unique identifier[3] by the Manager and stored. Any Repository created in this fashion can also be removed using the Manager's **deleteRepository()** method. The **getRepositories()** method returns the set of Repositories being managed.[4]

The RepositoryManager also provides methods for getting Assets. Specifically, the **getAsset()** method accepts a unique Asset identifier and returns the Asset. *This service is provided by the Manager rather than by a Repository because the Asset can reside in any Repository.* The OSID provides for implementations that will support versioning by date. The method **getAssetDates**() returns the dates for a given Asset. If there is no versioning support, this method might throw a not-implemented exception or return a single value in the org.osid.shared.LongValueIterator. Another form of **getAsset**() accepts a date along with the identifier. This allows for date-based, Asset versioning associated with a single unique identifier although the details are left to the implementation.

In addition to returning a single Asset by identifier, the Manager can return a set of Assets that satisfy search criteria. The **getAssets**() method accepts which Repositories are to be searched as well as the search criteria, a SearchType, and the searchProperties. The search criteria are defined simply as something Serializable. The criteria are purposefully general to allow the maximum flexibility across systems and for the future. This approach was preferred to offering a defined set of searches. Note that each Repository supports a specific set of SearchTypes[5]. Another method in the Manager is **copyAsset**() which accepts an Asset identifier and a Repository identifier. The method creates a copy of the identified Asset in the specified Repository and returns the identifier of the newly created Asset.

RepositoryManager Method Summary

Id	copyAsset(Repository Repository, Id assetId)
Repository	createRepository(String displayName, String description, Type repositoryType)
void	deleteRepository(Id RepositoryId)
Asset	getAsset(Id assetId)
Asset	getAssetByDate(Id assetId, long date)
org.osid.shared.LongValueIterator	getAssetDates(Id assetId)
AssetIterator	getAssetsBySearch(Repository[] repositories, Serializable searchCriteria, org.osid.shared.Type searchType, org.osid.shared.Properties searchProperties)
RepositoryIterator	getRepositories()
RepositoryIterator	getRepositoriesByType(Type repositoryType)
Repository	getRepository(Id repositoryId)
TypeIterator	getRepositoryTypes()

[2] Refer to the discussion of OsidManager for information on create methods and their use in implementation substitution.

[3] Refer to the discussion of Unique Identifiers in "Ids" and "OSID Shared".

[4] Objects are returned through a iterator, in this particular case a RepositoryIterator. Iterators return objects in sequence. Refer to the general discussion of Iterators for more information.

[5] Refer to the general discussion of Types in "Types" and "Osid Shared".

Copyright © 2004 Massachusetts Institute of Technology

org.osid.repository.Asset

An Asset's definition includes several methods for managing Records in addition to the common ones for display name, description, and id. The management methods are a little more complex than the ones found in the Repository.

To start, we have the **getDisplayName**(), **updateDisplayName**(), **getDescription**(), **updateDescription**(), and **getId**(). These methods are similar to those found in many OSIDs. Note that the name, description are set by the implementation to the values passed into the Repository's **createAsset(),** while the identifier is generated by the implementation. The **getAssetType**() method returns the type set during the createAsset() and it is immutable.

Assets can contain other Assets. There is no other, separate collection within the OSID. The **addAsset**() method adds an Asset, by Id, to an Asset. The **removeAsset**() method removes an Asset. Adding is not the same as creating in that adding simply is an organizing operation and does not bring a new Asset into existence. Similarly, remove is not delete in that remove does not destroy the Asset, it simply removes the Asset from being organized with another Asset. Assets can be organized in a hierarchy or other arrangement. The **removeAsset**() method provides an *includeChildren* boolean argument so an implementation may support optional cascading deletion.

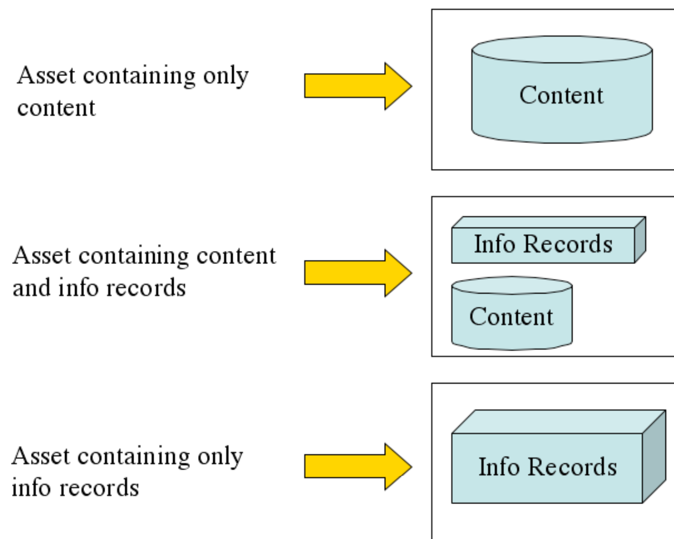
An Asset may have RecordStructures and the **getRecordStructures**() method returns them. The mandatory RecordStructures can be determined by asking the DigtialRepository. There are two methods relating to RecordStructures that require some explanation. The **copyRecordStructure**() method accepts an Asset's identifier and a RecordStructure's identifier; these act as sources. The method adds the source RecordStructure to the current Asset and then copies any of the source Asset's Records that are associated with the RecordStructure. These copies are added to the current Asset. This method is convenient for managing families of Records for similar Assets. A related method, **inheritRecordStructure**(), performs the same operation, except that subsequent changes to the sources are reflected in the current Asset. After this method is called, any subsequent changes to the source Asset are reflected. Specifically, the following are inherited:[6]

- Records of the RecordStructure created from the source Asset
- Records of the RecordStructure deleted from the source Asset
- PartStructures created for the source RecordStructure
- Parts created for the source Records or their Parts
- Parts deleted from the source Records or their Parts
- Updates to Part values

Assets can have content independent of any RecordStructures and Records. This content, returned by **getContent**() and set by **updateContent**() can comprise the Asset in total or this content can be some digital data with the RecordStructures and Records providing a description of the data (metadata), or the Asset can contain only RecordStructures and Records with no other content. The **getContentRecordStructure**() method returns an InfoStrcuture specifically for the Content. This allows both very general support for Content and some information about what the Content actually is.

[6] Note that as with any method, some implementations may not provide support. If that is the case for this method, the method will throw the RepositoryException with the org.osid.OsidException.UNIMPLEMENTED message.

Copyright © 2004 Massachusetts Institute of Technology



There are methods for getting all Records, getting a particular Record, and getting particular Parts or Part's values from those Records. These methods are convenient when the Id of the Record, Part, or Part's PartStructure is known. Otherwise, each structure can be interrogated separately for its content.

Asset Method Summary

void	addAsset(Id assetId)
void	copyRecordStructure(Id assetId, Id RecordStructureId)
Record	createRecord(Id RecordStructureId)
void	deleteRecord(Id RecordId)
AssetIterator	getAssets()
AssetIterator	getAssetsByType(Type assetType)
Type	getAssetType()
Serializable	getContent()
RecordStructure	getContentRecordStructure()
String	getDescription
String	getDisplayName()
long	getEffectiveDate()
long	getExpirationDate()
Id	getId()
Part	getPart(Id partId)
PartIterator	getPartsByPartStructure(Id partStructureId)
Serializable	getPartValue(Id partId)
ObjectIterator	getPartValuesByPartStructure(Id partStructureId)
Record	getRecord(Id RecordId)
RecordIterator	getRecords()
RecordIterator	getRecordsByRecordStructure(Id recordStructureId)

Copyright © 2004 Massachusetts Institute of Technology

RecordIterator	getRecordsByRecordStructureType(Type recordStructureType)
RecordStructureIterator	getRecordStructures()
Id	getRepository()
void	inheritRecordStructure(Id assetId, Id recordStructureId)
void	removeAsset(Id assetId, boolean includeChildren)
void	updateContent(Serializable content)
void	updateDescription(String description)
void	updateDisplayName(String displayName)
void	updateEffectiveDate(long effectiveDate)
void	updateExpirationDate(long expirationDate)

org.osid.repository.Part

A Part contains the data whose structure is defined by a RecordStructure's PartStructure. The getPartStructure() method returns that information. The methods, **createPart**(), **deletePart**(), and **getParts**() provide for managing any Parts in the Part. Parts have an identifier and the **getId**() method returns it.

The getValue() method returns the value of a Part. The updateValue() updates the value of a Part. The value can be any kind of object and of arbitrary complexity.

Part Method Summary

Part	createPart(Id partStructureId, Serializable value)
void	deletePart(Id partId)
String	getDisplayName()
Id	getId()
PartIterator	getParts()
PartStructure	getPartStructure()
Serializable	getValue()
void	updateDisplayName(String displayName)
void	updateValue(Serializable value)

org.osid.repository.PartStructure

A PartStructure has the common methods, **getDisplayName**(), **getDescription**(), and **getId**(). Note that these are all immutable. The implementation defines these objects rather than the OSID providing methods for creating them, etc. This is similar to what is done with the Asset, search, and status types. PartStructures can contain PartStructures. The **getPartStructures**() method returns the PartStructures in the PartStructure.

A PartStructure is associated with a RecordStructure and the **getRecordStructure**() method returns that. A PartStructure can be mandatory or optional; the **isMandatory**() method returns which one is the case. A PartStructure can be repeated or required to be solitary; the **isRepeatable**() method returns which is the case. A PartStructure can be populated automatically, for example in the case of an Asset's creation date. The **isPopulatedByDR**() method returns whether the part is populated automatically or explicitly.

The **validatePartStructure**() method validates a Part given the PartStructure. The OSID does not define the validation technique. If the validation fails, an exception is thrown with an appropriate message about why the Records were satisfactory.

Copyright © 2004 Massachusetts Institute of Technology

PartStructure Method Summary

String	getDescription()
String	getDisplayName()
Id	getId()
PartStructureIterator	getPartStructures()
RecordStructure	getRecordStructure()
Type	getType()
boolean	isMandatory()
boolean	isPopulatedByRepository()
boolean	isRepeatable()
void	updateDisplayName()
boolean	validatePart(Part part)

org.osid.repository.Record

A Record contains the data whose structure is defined by a RecordStructure. The **getRecordStructure**() method returns that information. The methods, **createPart**(), **deletePart**(), and **getParts**() provide for the management of the Parts in the record. Each Part is associated with a part in the RecordStructure. Note that an implementation can create both a Record and all its Parts directly. The reason for offering the additional ability to do this manually is to support repeatable Parts.

Records have an identifier and the **getId**() method returns it. An Asset may permit multiple Records of the same RecordStructure. The **isMultivalued**() method returns which is the case.

Record Method Summary

Part	createPart(Id partStructureId, Serializable value)
void	deletePart(Id partId)
String	getDisplayName()
Id	getId()
PartIterator	getParts()
RecordStructure	getRecordStructure()
void	updateDisplayName(String displayName)

org.osid.repository.RecordStructure

A RecordStructure has the common methods, **getDisplayName**(), **getDescription**(), and **getId**(). Note thatthese are all immutable. The OSID does not provide methods for creating RecordStructures but rather leaves the responsibility for defining them solely to the implementation. This is similar to what is done with the Asset, Search, and Status Types.

RecordStructures contain PartStructures, which in turn can contain PartStructures. The **getPartStructures**() method returns the PartStructures in the RecordStructure.

The method **getFormat**() returns a string that describes the format for this RecordStructure. An example of a format is XML. The OSID does not specify any format strings in particular. Which formats to use and how to interpret them is a convention established between the implementation and any applications. The method **getSchema**() returns a string that describes the schema for this RecordStructure. An example of

Copyright © 2004 Massachusetts Institute of Technology

a schema is Dublin Core. As with format, no specific strings are defined by the OSIDand are a convention of the implementation and any applications that use it.

The **validateRecord**() method validates a Record for the RecordStructure. The validation technique is not defined by the OSID. If the validation fails, an exception is thrown with an appropriate message about why the Records were satisfactory.

RecordStructure Method Summary

String	getDescription()
String	getDisplayName()
String	getFormat()
Id	getId()
PartStructureIterator	getPartStructures()
String	getSchema()
Type	getType()
boolean	isRepeatable()
PartStructureIterator	getPartStructures()
void	updateDisplayName()
boolean	validateRecord(Record Record)

org.osid.repository.Repository

Repository manages Assets of various types and information about the Assets. Assets are created, persisted, and validated by the repository OSID implementation. Similar to the RepositoryManager, Repository includes a **createAsset**() method which accepts a name, description, and Asset type. Note that the Asset type is immutable and that associated with each Asset type may be mandatory RecordStructures. There is only one Asset Type for an Asset. This approach makes validation wieldy. The format and schema attributes as well as the RecordStructures provide ample areas for characterizing Assets. The create method returns a unique identifier. The Repository includes the method **deleteAsset**() as well as two forms of **getAssets**(), one for all Assets and one for all Assets of a specified Asset Type. There are variations of **getAssets**() similar to the ones in RepositoryManager for getting Assets by Id and date, that match criteria, etc. There is also a similar method to copy an Asset.

In addition to managing Assets, the Repository has methods to return information about the Repository. The method **getDisplayName**() returns the name to be used when, for example, listing or identifying this Repository in a user interface. In contrast with the unique Id returned by the **getId**() method, this display name can be changed after creation and is not guaranteed or meant to be unique. The **updateDisplayName**() method changes this name. The use of "update" is significant in that "set" is for instance attribute changes while "update" is for changes the implementation must persist.[7] Matching those for name are the methods **getDescription**() and **updateDescription**().

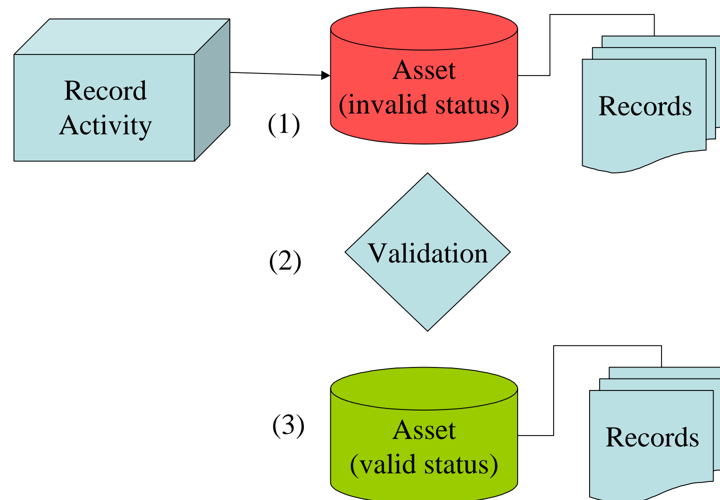
How to Describe a Repository
There may be information about a Repository or its contents that is not suitable for representation through the Description. One strategy is to make that information into an Asset of a specific AssetType. This information can then be easy retrieved and presented. Some examples of summary data are the number of Assets in the Repository or the usage rules. Another example is that in place of inspecting the Repository for the RecordStructures contained, that information could be maintained in this special Asset.

[7] Refer to the general discussion of Update vs. Assign methods in "Serialization".

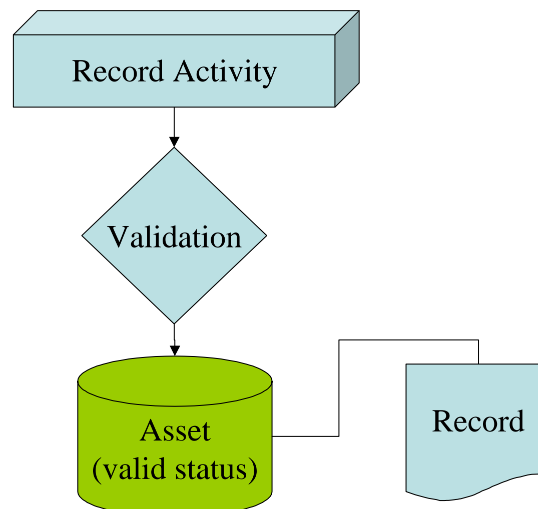
Copyright © 2004 Massachusetts Institute of Technology

Repositories have a set of Types associated with them. The set of unique Asset Types supported by the Repository are returned with the method **getAssetTypes**(). There are also Search Types, returned by the **getSearchTypes**() method, that make clear what kinds of searches are support. The RepositoryManager contains a getAssets() method that performs any search. Assets can have a status and the **get-StatusTypes**() method returns the set of supported kinds of status. The **getStatus()** method returns the status for a particular Asset. These types are not updated programmatically through the Repository OSID. It is the responsibility of the implementation to populate the iterators returned by these methods.

A note on validation: When initially created, an Asset has an immutable Asset type and unique identifier and its validation status is false. In this state, all methods can be called, but integrity checks are not enforced. When the Asset and its Records are ready to be validated, the **validateAsset**() method checks the Asset and sets the validation status.



When working with a valid Asset, all methods include integrity checks and an exception is thrown if the activity would result in an inappropriate state.

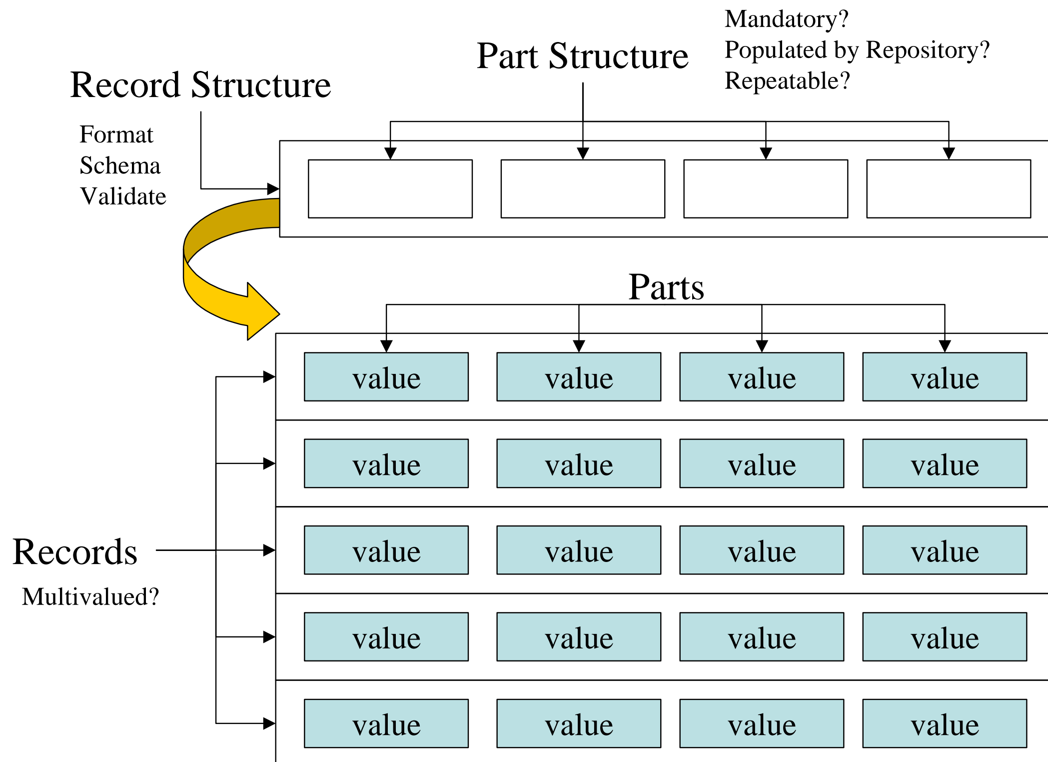


Optionally, the **invalidateAsset**() method can be called to release the requirement for integrity checks, but the Asset will not become valid again until **validateAsset**() is called and all the Records in the Asset are checked.

Repositories include RecordStructures that characterize data. The values for a given Asset's Record-Structure are stored in a Record. Info structures can contain sub-elements that are referred to as Part-Structures. The structure defined in the RecordStructure and its PartStructures are used for any Records for the Asset. Info records have Parts whose structure is based on PartStructures.

Copyright © 2004 Massachusetts Institute of Technology

The **getRecordStructures**() method returns the RecordStructures that a Repository supports. A variation, **getMandatoryRecordStructures**() returns the RecordStructures required for a specified Asset Type. When a Record is created, it is based on a RecordStructure and when a Part is created, it is based on a PartStructure. Each RecordStructure can validate a given Record.



Repository Method Summary

Id	copyAsset(Asset asset)
Asset	createAsset(String displayName, String description, Type assetType)
void	deleteAsset(Id assetId)
Asset	getAsset(Id assetId)
Asset	getAssetByDate(Id assetId, Calendar date)
org.osid.shared.LongValueIterator	getAssetDates(Id assetId)
AssetIterator	getAssets()
AssetIterator	getAssetsBySearch(Serializable searchCriteria, org.osid.shared.Type searchType, org.osid.shared.Properties searchProperties)
AssetIterator	getAssetsByType(Type assetType)
TypeIterator	getAssetTypes()
String	getDescription()
String	getDisplayName()
Id	getId()
RecordStructureIterator	getMandatoryRecordStructures(Type assetType)
PropertiesIterator	getProperties()

Copyright © 2004 Massachusetts Institute of Technology

Properties	getPropertiesByType(Type propertiesType)
TypeIterator	getPropertyTypes()
RecordStructureIterator	getRecordStructures()
RecordStructureIterator	getRecordStructuresByType(Type recordStructureType)
TypeIterator	getSearchTypes()
Type	getStatus(Id assetId)
TypeIterator	getStatusTypes()
Type	getType()
void	invalidateAsset(Id assetId)
void	supportsUpdate()
void	supportsVersioning()
void	updateDescription(String description)
void	updateDisplayName(String displayName)
boolean	validateAsset(Id assetId)

org.osid.repository.*Object* Iterators

Iterators return a set of elements of a specific type, one at a time. The purpose of all Iterators is to offer a way for OSID methods to return multiple values of a common type and not use an array. Returning an array may not be appropriate if the number of values returned is large or is fetched remotely. Iterators do not allow access to values by index, rather you must access values in sequence. Similarly, there is no way to go backwards through the sequence unless you place the values in a data structure, such as an array, that allows for access by index.

All iterators contain two methods. The **hasNext*<Object type>***() method returns true if there are more values of the iterator type available; false otherwise. The **next*<Object type>*** () method returns the next element in the sequence. Note that in many cases, the order of elements is not guaranteed. The following iterators are included in this OSID:

RepositoryIterator

AssetIterator

RecordStructureIterator

PartStructureIterator

RecordIterator

PartIterator

Type iterators, such as those used for Asset, Search, and Status Types are defined in osid.shared.TypeIterator.

org.osid.repository.RepositoryException

The OSIDs make use of Exceptions as a mechanism for responding to error or unusual conditions. All methods in the Repository OSID throw a **RepositoryException**. The Exception contains a message that is a String. The following message Strings are defined in RepositoryException:

- ALREADY_INHERITING_STRUCTURE
- CANNOT_COPY_OR_INHERIT_SELF
- EFFECTIVE_PRECEDE_EXPIRATION

Copyright © 2004 Massachusetts Institute of Technology

- NO_OBJECT_WITH_THIS_DATE
- UNKNOWN_REPOSITORY

The service definitions make use of Exceptions as a mechanism for responding to error or unusual conditions. Exceptional circumstances are signaled by the implementation throwing an exception, which is **RepositoryException** in the case of the Repository OSID. The Exception contains a message that is a String. Note that other kinds of Exception constructors are not used as all do or can devolve to a String. All methods of all interfaces of all OSIDs throw a subclass of osid.OsidException. This requires the caller of any implementation method handle the Exception.

If a method performs an operation without incident, an object or primitive may be returned, but in most cases, methods do not return error codes or a success or failure boolean. For example, a method that deletes an object with a particular identifier, would throw an Exception if the identifier were unknown; the method would not return, for example, false.

Copyright © 2004 Massachusetts Institute of Technology