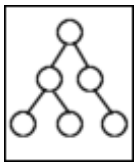
Open Service Interface Definition

Hierarchy



Document Release: 2.0
OSID Release: 2.0

Summary

The Hierarchy Open Service Interface Definition (OSID) provides a means of creating and traversing hierarchical structures of various types. These types include trees, forests, directed graphs with multiple parents, and directed cyclic graphs.

Service Definition

An examination of an Open Service Interface Definition (OSID) usually begins with the Manager. All Managers[1] provide the way to create the objects that implement the principal interfaces in the Service. Before discussing the Manager in detail, we will review the intended role of the Hierarchy Service overall.

Many collections of data used throughout software systems are organized in hierarchies. Examples of these are: management organizations, file systems, course structures, etc. The Hierarchy OSID creates a common tool for viewing and maintaining this type of structure. While these data collections have rich structures themselves, the Hierarchy OSID is concerned only with the hierarchical relationship among nodes and not with the data the nodes represent. The Hierarchy OSID doesn't contain a hierarchy's data, only its structure.

The hierarchy service can maintain many hierarchies; each of these is assigned a unique Id by the Hierarchy service, so that the application can store a permanent reference to it. A hierarchy is made up of nodes. Nodes can have parents (nodes) and children (nodes). Nodes are meant to represent external objects. They have basic data of Id, Name, and Description. The Node Id is assigned by the calling sys-

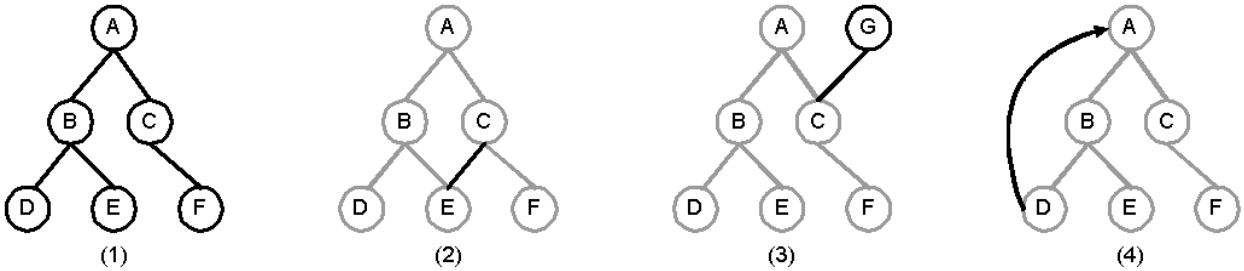
[1] org.osid.OsidManager defines the interface extended by the Managers in each OSID. Here we will be discussing org.osid.hierarchy.HierarchyManager. Refer to OsidManager for more information.

tem. (In other OSIDs the unique Ids are generated internally by the service, instead of being assigned by the calling application.)

The Hierarchy OSID can manipulate a selection of hierarchy types that are suited to these typical data collections. The simplest of these is a tree. A tree is a collection of nodes, each with a single parent[2]. A simple tree has a single parentless node called the root node. See diagram 1 below. File systems are well-known examples of trees. The Hierarchy OSID can manage one or more trees. An example of a hierarchy containing multiple trees is a file system with multiple mounted devices.

A second hierarchy type looks like a tree except it supports nodes having multiple parents. Such a hierarchy is no longer a tree, but a graph, since there are multiple paths from the root to some nodes. See diagram 2 below. The Hierarchy OSID also supports graphs that have multiple roots (diagram 3).

The Hierarchy OSID can also support a cyclic graph whose root node may have a parent (diagram 4).



The Hierarchy has a **traverse()** method that gets information from a hierarchy. Its arguments are the starting node, the traversal direction, and the number of levels to traverse. The traverse call returns an Iterator, which contains the Id and the name of the node as well as the relative distance (level) from the initial node. In the case of an implementation that allows for recursion (where a traversal may return to its starting node), it is expected that the traversal will stop when the starting Node is reached.

Given the variety of backend systems that an implementation may use to support the Hierarchy OSID, not all the methods may be implemented. Methods that are unimplemented due to the limitations of the backend system should throw a HierarchyException with the defined message, UNIMPLEMENTED.

org.osid.hierarchy.HierarchyManager

An installation may have several implementations of HierarchyManager available. Like managers in others O.K.I. services, HierarchyManagers with desired features can be obtained through OsidLoader. Once a HierarchyManager is available, it can retrieve all available Hierarchies using the method **getHierarchies()** or retrieve a specific Hierarchy by Id using the method **getHierarchy()**. HierarchyException may be thrown when arguments are null, methods have not been implemented, default HierarchyManager does not load, or the HierarchyManager property file does not load.

All implementations of OsidManager provide create, delete, and get methods for the various objects defined in the package. Most managers also include methods for returning Types. We use create methods in place of the new operator. Create method implementations should both instantiate and persist objects. The reason we avoid the new operator is that it makes the name of the implementing package explicit and requires a source code change in order to use a different package name. In combination with OsidLoader, applications developed using managers permit implementation substitution without source code changes[3].

[2] Graph terminology draws indiscriminately from genealogy and forestry, which results in unavoidable mixed metaphors.

[3] Refer to the discussion of OsidLoader in the document titled “ManagerLoader”.

HierarchyManager Method Summary

Hierarchy	createHierarchy(String displayName, Type[] nodeTypes, String description, boolean allowsMultipleParents, boolean allowsRecursion)
void	deleteHierarchy(Id hierarchyId)
HierarchyIterator	getHierarchies()
Hierarchy	getHierarchy(Id hierarchyId)
boolean	supportsMaintenance()

org.osid.hierarchy.Hierarchy

A Hierarchy is a structure comprised of nodes arranged in root, parent, and child form. The Hierarchy can be traversed in several ways to determine the arrangement of nodes. A Hierarchy may allow multiple parents and may allow recursion. The implementation is responsible for ensuring that the integrity of the Hierarchy is always maintained.

If the Hierarchy allows a traversal to return to its starting Node, the method **allowsRecursion()** returns true. If the Hierarchy allows a Node to have more than one parent, the method **allowsMultipleParents()** returns true. You may retrieve all the nodes in a Hierarchy through the method **getAllNodes()**, though the order that the Nodes are returned is undefined. To retrieve Nodes in order use the **traverse()** method.

Hierarchy Method Summary

void	addNodeType(Type type)
boolean	allowsMultipleParents()
boolean	allowsRecursion()
Node	createNode(Id nodeId, Id parentId, Type type, String displayName, String description)
Node	createRootNode(Id nodeId, Type nodeType, String displayName, String description)
void	deleteNode(Id nodeId)
NodeIterator	getAllNodes()
String	getDescription()
String	getDisplayName()
Id	getId()
Node	getNode(Id nodeId)
TypeIterator	getNodeTypes()
NodeIterator	getRootNodes()
void	removeNodeType(Type type)

TraversalInfoIterator	traverse(Id startId, int mode, int direction, int levels)
void	updateDescription(String description)

Field Summary

TRAVERSE_DIRECTION_DOWN	Constant indicating traversal down the Hierarchy, or traversal of children.
TRAVERSE_DIRECTION_UP	Constant indicating traversal up the Hierarchy, or traversal of parents.
TRAVERSE_LEVELS_ALL	Constant indicating no limit on the depth of traversal.
TRAVERSE_MODE_BREADTH_FIRST	Constant indicating breadth-first traversal.
TRAVERSE_MODE_DEPTH_FIRST	Constant indicating depth-first traversal.

osid.hierarchy.HierarchyIterator
osid.hierarchy.NodeIterator
osid.hierarchy.TraversalInfoIterator

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.

HierarchyIterator Method Summary

boolean	hasNextHierarchy()
Hierarchy	nextHierarchy()

NodeIterator Method Summary

boolean	hasNextNode()
Node	nextNode()

TraversalInfoIterator Method Summary

boolean	hasNextTraversalInfo()
TraversalInfo	nextTraversalInfo()

org.osid.hierarchy.Node

A Node is a Hierarchy's representation of an external object organized by the Hierarchy. Nodes do not provide a data store or storage for the object represented. The Hierarchy OSID focuses on the relationships between the objects, not the data management of the objects themselves. Unlike most other OSIDs the caller assigns the Node Id. The Id is the sole means for a Hierarchy to retrieve the data represented by a Node.

A node is considered a leaf if it has no children; it is considered a root if it was created by the **Hierarchy.createRootNode()** method. The methods **isLeaf()** and **isRoot()** test for these conditions. A Node's parents may be retrieved, added, or removed using the methods **addParent()**, **removeParent()**, and **getParents()**. Note that **getParents()** returns a single parent if **Hierarchy.allowsMultipleParents()** returns false, and may return parents if **isRoot()** returns false. The methods **addParent()** and **removeParent()** will return an exception (either UNIMPLEMENTED or SINGLE_PARENT_HIERARCHY) if **allowMultipleParents()** returns false. The method **changeParent()** allows changing a Node's parent without introducing inconsistency into a single parent hierarchy.

Node Method Summary

void	addParent(Id nodeId)
void	changeParent(Id oldParentId, Id newParentId)
NodeIterator	getChildren()
String	getDescription()
String	getDisplayName()
Id	getId()
NodeIterator	getParents()
Type	getType()
boolean	isLeaf()
boolean	isRoot()
void	removeParent(Id parentId)
void	updateDescription(String description)
void	updateDisplayName(String displayName)

org.osid.hierarchy.HierarchyException

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

Exception Message Summary

Constant	Message String

ATTEMPTED_RECURSION	Hierarchy does not allow recursion
HIERARCHY_NOT_EMPTY	Cannot delete a Hierarchy containing Nodes
INCONSISTENT_STATE	Removing node will result in an inconsistent state
NODE_TYPE_IN_USE	Cannot remove NodeType referenced by a Node
NODE_TYPE_NOT_FOUND	NodeType has never been added
SINGLE_PARENT_HIERARCHY	Hierarchy does not allow multiple parents
UNKNOWN_PARENT_NODE	Cannot create Node with unknown parent
UNKNOWN_TRAVERSAL_DIRECTION	Unrecognized traversal direction
UNKNOWN_TRAVERSAL_MODE	Unrecognized traversal mode
UNSUPPORTED_CREATION	Hierarchy does not support allowsMultipleParents is false and allowsRecursion is true

If an implementation uses these messages, consumers of the implementation can easily test and conditionally respond to the Exception. 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 org.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.

HierarchyException Method Summary

HierarchyException has no defined methods.

org.osid.hierarchy.TraversalInfo

The TraversalInfo is the data structure that represents the nodes returned by the **Hierarchy.traverse()** method. A TraversalInfo contains the traversed Node's Id, display name, and level. The level of the Node represented by the Node Id is in relation to the startId of the **Hierarchy.traverse()** method call. The level of the starting node is 0. Children Nodes are represented by positive levels, parent Nodes by negative levels. For example, a traverse of a Hierarchy has level -1 for parents of the Node represented by startId, and a level -2 for grandparents. Similarly, the children of the Node would have level 1, and grandchildren would have level 2.

TraversalInfo Method Summary

String	getDisplayName()
int	getLevel()
Id	getNodeId()