OpenClovis
Software Development Kit (SDK)
Service Description and API Reference for Clovis Object Repository (COR) Service

For OpenClovis SDK Release 2.3 V2.0

Document Revision Date: February 22, 2007

Contents

Chapter 1

Functional Overview

The ASP COR service provides interfaces to access, modify and manage the life cycle of objects. These objects are called Managed Objects as they contain management information about network elements such as a Chassis. They are used to exchange information between system management applications and applications running on network elements. The applications running on a network element use or produce management information. COR objects and metadata associated with these Managed Objects are located in an in-memory object storage.

System Management Applications reside outside the network and use the services of a local management agent to interface with COR. These local agents are referred to as Object Managers (OM).

Applications in a network element that use or produce the management information are referred to as Object Implementers. An ASP Service can act as an Object Implementer for an object of interest.

Object Manager and Object Implementer are roles performed by applications. In the current implementation of COR, COR is unaware of Object Manager and Object Implementer roles and does not enforce any semantics related to these roles. It is assumed that the OM and OI roles are maintained by careful application design.

Currently, COR does not provide support for system management applications to use their own agents while interacting with network elements. Management agents would need to gain exclusive control of a set of managed objects temporarily to perform some change. In the current implementation, it is assumed that multiple Object Managers, if present, co-operate with each other while managing or making changes to objects.

Chapter 2

Service Model

2.1 Usage model

Applications can model their network elements and represent them as COR Information Model. These applications can use MO classes and the containment relationship in COR to interpret the relationship between various network elements.

Management Applications (North-bound) Management application can perform CREATE, DELETE or SET operations on Managed Objects for configuration purposes. Management applications can also subscribe to CREATE, DELETE or MODIFY notifications on Managed Objects. Such Managed Objects reflect the status of associated network elements.

Object Implementor An Object Implementer (OI) implements the configuration supplied by the north bound. An OI can also perform CREATE, DELETE, and MODIFY operations on Managed Objects to reflect its run-time status.

2.2 Functional Description

2.2.1 Managed Object Class Characteristics

2.2.1.1 COR MO-Class and MSO-Class

A MO COR-class is a collection of attributes. Each class has a name and an integral identifier. An MO-class has two services associated with it:

- Provisioning service
- Alarm service

COR groups related attributes of these two services in two different Managed Service Object Class (also called MSO class). Thus, an MO class has two different MSO classes. One corresponding to Provision-able attributes and the other corresponding to Alarm attributes. The MSO class also has a name and a list of attributes.

ASP Alarm service determines the structure of the Alarm MSO class and uses this class to model alarms.

The Provision-able MSO class structure is defined by the application. From the modeling perspective, it is only the PROV MSO class that is of interest to the application. For example, a GigePort class can have a group of provisionable attributes (MTU size) and a group of alarms (such as LOS, LOF and so on). COR considers these two groups as two distinct MSO classes associated with GigePort class. An MSO class is identified by an MO class and its Service ID The Service ID indicates if the MSO is Provisionable or corresponding to Alarm. The enumeration, `ClCorServiceIdT`, specifies the service ID of a class.

2.2.1.2 MO tree and Containment

All Managed Objects in COR are organized in a tree hierarchy called the MO tree. The relationship of an MO to its parent MO in the hierarchy is referred to as its "containment" relationship. An MO can only be contained in MOs belonging to other MO classes. The rules related to permitted containment relations are part of the definition of an MO Class.

The containment relationship allows COR to organize all MOs in a tree and is therefore mandatory from COR's perspective. Containment relationship is of no other significance to COR. However, it can have some special significance to management applications as some OIs can be interested in a group of related objects (in a subtree). For example, a subtree could reflect the containment of the Availability Management Service (AMS) related objects and this containment can have special significance to AMS.

2.2.1.3 Blueprint for the MO Tree

The MO Class definitions and the permitted containment relations in COR are also referred to as the "blueprint" for the MO tree. This blueprint refers to the complete set of object metadata associated with a COR instance. COR permits objects to be created and attached to a tree according to the blueprint. This blueprint can be constructed using the OpenClovis Integrated Development Environment (IDE). The blueprint is exported in the COR XML file and imported by COR when it is initialized for the first time. The blueprint is kept in the COR's persistent storage referred to as COR-DB. COR obtains the blueprint from COR-DB during its subsequent startup.

2.2.2 MO Attributes

2.2.2.1 Attribute Characteristics

Each MO attribute has the following characteristics:

- Name
- Type associated with attribute values
- Default value
- Qualifier
- Sub qualifiers

The qualifiers supported are:

- Config
- Run-time

The sub-qualifiers supported are:

- Cached
- Persistent
- Initialized
- Writable

2.2.2.2 Attribute Types Supported

Integer and String Types

The supported types are signed and unsigned integers of various sizes. Arrays of these integer types are also supported. For integer types, a maximum, minimum, and default value can be specified.

2.2.2.3 Caching and Persistence of COR objects

All COR metadata and objects are persisted in the COR-DB. However, the attribute values are not always persisted. This control can be exercised at the object attribute level. Persisted object attributes are restored in a COR instance automatically after a GMS-Cluster reboot.

COR stores the values of attribute in the memory. This is referred to as caching. Caching results in better read access times for cached attributes. Caching of data is not required when the cached data can quickly become stale. Controls are provided to prevent caching of this type of data. These are explained in subsequent sections.

2.2.2.4 Attribute Qualifiers

Every attribute must be qualified as a CONFIG, run-time or KEY. These qualifiers are mutually exclusive.

Configuration Attributes

A configuration attribute is always persisted and cached. Configuration attributes contain data provided by the Object Managers. They are read-only from the perspective of the Object Implementer.

Run-time Attributes

A run-time attribute is not persisted or cached by default. The following sub-qualifiers (either singly or in any combination) can be associated with a run-time attribute:

- Persistent - This sub-qualifier for run-time attributes indicates that the attribute must be made persistent by the COR Service.
- Cached - This sub-qualifier for run-time attribute indicates that the attribute must be cached by the COR service.

Run-time attributes are used for data provided by Object Implementers. They are read only from an Object Manager's perspective. A run-time attribute that is cached is updated by the Object Implementer when the value changes. When a read request is made on such a run-time attribute, COR reads the value from the cache. A read request on a run-time

attribute that is not cached triggers a synchronous request to the Object Implementer that returns with the value of the attribute.
For examples:
`run-time + cached ->` Attribute showing Application log stream file name implemented by the Log service.
`run-time + cached + persistent ->` Administrative state of a service unit implemented by AMF.
`run-time + multi-valued ->` List of Service Instances currently assigned to a Service Unit, implemented by AMF.

2.2.3 Object Addressing

2.2.3.1 MOID

Every object in the COR MO tree has a unique ID referred to as the MOID (Managed Object ID). Every object has a `relative_ID` that is formed from the tuple, `class_ID` and `instance_ID`, where:

- class_ID - uniquely identifies the MO class of which this object is an instance and
- instance_ID - uniquely identifies this instance of the object from other instances having the same parent.

The MOID of an object is formed from the tuple, MOID of the parent and `relative_ID` of the object. For example, an AMF Service Unit instance can have the following MOID:
`COR_ROOT_CLASS_ID:0/APPLICATION_CLASS_ID:0/SERVICE_GROUP_CLASS_-ID:1/SERVICE_UNITCLASS_ID:3`

COR provides APIs to manage the life cycle of an object and to read or modify its attributes. The user is required to know the MOID of the object or its parents.

A MOID uniquely identifies an object in COR tree. However, a variant of the MOID is also used to specify ranges of objects. These are called wild card MOIDs. A wild card `class_ID` can represent any class and a wild card `instance_ID` can represent any instance. The wild card MOID is used for OI subscription and for specifying an MO instance search criteria. Following are some of the examples of a wild card MOID:

- `COR_ROOT_CLASS_ID:0/APPLICATION_CLASS_ID:0/SERVICE_GROUP_CLASS_-ID:1/SERVICE_UNITCLASS_ID:*`

  Specifies all the MO instances of `SERVICE_UNITCLASS_ID` class under the hierarchy.

  `COR_ROOT_CLASS_ID:0/APPLICATION_CLASS_ID:0/SERVICE_GROUP_CLASS_-ID:1`

- `COR_ROOT_CLASS_ID:0/APPLICATION_CLASS_ID:0/SERVICE_GROUP_CLASS_-ID:1/*:*`

  Specifies all the MO instance of any class under the hierarchy.

  `COR_ROOT_CLASS_ID:0/APPLICATION_CLASS_ID:0/SERVICE_GROUP_CLASS_-ID:1`

- `COR_ROOT_CLASS_ID:0/APPLICATION_CLASS_ID:0/SERVICE_GROUP_CLASS_-ID:*/*:*`

  Specifies all the MO instance of any class under the subtree `COR_ROOT_CLASS_-ID:0/APPLICATION_CLASS_ID:0/SERVICE_GROUP_CLASS_ID:*`.

The subtree specified by `COR_ROOT_CLASS_ID:0/APPLICATION_CLASS_-ID:0/SERVICE_GROUP_CLASS_ID:*` covers all the MO instances

of `SERVICE_GROUP_CLASS_ID under COR_ROOT_CLASS_ID:0/APPLICATION_CLASS_ID:0/`

2.2.4 Object Management Interfaces

OI and Primary OI

An Object Implementer performs several distinct operations in relation to changes in objects that include:

- Permits or denies an object creation or deletion request.
- Permits or denies changes to a configuration attribute value.
- Implements a change in a configuration attribute when the change has occurred in COR.
- Asynchronously updates a cached runtime attribute in COR as and when the related variable changes internally.
- Synchronously provides the value of a (non cached) run-time attribute when requested by COR.

Every object can have one or more object implementers. While multiple OIs can perform the first three activities, there can be only one OI that is allowed to perform the last two activities.

OI Registration

A component can act as an OI for an MO as specified in a wild card MOID. For example,

- An MO Instance specified through its MoID. For example, the MoID can be

  COR_ROOT_CLASS_ID:0APPLICATION_CLASS_ID:0SERVICE_GROUP_CLASS_-ID:1SERVICE_UNITCLASS_ID:1

- Instances of a particular MO Class hanging from a particular hierarchy.

  COR_ROOT_CLASS_ID:0APPLICATION_CLASS_ID:0SERVICE_GROUP_CLASS_-ID:1SERVICE_UNITCLASS_ID:*

- A complete subtree. A Component can specify itself as the OI for the wild card

  COR_ROOT_CLASS_ID:0APPLICATION_CLASS_ID:0SERVICE_GROUP_CLASS_-ID:*SERVICE_UNITCLASS_ID:*

As a component can act as an OI for multiple MO instances corresponding to different MO classes, COR provides a mechanism to associate an OI callback APIs for each MO class.

The OI and MO association is performed during modeling. The following table provides a list of OI callback APIs provided by COR.

No.	OI Callback Function	Description
1	Constructor	COR invokes this function when an MO is required to be created. An application can embed its custom logic to implement MO creation. Currently, MO creation/deletion does not have a validate callback function.
2	Destructor	COR invokes this callback when an MO is required to be deleted. An application can embed its custom logic to implement MO deletion. Currently, MO creation/deletion does not have a validate callback function.
3	Validate	COR invokes this callback to validate the attribute that is being SET. The callback either permits or denies the application containing this attribute. The semantics of validation is specific to the OI. An OI can acquire resources (such as memory) required to APPLY this attribute to ensure that the APPLY operation is successful. This pre-acquisition reduces the possibility of failure of APPLY operation.
4	Rollback	This callback is invoked when the validate operation fails on this attribute or validate fails on another attribute that is part of the transaction. This callback allows the OI to free any pre-allocated resources, acquired in the validate phase. The rollback API is called if any other operation that is part of this transaction fails to validate.
5	Apply	This function applies the attribute to the resource.

COR Session Capability

The COR session capability provides a mechanism to execute a group of jobs (CREATE, DELETE and MODIFY operations on a MO) in an efficient manner by minimizing the number of RMD calls between the COR client, COR server, and Object Implementer. The COR session capability provides a mechanism to execute CREATE, SET, and DELETE operations on a group of MOs with all-or-none semantics as described in this section.

The COR server performs a basic validation on these jobs. If one of the job validation fails, the session becomes invalid and no operations are performed.

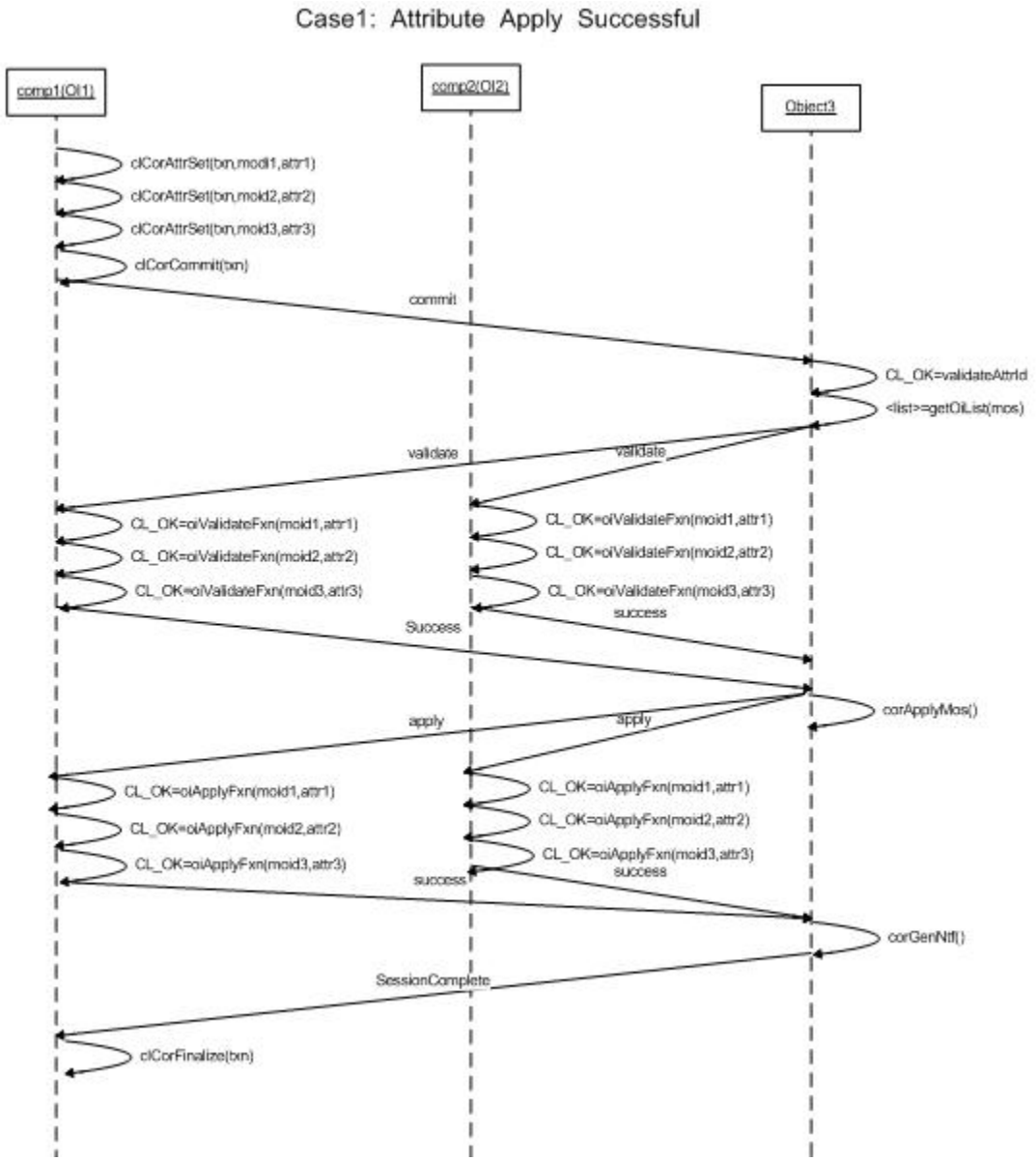
COR determines the OI for each participating MO, and invokes the validate callback of the OI. If the OIs successfully validate their MO operation, COR invokes the apply APIality of the OI. COR also internally updates its database with these changes on the MOs. On successful completion, COR sends a notification for the changed MOs.

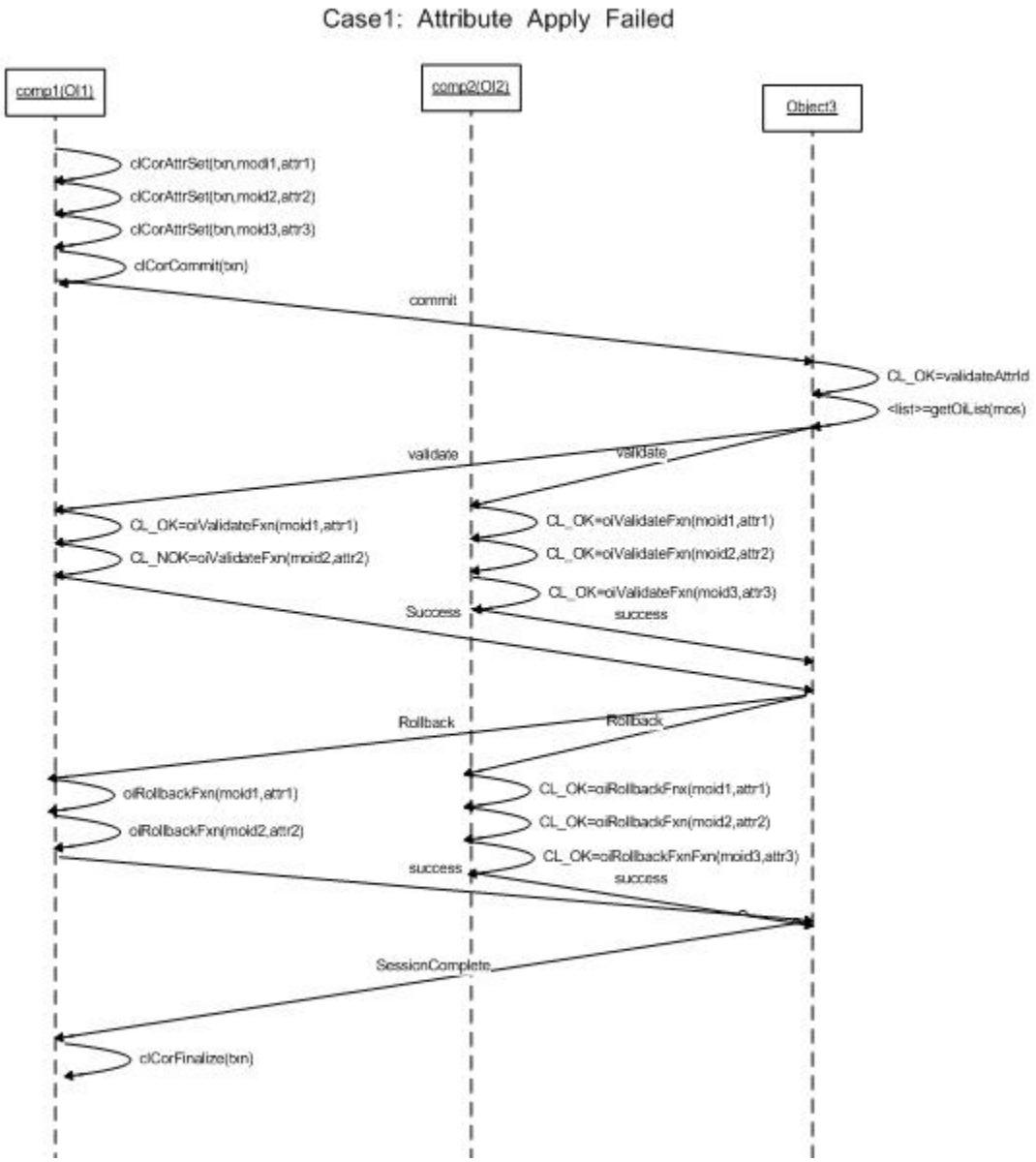
If an OI fails validation, COR does not proceed with the apply. COR determines the set of OIs that have completed validation and calls the rollback APIs.

The following sequence diagrams describe the control flow between COR-client, COR-server, and OIs for the following two cases.

- A session is initiated by *comp1*. In the session *<mod1,attr1>* ,*<moid2,attr2>* and *<moid3,attr3>* are getting updated.
- Both *comp1* and *comp2* are OIs for *<moid1>* ,*<moid2>*, and *<moid2>*.
- All OIs have the same callback API name. The name of the validate, rollback, and apply callbacks are `oiValidateFxn`, `oiRollbackFxn`, and `oiApplyFxn`.

- Case 1:
  - Both OIs validate *<moid1,attr1>*, *<moid2,attr2>*, and *<moid3,attr3>* successfuly.
  - Thus, the apply of these attributes is successful.
- Case2:
  - *comp1* successfuly validates *<moid1,attr1> ,<moid2,attr2>*, and *<moid3,attr3>*.
  - *comp2* successfuly validates *<moid1,attr1>* and fails validation in *<moid2,attr2>*. A rollback can now be called.
  - Rollback is called for *comp1* for the attributes: *<mod1,attr1>*, *<moid2,attr2>*, and *<moid3, attr3>*.
  - Rollback is called for *comp2* for the attributes: *<moid1,attr1>* and *<moid2,attr2>*.
  - No attributes are applied.


Case1: Attribute Apply Successful
comp1(OI1)
comp2(OI2)
Object3
clCorAttrSet(txn,modl1,attr1)
clCorAttrSet(txn,mokd2,attr2)
clCorAttrSet(txn,mokd3,attr3)
clCorCommit(txn)
commit
CL_OK=validateAttrId
<list>=getOiList(mos)
validate
validate
CL_OK=oiValidateFxn(mokd1,attr1)
CL_OK=oiValidateFxn(mokd2,attr2)
CL_OK=oiValidateFxn(mokd3,attr3)
CL_OK=oiValidateFxn(mokd1,attr1)
CL_OK=oiValidateFxn(mokd2,attr2)
CL_OK=oiValidateFxn(mokd3,attr3)
success
Success
corApplyMos()
apply
apply
CL_OK=oiApplyFxn(mokd1,attr1)
CL_OK=oiApplyFxn(mokd2,attr2)
CL_OK=oiApplyFxn(mokd3,attr3)
CL_OK=oiApplyFxn(mokd1,attr1)
CL_OK=oiApplyFxn(mokd2,attr2)
CL_OK=oiApplyFxn(mokd3,attr3)
success
success
corGenNtf()
SessionComplete
clCorFinalize(txn)



COR Bundle Capability

The COR bundle capability provides a mechanism to execute groups of attribute reads in an efficient manner by minimizing the number of RMD calls between COR client, COR server, and Object Implementer.

A bundle is non-transactional in nature. A non-transactional bundle can contain attributes that are not successfully read.

The bundle semantics of executing jobs is performed in four phases by the application:

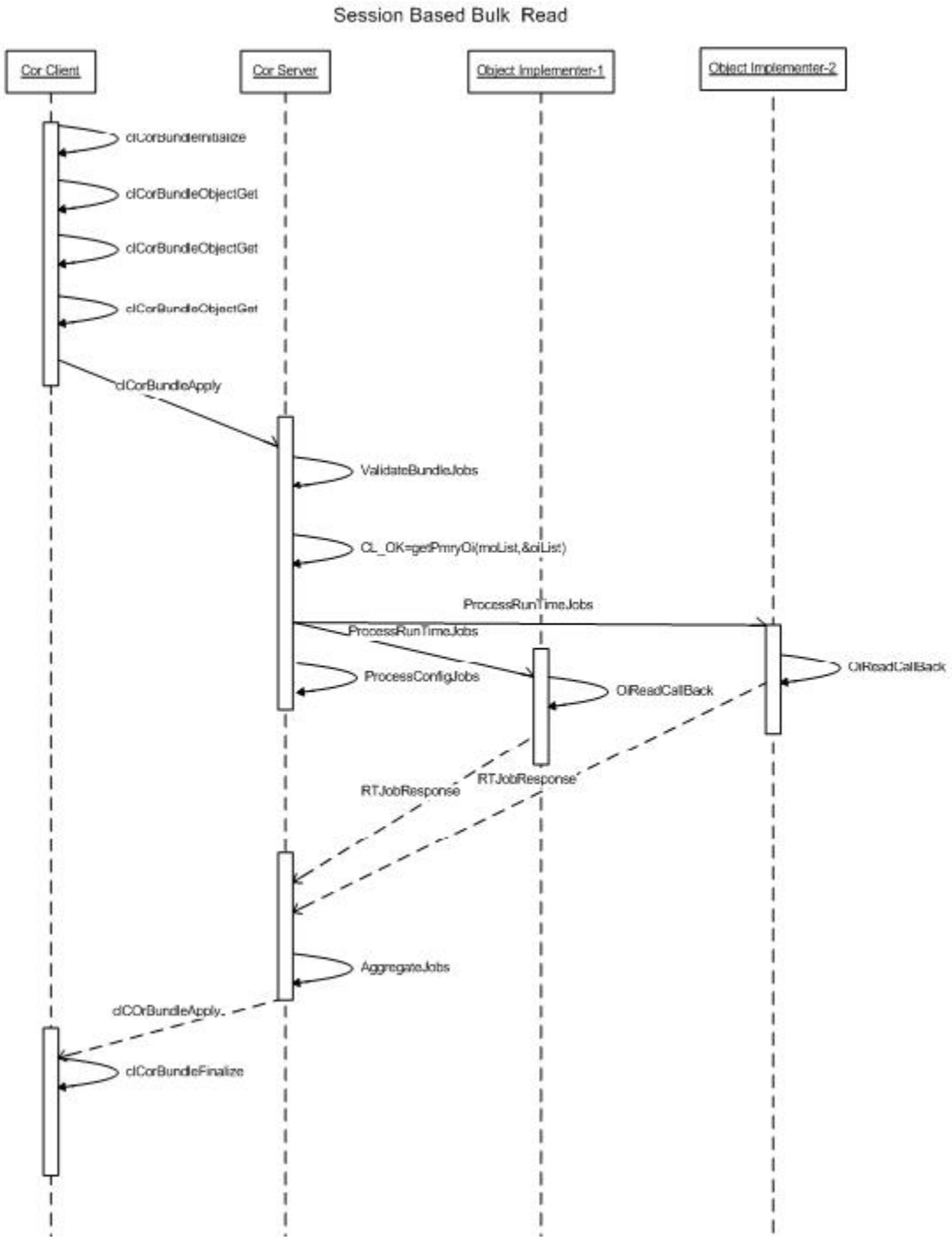
1. Bundle Creation - A bundle is empty with no jobs associated when it is created. A bundle handle that identifies this bundle is returned.
2. Job Population - In the next phase, jobs are added to the bundle. A job corresponds to an MO and list of attributes that needs to be read. Multiple jobs consisting of different MOs

can be a part of a bundle. These jobs are queued at the COR client. Every job has a status and a buffer descriptor. The buffer descriptor contains the value of the attribute to be `set` or `get`. The success or failure of jobs execution is reflected in the status.

3. Bundle Apply - The application performs the APPLY operation synchronously when the population phase is completed. The bundle is submitted to the COR server that reads the value from the database or contacts the OI to obtain the value. The values are streamed back to the COR client at the end of this call.

4. Bundle Finalize - This frees up the resources allocated in bundle operation. The following sequence diagram explains the control flow between COR client, COR server and OIs for the given case.

    - A bundle operation is initiated by comp1. The bundle contains jobs: <mod1, attr1>,<moid2,attr1>, and <moid2,attr2>. <moid1, attr1> , <moid2,attr1> are run-time attributes.
    - These jobs are submitted by comp1.
    - COR determines the PrimaryOI for moid1 and moid2 as <CompOI1> and <CompOI2>.
    - COR obtains the values the run-time attributes of CompOI1 and CompOI2 for <moid1,attr1> and <moid2,attr1>.
    - COR obtains the Config attribute value from its database.
    - COR returns back the value to comp1.



COR Search Capability

COR provides the capability to retrieve objects matching a particular criterion. This criterion is called the filter and the search is termed as object-walk based search. Following are the parameters for this search:

- Root - The root from where search needs to commence. The root is specified as MOID.
- Containment Tree - The subtree on which the search needs to be performed. This subtree

is specified using wild-card MOID.

- Callback function - This is executed for each matched MO instance.

COR provides a mechanism to filter objects based on the value of their attributes. This search is called attribute-walk search. This search is used in conjunction with object-walk search. Object-walk returns all the MO instances that match a particular search criterion. Each of these instance can be filtered based on a particular attribute and its value. Following are the parameters for this search:

- MO - The MO tree on which the search is required to be performed.
- Attribute and its value - This is required to be compared.
- Callback function - This is required to be called when the attribute comparison is successful.

COR Event Generation and Subscription

COR generates an event for every object creation/deletion/set operation on the `COR_EVT_CHANNEL`. Subscribing applications can specify the following filters to narrow down their event of interest.

- MoID: Specifies the set of objects which is of interest to the application. The MoID can be a wild card MoID.
- Operation: Specifies the operation(s) that is of interest to the application. An operation can be a combination of Object-create, Object-delete, and Object-set activities.
- AttributeId: This specifies the attribute of interest. An application callback is executed when an event matching the filter is detected.

Chapter 3

Service APIs

3.1 Error Definitions

- # define CL_COR_ERR_BUFFER_OVERRUN (CL_COR_SET_RC(CL_ERR_BUFFER_OVERRUN))
- # define CL_COR_ERR_BUNDLE_APPLY_FAILURE (CL_COR_SET_RC(CL_COR_BUNDLE_ERR_MAX + 3))
- # define CL_COR_ERR_BUNDLE_FINALIZE (CL_COR_SET_RC(CL_COR_BUNDLE_ERR_MAX + 6))
- # define CL_COR_ERR_BUNDLE_IN_EXECUTION (CL_COR_SET_RC(CL_COR_BUNDLE_ERR_MAX + 8))
- # define CL_COR_ERR_BUNDLE_INIT_FAILURE (CL_COR_SET_RC(CL_COR_BUNDLE_ERR_MAX + 2))
- # define CL_COR_ERR_BUNDLE_INVALID_TYPE (CL_COR_SET_RC(CL_COR_BUNDLE_ERR_MAX + 9))
- # define CL_COR_ERR_BUNDLE_TIMED_OUT (CL_COR_SET_RC(CL_COR_BUNDLE_ERR_MAX + 7))
- # define CL_COR_ERR_CLASS_ATTR_INVALID_INDEX (CL_COR_SET_RC(CL_ERR_COMMON_MAX + 16))
- # define CL_COR_ERR_CLASS_ATTR_INVALID_RELATION (CL_COR_SET_RC(CL_ERR_COMMON_MAX + 17))
- # define CL_COR_ERR_CLASS_ATTR_NOT_PRESENT (CL_COR_SET_RC(CL_ERR_COMMON_MAX + 13))
- # define CL_COR_ERR_DUPLICATE (CL_COR_SET_RC(CL_ERR_DUPLICATE))
- # define CL_COR_ERR_GET_DATA_NOT_FOUND (CL_COR_SET_RC(CL_COR_BUNDLE_ERR_MAX + 1))
- # define CL_COR_ERR_INVALID_BUFFER (CL_COR_SET_RC(CL_ERR_INVALID_BUFFER))
- # define CL_COR_ERR_INVALID_DEPTH (CL_COR_SET_RC(CL_ERR_COMMON_MAX + 5))
- # define CL_COR_ERR_INVALID_HANDLE (CL_COR_SET_RC(CL_ERR_INVALID_HANDLE))
- # define CL_COR_ERR_INVALID_PARAM (CL_COR_SET_RC(CL_ERR_INVALID_PARAMETER))
- # define CL_COR_ERR_INVALID_SIZE (CL_COR_SET_RC(CL_ERR_COMMON_MAX + 34))

- # define CL_COR_ERR_MAX_DEPTH (CL_COR_SET_RC(CL_ERR_COMMON_MAX + 4))
- # define CL_COR_ERR_NO_MEM (CL_COR_SET_RC(CL_ERR_NO_MEMORY))
- # define CL_COR_ERR_NO_RESOURCE (CL_COR_SET_RC(CL_ERR_NO_RESOURCE))
- # define CL_COR_ERR_NOT_EXIST (CL_COR_SET_RC(CL_ERR_NOT_EXIST))
- # define CL_COR_ERR_NOT_INITIALIZED (CL_COR_SET_RC(CL_ERR_COMMON_MAX + 30))
- # define CL_COR_ERR_NOT_SUPPORTED (CL_COR_SET_RC(CL_ERR_COMMON_MAX + 27))
- # define CL_COR_ERR_NULL_PTR (CL_COR_SET_RC(CL_ERR_NULL_POINTER))
- # define CL_COR_ERR_OBJ_ATTR_INVALID_SET (CL_COR_SET_RC(CL_ERR_COMMON_MAX + 18))
- # define CL_COR_ERR_OBJ_ATTR_NOT_PRESENT (CL_COR_SET_RC(CL_ERR_COMMON_MAX + 31))
- # define CL_COR_ERR_OBJ_NOT_PRESENT (CL_COR_SET_RC(CL_ERR_COMMON_MAX + 33))
- # define CL_COR_ERR_OI_ALREADY_REGISTERED (CL_COR_SET_RC(CL_COR_OI_ERR_MAX + 2))
- # define CL_COR_ERR_OI_NOT_REGISTERED (CL_COR_SET_RC(CL_COR_OI_ERR_MAX + 1))
- # define CL_COR_ERR_ROUTE_NOT_PRESENT (CL_COR_SET_RC(CL_ERR_COMMON_MAX + 26))
- # define CL_COR_ERR_ROUTE_PRESENT (CL_COR_SET_RC(CL_ERR_COMMON_MAX + 21))
- # define CL_COR_ERR_RUNTIME_ATTR_WRITE (CL_COR_SET_RC(CL_COR_UTILS_ERR_MAX + 2))
- # define CL_COR_ERR_RUNTIME_CACHED_SET (CL_COR_SET_RC(CL_COR_INST_ERR_MAX + 11))
- # define CL_COR_ERR_VERSION_UNSUPPORTED (CL_COR_SET_RC(CL_ERR_COMMON_MAX + 35))
- # define CL_COR_ERR_ZERO_JOBS_BUNDLE (CL_COR_SET_RC(CL_COR_BUNDLE_ERR_MAX + 4))
- # define CL_COR_INST_ERR_CHILD_MO_EXIST (CL_COR_SET_RC(CL_COR_MO_TREE_ERR_MAX + 9))
- # define CL_COR_INST_ERR_INVALID_MOID (CL_COR_SET_RC(CL_COR_MO_TREE_ERR_MAX + 5))
- # define CL_COR_INST_ERR_MAX_INSTANCE (CL_COR_SET_RC(CL_COR_MO_TREE_ERR_MAX + 11))
- # define CL_COR_INST_ERR_MO_ALREADY_PRESENT (CL_COR_SET_RC(CL_COR_MO_TREE_ERR_MAX + 3))
- # define CL_COR_INST_ERR_MSO_ALREADY_PRESENT (CL_COR_SET_RC(CL_COR_MO_TREE_ERR_MAX + 2))
- # define CL_COR_INST_ERR_MSO_EXIST (CL_COR_SET_RC(CL_COR_MO_TREE_ERR_MAX + 10))
- # define CL_COR_INST_ERR_MSO_NOT_PRESENT (CL_COR_SET_RC(CL_COR_MO_TREE_ERR_MAX + 4))
- # define CL_COR_INST_ERR_NODE_ALREADY_PRESENT (CL_COR_SET_RC(CL_COR_MO_TREE_ERR_MAX + 7))
- # define CL_COR_INST_ERR_NODE_NOT_FOUND (CL_COR_SET_RC(CL_COR_MO_TREE_ERR_MAX + 1))

- # define CL_COR_INST_ERR_PARENT_MO_NOT_EXIST (CL_COR_SET_RC(CL_COR_MO_TREE_ERR_MAX + 12))
- # define CL_COR_MO_TREE_ERR_MAX_INST (CL_COR_SET_RC(CL_COR_BASE_ERR_MAX + 7))
- # define CL_COR_NOTIFY_ERR_CANNOT_RESOLVE_CLASS (CL_COR_SET_RC(CL_COR_TXN_ERR_MAX + 2))
- # define CL_COR_NOTIFY_ERR_INVALID_OP (CL_COR_SET_RC(CL_COR_TXN_ERR_MAX + 1))
- # define CL_COR_SVC_ERR_INVALID_ID (CL_COR_SET_RC(CL_COR_NOTIFY_ERR_MAX + 2))
- # define CL_COR_UTILS_ERR_INVALID_KEY (CL_COR_SET_RC(CL_COR_SVC_ERR_MAX + 1))

3.2 Object Management Type Definitions

3.2.1 ClCorTxnIdT

typedef ClHandleT ClCorTxnIdT;

The type of the COR transaction ID used to identify a transaction session.

3.2.2 ClCorTxnJobIdT

typedef ClUint32T ClCorTxnJobIdT;

The type of COR transaction Job Id, used to identify a job uniquely within a transaction.

3.2.3 ClCorTxnFuncT

*typedef ClRcT (*ClCorTxnFuncT) (*
ClCorTxnIdT txnId,
ClCorTxnJobIdT jobId,
*void *cookie);*

The type for the callback function that is passed as an argument to the job walk function, `clCorTxnJobWalk()`. This callback function will be called for every job found in the transaction. The parameter `cookie` contains the user-data passed when the walk function is called. The parameters `txnId` contains the Transaction ID and `jobId` contains the ID of the job being walked.

3.2.4 ClCorTxnSessionIdT

typedef ClhandleT ClCorTxnSessionIdT;

The type of the handle of a COR transaction session. It is used by APIs that manipulate the COR object such as:

- Creating objects
- Setting attributes
- Deleting objects

3.3 Object Addressability Type Definitions

3.3.1 ClCorServiceIdT

typedef enum {
CL_COR_INVALID_SRVC_ID,

CL_COR_SVC_ID_FAULT_MANAGEMENT,
CL_COR_SVC_ID_ALARM_MANAGEMENT,
CL_COR_SVC_ID_PROVISIONING_MANAGEMENT,
CL_COR_SVC_ID_DUMMY_MANAGEMENT,
} ClCorServiceIdT;

The values of the `ClCorServiceIdT` enumeration type contains the service ID (fixed) for all MSPs.

- *CL_COR_SVC_ID_FAULT_MANAGEMENT* - represents the OpenClovis Fault Manager.
- *CL_COR_SVC_ID_ALARM_MANAGEMENT* - represents the OpenClovis Alarm Agent.
- *CL_COR_SVC_ID_PROVISIONING_MANAGEMENT* - represents the OpenClovis Provisioning Manager.
- *CL_COR_SVC_ID_DUMMY_MANAGEMENT* - represents the OpenClovis Dummy MSP.

3.3.2 ClCorMOServiceIdT

typedef ClInt16T ClCorMOServiceIdT;

`ClCorMOServiceIdT` is a part of the `ClCorMOIdT` structure to access the MO or MSO. This stores the service ID of the MO/MSO whose value is equal to any of the values in the enumeration type `ClCorServiceIdT`. For PROV MSO, the value of the service ID is CL_COR_SVC_ID_PROVISIONING_MANAGEMENT.

3.3.3 ClCorMoPathQualifierT

typedef enum {
CL_COR_MO_PATH_ABSOLUTE = 0,
CL_COR_MO_PATH_RELATIVE,
CL_COR_MO_PATH_RELATIVE_TO_BASE, CL_COR_MO_PATH_QUALIFIER_MAX =
CL_COR_MO_PATH_RELATIVE_TO_BASE
} ClCorMoPathQualifierT;

The enumeration `ClCorMoPathQualifierT` indicates if `ClCorMOId` or `corPath` is relative or absolute.

- *CL_COR_MO_PATH_ABSOLUTE = 0* - Specifies that the path is absolute. It is the equivalent for '/' in Unix.
- *CL_COR_MO_PATH_RELATIVE* - Specifies that the path is relative.
- *CL_COR_MO_PATH_RELATIVE_TO_BASE, CL_COR_MO_PATH_QUALIFIER_MAX = CL_COR_MO_PATH_RELATIVE_TO_BASE* - Specifies that the path is relative to the position of the blade in the COR hierarchy. It is the equivalent for ' ' in Unix.

This enumeration is also part of the `ClCorMOIdT` structure. The values of the enumeration indicate the type of the MOID required while working with the MOId. If the flag is `CL_COR_MO_PATH_ABSOLUTE`, the MOId is absolute and the first class ID in the MOId is taken as the root. If the flag is `CL_COR_MO_PATH_RELATIVE`, the first class ID in the MOId is considered as the current location. The current implementation supports the `CL_COR_MO_PATH_ABSOLUTE` flag only.

3.3.4 ClCorAddrT

typedef ClIocPhysicalAddressT ClCorAddrT;

This type definition contains the IOC physical address and the port address of the component that registers itself as an OI for a MO.

3.3.5 ClCorAddrPtrT

*typedef ClCorAddrT * ClCorAddrPtrT;*

A pointer to the IOC physical address structure. This is populated with the IOC physical address of the component that acts as an Object Implementor (OI) for a Managed Object (MO). It is passed as a parameter to the OI registration API.

3.3.6 ClCorMOId

```
typedef struct ClCorMOId{
        ClCorMOhandleT node[CL_COR_HANDLE_MAX_DEPTH];
        ClCorMOServiceIdT svcId;
        ClUint16T depth;
        ClCorMoPathQualifierT qualifier;
};

#define CL_COR_CLASS_WILD_CARD ((ClCorClassTypeT) 0xFFFFFFFE)

#define CL_COR_INSTANCE_WILD_CARD 0xFFFFFFFE
```

The structure `ClCorMOId` contains `MoId` of the object, which is the address of the COR object. It provides a unique identification for the MO object. The attributes of this structure are:

- *node* - MO handle address. This is the combination of class ID and instance ID that uniquely identifies a node and provides the path to access the object node. The class ID and instance ID can use wild card entries by assigning the macros, `CL_COR_CLASS_WILD_CARD` and `CL_COR_INSTANCE_WILD_CARD`.
- *svcId* - Service ID. It is 16 bits in length. This takes the values of the enumeration `ClCorServiceIdT`. The service ID `CL_COR_INVALID_SRVC_ID`, is used to access the MO. The service ID `CL_COR_SVC_ID_ALARM_MANAGEMENT` is used to access the alarm MSO. The service ID `CL_COR_SVC_ID_PROVISIONING_MANAGEMENT` is used to access the provisioning MSO.
- *depth* - Depth of the MoId. It is the number of elements in `ClCorMOhandleT` - 1.
- *qualifier* - Handle qualifier. This must contain the value, `CL_COR_MO_PATH_ABSOLUTE`.

3.3.7 ClCorMOIdPtrT

typedef ClCorMOIdT ClCorMOIdPtrT;*

A pointer type to *ClCorMOIdT*.

3.3.8 ClCorObjectHandleT

typedef struct ClCorObjectHandle {
ClUint8T tree [8];
} ClCorObjectHandleT;

The structure `ClCorObjectHandle` is the handle to MO. This handle is a compressed version of the `clCoirMOIdT` and identifies the hierarchy in the object tree. `objhandle` hierarchy is compressed and indicates the indexes. `objTree` handle represents the tree. The COR server returns this handle to the client after the object is created.

3.3.9 ClCorAttrPathT

typedef struct {
ClUint16T depth;
ClCorAttrIdIdxPairT node [CL_COR_CONT_ATTR_MAX_DEPTH];
ClUint16T tmp;
}ClCorAttrPathT;

The structure `ClCorAttrPathT` contains the path-list of the attribute. This structure is deprecated.

- *depth* - Depth of the path.
- *node* - Attribute ID and index pair.
- *tmp* - Used for padding.

3.3.10 ClCorAttrPathPtrT

typedef ClCorAttrPathT ClCorAttrPathPtrT;*

The pointer to `ClCorAttrPathT`.

3.3.11 ClCorMoIdClassGetFlagsT

typedef enum {
CL_COR_MO_CLASS_GET,
CL_COR_MSO_CLASS_GET
} ClCorMoIdClassGetFlagsT;

The values of the `ClCorMoIdClassGetFlagsT` enumeration type must be used in the `clCorMoIdToClassGet()` API. This API is used to request the class ID of the MO or MSO. While the class ID of the MO or MSO is requested, the MoId must be provided.

3.3.12 ClCorObjTypesT

typedef enum {
CL_COR_OBJ_TYPE_SIMPLE,

```
        CL_COR_OBJ_TYPE_MO,
        CL_COR_OBJ_TYPE_MSO
} ClCorObjTypesT;
```

The values of the `ClCorObjWalkFlagsT` enumeration type is used to obtain the type of the COR object. A COR object can be of the type MO or MSO. This is used by the `clCorObjectHandleToTypeGet()` API which takes the object handle as the IN parameter and returns the type of the object as the OUT parameter.

3.4 Object Search Type definitions

3.4.1 ClCorAttrWalkOpT

```
typedef enum{
        CL_COR_ATTR_INVALID_OPTION = 0,
        CL_COR_ATTR_WALK_ALL_ATTR,
        CL_COR_ATTR_WALK_ONLY_MATCHED_ATTR,
        CL_COR_ATTR_WALK_MAX
}ClCorAttrWalkOpT;
```

The enumeration `ClCorAttrWalkOpT` contains the various options for walk operation on the attributes. If `CL_COR_ATTR_WALK_ONLY_MATCHED_ATTR` is specified, the walk is performed only on the matching attributes in MO. If `CL_COR_ATTR_WALK_ALL_ATTR` is specified, the walk is performed on all the attributes in MO.

3.4.2 ClCorAttrCmpFlagT

```
typedef enum{
        CL_COR_ATTR_CMP_FLAG_INVALID = 0,
        CL_COR_ATTR_CMP_FLAG_VALUE_EQUAL_TO,
        CL_COR_ATTR_CMP_FLAG_VALUE_LESS_THAN,
        CL_COR_ATTR_CMP_FLAG_VALUE_LESS_OR_EQUALS,
        CL_COR_ATTR_CMP_FLAG_VALUE_GREATER_THAN,
        CL_COR_ATTR_CMP_FLAG_VALUE_GREATER_OR_EQUALS,
        CL_COR_ATTR_CMP_FLAG_MAX
        }ClCorAttrCmpFlagT;
```

The enumeration `ClCorAttrCmpFlagT` contains the comparison flags used to compare the attribute values with a specified value. It is used to filter the attributes of the MO while performing the walk operation.

3.4.3 ClCorObjectWalkFunT

```
typedef ClRcT(*ClCorObjectWalkFunT)(void *data, void *cookie);
```

This typedef is the prototype for the *COR Object Walk* callback function that must be provided as a parameter to the `clCorObjectWalk()` API. The callback API is defined by the application and is called for each object.

3.4.4 ClCorObjAttrWalkFilter

typedef struct ClCorObjAttrWalkFilter{
ClUint8T baseAttrWalk;
*ClCorAttrPathT *pAttrPath;*
ClCorAttrIdT attrId;
ClInt32T index;
ClCorAttrCmpFlagT cmpFlag;
ClCorAttrWalkOpT attrWalkOption;
ClUint32T size;
*void *value;*
};

typedef struct ClCorObjAttrWalkFilter ClCorObjAttrWalkFilterT;

The structure `ClCorObjAttrWalkFilter` is used to specify filter properties while performing attribute walk operation. The attributes of the structure are:

- *baseAttrWalk* - This is a depreciated feature and must be `CL_TRUE` for attribute walk.
- *contAttrPath* - This is a depreciated feature and must be `CL_TRUE` for attribute walk.
- *pAttrPath* - This is a deprecated feature and must be set to NULL.
- *attrId* - This must contain either a valid attribute ID or `CL_COR_INVALID_ATTR_ID`. If the value is set to `CL_COR_INVALID_ATTR_ID`, no attribute value comparison is performed.
- *index* - It is used to specify the index for ARRAY attributes. For a SIMPLE attribute, the index is set to `CL_COR_INVALID_ATTR_IDX`.
- *cmpFlag* - The comparison flag is used to compare an attribute ID against a specified value. Following are the comparison flags:
  - `CL_COR_ATTR_CMP_FLAG_VALUE_EQUAL_TO`: The attributes whose value is equal to the specified value is matched.
  - `CL_COR_ATTR_CMP_FLAG_VALUE_LESS_THAN`: The attributes whose value is greater than the specified value is matched.
  - `CL_COR_ATTR_CMP_FLAG_VALUE_LESS_OR_EQUALS`: The attributes whose value is greater than or equal to the specified value is matched.
  - `CL_COR_ATTR_CMP_FLAG_VALUE_GREATER_THAN`: The attributes whose value is less than the specified value is matched.
  - `CL_COR_ATTR_CMP_FLAG_VALUE_GREATER_OR_EQUALS`: The attributes whose value is less than or equal to the specified value is matched.
- *attrWalkOption* - If comparison condition is true, `attrWalkOption` can be set to be `CL_COR_ATTR_WALK_ALL_ATTR` or `CL_COR_ATTR_WALK_ONLY_MATCHED_ATTR`.
- *size* - The size of the value.
- *value* - Pointer to the value.

3.4.5 ClCorObjAttrWalkFuncT

typedef ClRcT (ClCorObjAttrWalkFuncT) ({
ClCorAttrPathPtrT pAttrPath,
ClCorAttrIdT attrId,
ClCorAttrTypeT attrType,
ClCorTypeT attrDataType,
*void *value,*
ClUint32T size,
ClPtrT attrData,
*void *cookie);*

The type of the callback API that is invoked for every attribute within a COR object, during the walk operation.

- *pAttrPath* - Path of contained object whose attribute is being walked. This must be NULL for base object attributes.
- *attrId* - Attribute ID.
- *attrType* - Attribute type. It can take one of the following values:
  - `CL_COR_SIMPLE_ATTR`
  - `CL_COR_ARRAY_ATTR`
  - `CL_COR_ASSOCIATION_ATTR`
- *attrDataType* - Data type of the attribute. This is valid for SIMPLE and ARRAY attributes only. For ASSOCIATION attributes, `attrDataType` is `CL_COR_INVALID_DATA_TYPE`.
- *value* - Pointer to the value of the attribute.
- *size* - Size of the value.
- *attrData* - Pointer to the attribute data, `ClCorAttrFlagT` which contains the flags set for the attribute.
- *cookie* - The user data (or cookie) that is passed as a parameter to the `clCorObjectAttributeWalk()` API.

3.4.6 ClCorObjWalkFlagsT

typedef enum {
CL_COR_MOTREE_WALK,
CL_COR_MO_WALK,
CL_COR_MSO_WALK,
CL_COR_MO_SUBTREE_WALK,
CL_COR_MSO_SUBTREE_WALK,
CL_COR_MO_WALK_UP,
CL_COR_MSO_WALK_UP
} ClCorObjWalkFlagsT;

The `ClCorObjWalkFlagsT` enumeration type contains the walk related definitions. It is used to perform a walk operation on a MO tree. The following are currently supported:

- *CL_COR_MO_WALK* - The walk is performed through the object tree and returns the MOs below the root MoId that satisfies the filter criteria.
- *CL_COR_MSO_WALK* - Returns the object handle of all the MSO objects below the root MOId that satisfies the filter criteria.

3.5 Managed Object Class Type Definitions

3.5.1 ClCorClassTypeT

typedef ClInt32T ClCorClassTypeT;

The type of an identifier for the COR class.

3.5.2 ClCorInstanceIdT

typedef ClInt32T ClCorInstanceIdT;

The type of an identifier for a COR instance.

3.5.3 ClCorTypeT

```
typedef enum ClCorType{
        CL_COR_INVALID_DATA_TYPE = -1,
        CL_COR_VOID,
        CL_COR_INT8,
        CL_COR_UINT8,
        CL_COR_INT16,
        CL_COR_UINT16,
        CL_COR_INT32,
        CL_COR_UINT32,
        CL_COR_INT64,
        CL_COR_UINT64,
        CL_COR_FLOAT,
        CL_COR_DOUBLE,
        CL_COR_COUNTER32,
        CL_COR_COUNTER64,
        CL_COR_SEQUENCE32,
} ClCorTypeT;
```

The `ClCorType` enumeration contains the basic COR data types. Following are its values:

- *CL_COR_VOID* - Void data type.
- *CL_COR_INT8* - Character data type.
- *CL_COR_UINT8* - Unsigned character.
- *CL_COR_INT16* - Short data type.
- *CL_COR_UINT16* - Unsigned short.

- *CL_COR_INT32* - Integer data type.
- *CL_COR_UINT32* - Unsigned integer data type.
- *CL_COR_INT64* - Long long data type.
- *CL_COR_UINT64* - Unsigned long long data type.
- *CL_COR_FLOAT* - Float data type. This data type will be supported in future releases. The default value in the current implementation is `CL_COR_UINT32`.
- *CL_COR_DOUBLE* - Double data type. This data type will be supported in future releases. The default value in the current implementation is `CL_COR_UINT32`.
- *CL_COR_COUNTER32* - Counter data type. This data type will be supported in future releases. The default value in the current implementation is `CL_COR_UINT32`.
- *CL_COR_COUNTER64* - Counter 64-bits data type. This data type will be supported in future releases. The default value in the current implementation is `CL_COR_UINT32`.
- *CL_COR_SEQUENCE32* - Sequence number data type.This data type will be supported in future releases. The default value in the current implementation is `CL_COR_UINT32`.

3.5.4 ClCorAttrTypeT

typedef enum {
CL_COR_MAX_TYPE,
CL_COR_SIMPLE_ATTR,
CL_COR_ARRAY_ATTR,
CL_COR_CONTAINMENT_ATTR,
CL_COR_ASSOCIATION_ATTR,
CL_COR_VIRTUAL_ATTR
} ClCorAttrTypeT;

The values of the `ClCorAttrTypeT` enumeration type refer to the COR attribute types. Following are its values:

- *CL_COR_SIMPLE_ATTR* - SIMPLE attribute type.
- *CL_COR_ARRAY_ATTR* - ARRAY attribute type.
- *CL_COR_CONTAINMENT_ATTR* - Containment attribute type. This is currently not supported.
- *CL_COR_ASSOCIATION_ATTR* - Association attribute type. This is currently not supported.
- *CL_COR_VIRTUAL_ATTR* - Virtual attribute type. This is currently not supported.

3.5.5 ClCorOpsT

typedef enum {
CL_COR_OP_RESERVED,
CL_COR_OP_CREATE,
CL_COR_OP_SET,

```
        CL_COR_OP_DELETE,
        CL_COR_OP_ALL
} ClCorOpsT;
```

The values of the `ClCorOpsT` enumeration type contain the Operation IDs. A combination of Operation IDs can be used.

3.5.6 ClCorAttrIdT

typedef ClInt32T ClCorAttrIdT;

The type of an identifier for a COR attribute.

- *#define CL_COR_INVALID_ATTR_ID -1*
- *#define CL_COR_INVALID_ATTR_IDX -1*

The macro `CL_COR_INVALID_ATTR_ID` defines an invalid attribute ID used to specify the search criterion during the object attribute walk.

The macro `CL_COR_INVALID_ATTR_IDX` defines an invalid array index used to specify the index of non array attribute.

3.5.7 ClCorAttrFlagT

typedef ClUint32T ClCorAttrFlagT;

The type of an identifier for specifying the attribute flag. This is 32 bit value containing the ORed value of the valid combinations of the attribute flags.

- *#define CL_COR_ATTR_CACHED 0x00000100*
- *#define CL_COR_ATTR_NON_CACHED 0x00000200*
- *#define CL_COR_ATTR_PERSISTENT 0x00000400*
- *#define CL_COR_ATTR_NON_PERSISTENT 0x00008400*
- *#define CL_COR_ATTR_CONFIG 0x01000000*
- *#define CL_COR_ATTR_RUNTIME 0x02000000*

3.6 Library Life Cycle APIs

3.6.1 clCorBundleInitialize

clCorBundleInitialize

Synopsis:
Creates a bundle and returns a unique handle identifying the bundle.

Header File:
clCorUtilityApi.h

Syntax:

```
   ClRcT clCorBundleInitialize (
CL_OUT ClCorBundleHandleT *bundleHandle,
CL_IN  ClCorBundleConfigT  *config);
```

Parameters:

bundleHandle: (out) This parameter identifies the bundle.

config: (in) The `config->bundleType` element indicates if the bundle is a transactional bundle or a non transactional bundle.

Return values:

CL_OK: The API executed successfully. The job was successfully queued in the bundle.

CL_COR_ERR_NO_MEM: Bundle initialization failed due to insufficient memory.

CL_COR_ERR_BUNDLE_INIT_FAILURE: Generic bundle initialization failure.

CL_COR_ERR_BUNDLE_INVALID_CONFIG: The transactional bundle configuration is currently not supported.

CL_COR_ERR_NULL_PTR: `bundleHandle` is NULL.

Description:
This API creates a bundle and returns a unique handle that identifies this bundle. `clCorBundleObjectGet()`, `clCorBundleApply()`, `clCorBundleApplyAsync()`, and `clCorBundleFinalize()` APIs use this handle to uniquely identify the bundle.

Library File:
ClCorClient

Related Function(s):
clCorBundleApply, clCorBundleApplyAsync, clCorBundleFinalize.

3.6.2 clCorBundleFinalize

clCorBundleFinalize

Synopsis:
Finalizes the bundle.

Header File:
clCorUtilityApi.h

Syntax:

```
ClRcT clCorBundleFinalize(
 CL_INOUT ClCorBundleHandleT *bundleHandle);
```

Parameters:
bundleHandle: (in/out) This parameter identifies the bundle.

Return values:
CL_OK: Bundle is finalized successfully.

CL_COR_ERR_BUNDLE_FINALIZE: Failure while deleting resources.

CL_COR_ERR_INVALID_HANDLE: `bundleHandle` is invalid.

Description:
This API finalizes the bundle and frees all the resources associated with the bundle. `bundleHandle` then becomes invalid.

Library File:
ClCorClient

Related Function(s):
clCorBundleInitialize.

3.7 Object Management APIs

3.7.1 clCorObjectCreate

clCorObjectCreate

Synopsis:
Creates a COR object.

Header File:
clCorApi.h

Syntax:

```
ClRcT clCorObjectCreate(
                            CL_INOUT ClCorTxnSessionIdT *txnSessionId,
                            CL_IN  ClCorMOIdPtrT moId,
                            CL_OUT ClCorObjectHandleT *handle);
```

Parameters:

txnSessionID: (in/out) For a SIMPLE transaction containing one job, the value of the parameter must be `CL_COR_SIMPLE_TXN`. For a COMPLEX transaction containing multiple jobs, it must be initialized to zero and passed to the API for adding the first job. To add the subsequent jobs, the returned value from the previous API should be passed as the parameter.

MoId: (in) ID of the object to be created.

handle: (out) Pointer to the object handle.

Return values:

CL_OK: The API executed successfully.

CL_COR_ERR_NULL_PTR: `txnSessionId` or `moId` is a NULL pointer.

CL_COR_ERR_NO_MEM: Memory allocation failure.

CL_COR_INST_ERR_INVALID_MOID: `MoId` is invalid.

CL_COR_ERR_CLASS_NOT_PRESENT: The specified Class is not present.

CL_COR_INST_ERR_NODE_NOT_FOUND: Parent class is not present in the instance tree.

CL_COR_MO_TREE_ERR_NODE_NOT_FOUND: `moTree` node is not present for the class.

CL_COR_INST_ERR_MAX_INSTANCE: Maximum instance count for this class is reached.

CL_COR_INST_ERR_MO_ALREADY_PRESENT: MO is present in the instance tree.

CL_COR_ERR_VERSION_UNSUPPORTED: Version is not supported.

CL_COR_INST_ERR_MSO_ALREADY_PRESENT: MSO exists in the Object Tree.

CL_COR_MO_TREE_ERR_CLASS_NO_PRESENT: MSO Class is not present in the MO Tree.

Description:
This API is used to create MO and MSO COR objects. The `MoId`, passed to this API must be valid (the MO class tree must be present for that `MoId`.

To create an MO object, the service ID should be specified as `CL_COR_INVALID_SVC_ID` in the MoId. To create an MSO object, the service ID corresponding to the MSO should be specified in the MOID.
If the creation of the object is part of a COMPLEX transaction, the object handle returned is valid only after the transaction is committed using the `clCorTxnSessionCommit()` API. For a COMPLEX transaction involving multiple jobs, the `txnSessionId` must be initialized to 0 and sent to the API to add the first job. This API returns a transaction session ID as the OUT parameter. This can be used for adding the subsequent jobs using `clCorObjectAttributeSet()/clCorObjectDelete()/clCorObjectCreate()` APIs. A COMPLEX transaction can be committed by using the `clCorTxnSessionCommit()` API.
If the creation of the object is part of a SIMPLE transaction, `txnSessionId` must be set to the value `CL_COR_SIMPLE_TXN`. A SIMPLE transaction can take only one job.

Library File:
ClCorClient

Related Function(s):
clCorObjectAttributeGet, clCorObjectAttributeSet, clCorObjectDelete.

3.7.2 clCorObjectAttributeSet

clCorObjectAttributeSet

Synopsis:
Sets the attribute of a COR object

Header File:
clCorApi.h

Syntax:

```
ClRcT clCorObjectAttributeSet(
                                    CL_INOUT ClCorTxnSessionIdT *txnSessionId
                                    CL_IN    ClCorObjectHandleT phandle,
                                    CL_IN    ClCorAttrPathPtrT contAttrPath,
                                    CL_IN    ClCorAttrIdT attrId,
                                    CL_IN    ClUint32T index,
                                    CL_IN    void *value,
                                    CL_IN    ClUint32T size);
```

Parameters:

txnSessionID: (in/out) Transaction Session ID.

phandle: (in) Handle of the object whose attribute is being set.

contAttrPath: (in) Containment hierarchy path.

attrID: (in) ID of the attribute.

index: (in) Index of the attribute. It must be set to `CL_COR_INVALID_ATTR_IDX` for a SIMPLE attribute.

value: (in) Pointer to the value that is required to be set.

size: (in) Size of the value.

Return values:

CL_OK: The API executed successfully.

CL_COR_ERR_NULL_PTR: `txnSessionId` or `value` is a NULL pointer.

CL_COR_TXN_ERR_INVALID_JOB_ID: The job ID is invalid .

CL_COR_ERR_CLASS_ATTR_INVALID_INDEX: Index is used for the SIMPLE attribute.

CL_COR_ERR_CLASS_ATTR_NOT_PRESENT: The attribute ID is not present.

CL_COR_ERR_INVALID_SIZE: For SIMPLE attributes, the parameter `size` is less than the `size` associated with the attribute `attrId`. For array attributes, this error is returned in one of the following cases:

- `size` is greater than the size of the associated array elements that are required to be updated.
- the parameter `size` is less than the size of an individual array element.
- The parameter `size` is not an integer. This integer value must be the multiple of the size of the individual array element.

CL_COR_ERR_OBJ_ATTR_INVALID_SET: This error code is applicable to SIMPLE attributes, when the value is not within the range specified by *min* and *max* values associated with the attribute.

CL_COR_ERR_CLASS_ATTR_INVALID_INDEX: Invalid index for the attribute.

CL_COR_ERR_CLASS_ATTR_INVALID_RELATION: Size of the attribute is invalid.

CL_COR_ERR_VERSION_UNSUPPORTED: Version is not supported.

Description:

This API is used to set the value of the attributes of an object . `contAttrPath` can contain a valid attribute path or NULL. The current implementation supports only a NULL value for this parameter.

`txnSessionId` contains the ID for the transaction. For a SIMPLE transaction involving only one attribute set, this must be set to `CL_COR_SIMPLE_TXN`. For a COMPLEX transaction involving multiple attribute sets, this should be initialized to zero and sent to the API for adding the first job. The API updates `txnSessionId` and this updated value can be passed to add subsequent jobs to the transaction using `clCorObjectAttributeSet()/clCorObjectCreate()/clCorObjectDelete()` APIs. The transaction must be committed by using the `clCorTxnSessionCommit()` API. For SIMPLE attributes, the index must be set to `-1` or `CL_COR_INVALID_ATTR_IDX`. The size of the value must be equal to its actual size.

Library File:

ClCorClient

Related Function(s):

clCorObjectCreate, clCorObjectAttributeGet, clCorObjectDelete.

3.7.3 clCorObjectDelete

clCorObjectDelete

Synopsis:
Deletes a COR object.

Header File:
clCorApi.h

Syntax:

```
ClRcT clCorObjectDelete(
                        CL_INOUT  ClCorTxnSessionIdT *txnSessionId,
                        CL_IN     ClCorObjecthandleT handle);
```

Parameters:
txnSessionID: (in/out) Transaction Session ID.
handle: (in) Handle of the object to be deleted.

Return values:
CL_OK: The API executed successfully.
CL_COR_ERR_NULL_PTR: `txnSessionId` is a NULL pointer.
CL_COR_INVALID_SRVC_ID: Service ID is incorrect.
CL_COR_INST_ERR_INVALID_MOID: `MoId` is invalid.
CL_COR_INST_ERR_CHILD_MO_EXIST: Child MO exists for the MO object node.
CL_COR_INST_ERR_MSO_EXIST: MSO exists for the MO object node.
CL_COR_INST_ERR_NODE_NOT_FOUND: Node not found in the object tree.
CL_COR_ERR_VERSION_UNSUPPORTED: Version is not supported.
CL_COR_MO_TREE_ERR_NODE_NOT_FOUND: MO tree node not found.

Description:
This API is used to delete a COR object. The parameter, `handle`, contains the handle of the object (obtained through `clCorObjectHandleGet()`) to be deleted.
To delete an MSO object, the MoId containing the corresponding service ID must be sent to the `clCorObjectHandleGet()` API. This API returns a handle to the object that can be used to delete the MSO object.
To delete the MO object, the service ID of the MoId must be set to `CL_COR_INVALID_SVC_ID`. The MSOs and child MOs must be deleted before deleting the parent MO. The MOs and MSOs can be deleted in a single transaction.
`txnSessionId` contains the ID for the transaction. For a SIMPLE transaction involving only one object delete, this must be set to `CL_COR_SIMPLE_TXN`. For a COMPLEX transaction involving multiple `CREATE, SET,` and `DELETE`, this should be initialized to '0' and sent to the API to add the first job. The API updates this transaction ID. The updated transaction ID must be sent to add subsequent jobs to the transaction using `clCorObjectCreate()/clCorObjectAttributeSet()/clCorObjectDelete()` APIs. The transaction should be committed by using the `clCorTxnSessionCommit()` API.

Library File:
ClCorClient

Related Function(s):
clCorObjectCreate.

3.7.4 clCorUtilMoAndMSOCreate

clCorUtilMoAndMSOCreate

Synopsis:
Creates MO and MSO objects.

Header File:
clCorUtilityApi.h

Syntax:

```
   ClRcT clCorUtilMoAndMSOCreate(
CL_IN ClCorMOIdPtrT pMoId,
CL_OUT ClCorObjecthandleT *pHandle);
```

Parameters:
pMoid: (in) MOId of the MO to be created.

pHandle: (out) Handle of the MO object created.

Return values:
CL_OK: The API executed successfully.

CL_COR_INST_ERR_MO_ALREADY_PRESENT: MO exists in the instance tree.

CL_COR_ERR_NULL_PTR: `pmoId` is a NULL pointer.

CL_COR_ERR_NO_MEM: Memory allocation failure.

CL_COR_INST_ERR_INVALID_MOID: `pMoId` is invalid.

CL_COR_ERR_CLASS_NOT_PRESENT: The specified class is not present.

CL_COR_INST_ERR_NODE_NOT_FOUND: Parent class is not present in the instance tree.

CL_COR_MO_TREE_ERR_NODE_NOT_FOUND: `moTree` node not found for the class.

CL_COR_INST_ERR_MAX_INSTANCE: Maximum Instance count for this class is reached.

CL_COR_ERR_VERSION_UNSUPPORTED: Version is not supported.

Description:
This API creates the MO with its associated MSO objects and returns a handle to it. The service ID of `pMoId` must be set to `CL_COR_INVALID_SVC_ID`.

Library File:
ClCorClient

Related Function(s):
clCorUtilMoAndMSODelete.

3.7.5 clCorUtilMoAndMSODelete

clCorUtilMoAndMSODelete

Synopsis:
Deletes MO and MSO objects.

Header File:
clCorUtilityApi.h

Syntax:

```
   ClRcT clCorUtilMoAndMSODelete(
CL_IN ClCorMOIdPtrT pMoId);
```

Parameters:
pMoid: (in) MOID of the MO to be deleted.

Return values:
CL_OK: The API executed successfully.
CL_ERR_NOT_EXIST: MO class type does not exist.
CL_COR_ERR_INVALID_PARAM: `pMoId` of MSO passed instead of MO.
CL_COR_ERR_NULL_PTR: `pMoId` is a NULL pointer.
CL_COR_INST_ERR_INVALID_MOID: MoId is invalid.
CL_COR_INST_ERR_CHILD_MO_EXIST: A child MO exists for the MO object node.
CL_COR_INST_ERR_NODE_NOT_FOUND: Node not found in the object tree.
CL_COR_ERR_VERSION_UNSUPPORTED: Version is not supported.
CL_COR_MO_TREE_ERR_NODE_NOT_FOUND: MO tree node not found.

Description:
This API is used to delete an MO and its associated MSO objects. The service ID of MOID must be set to `CL_COR_INVALID_SVC_ID` and passed to this API.

Library File:
ClCorClient

Related Function(s):
clCorUtilMoAndMSOCreate.

3.7.6 clCorObjectAttributeGet

clCorObjectAttributeGet

Synopsis:
Retrieves the value of an attribute belonging to an object.

Header File:
clCorApi.h

Syntax:

```
ClRcT clCorObjectAttributeGet(
                                    CL_IN ClCorObjecthandleT pHandle,
                                    CL_IN ClCorAttrPathPtrT contAttrPath,
                                    CL_IN ClCorAttrIdT attrId,
                                    CL_IN ClInt32T index,
                                    CL_OUT void *value,
                                    CL_INOUT ClUint32T *size);
```

Parameters:

pHandle: (in) handle of the object whose attribute is being read.

contAttrPath: (in) Path of the containment hierarchy.

attrID: (in) ID of the attribute.

index: (in) Attribute index. It is set to `CL_COR_INVALID_ATTR_IDX` for a SIMPLE attribute.

value: (out) Pointer to the value. The attribute value is copied into this parameter.

size: (in/out) Size of the value.

Return values:

CL_OK: The API executed successfully.

CL_COR_ERR_NULL_PTR: `value` or `size` is a NULL pointer.

CL_COR_ERR_NO_MEM: Memory allocation failure.

CL_COR_ERR_CLASS_ATTR_NOT_PRESENT: Attribute ID passed is not present.

CL_COR_ERR_CLASS_ATTR_INVALID_RELATION: Size of the attribute is invalid.

CL_COR_ERR_INVALID_SIZE: For SIMPLE attributes, the parameter size, must be equal to the size if the attribute. For array attributes, this error is returned in one of the following cases:

- `size` is greater than the size of the associated array elements that must be updated.
- the parameter `size` is less than size of a single array element.
- The parameter `size` is not an integer. This integer must be a multiple of the size of the single array element.

CL_COR_ERR_CLASS_ATTR_INVALID_INDEX: `index` is used for SIMPLE attributes.

CL_COR_ERR_VERSION_UNSUPPORTED: Version is not supported.

Description:
This API is used to `GET` (retrieve) the value of an attribute. `pHandle` is the handle to the object from which the attribute value is to be read. The object handle is returned when the object is created and can also be obtained by passing MOID to the API `clCorObjectHandleGet()`.

Parameter `pAttrPath` contains the containment path hierarchy where the attribute resides. This is currently not supported. So, it must be set to NULL.
Parameter `index` contains the index of the attribute. For SIMPLE attributes, this can be set to `-1` or `CL_COR_INVALID_ATTR_IDX`.
The parameter `value` contains the address where the retrieved value should be copied.
The parameter `size` must contain the size of the value to be retrieved from an attribute.

Library File:
ClCorClient

Related Function(s):
clCorObjectCreate, clCorObjectAttributeSet.

3.7.7 clCorObjectHandleGet

clCorObjectHandleGet

Synopsis:
Retrieves the compressed MO handle corresponding to MOID.

Header File:
clCorApi.h

Syntax:

```
 ClRcT clCorObjectHandleGet(
CL_IN ClCorMOIdPtrT pMoId,
CL_OUT ClCorObjectHandleT *objhandle);
```

Parameters:

pMoid: (in) Pointer to the MOID.

****objHandle:*** (out) Pointer to object handle.

Return values:

CL_OK: The API executed successfully.

CL_COR_ERR_NULL_PTR: `pMoId` is a NULL pointer.

CL_COR_INST_ERR_INVALID_MOID: `MoId` is invalid.

CL_COR_ERR_INVALID_DEPTH: The depth of the `MoId` is greater than the maximum depth that is configured.

CL_COR_UTILS_ERR_INVALID_KEY: Failure to locate the node in the MO tree.

CL_COR_ERR_INVALID_PARAM: On passing an invalid parameter.

CL_COR_ERR_VERSION_UNSUPPORTED: Version is not supported.

Description:
This API is used to get the handle of an object for a given MOID. The object handle returned is the compressed value of the MOID. This object handle can be used for performing `SET`, `GET`, and `DELETE` operations on the MO.

Library File:
ClCorClient

Related Function(s):
clCorObjectHandleToMoIdGet, clCorObjectHandleToTypeGet.

3.7.8 clCorObjectHandleToTypeGet

clCorObjectHandleToTypeGet

Synopsis:
Returns the type of an object when its object handle is provided.

Header File:
clCorApi.h

Syntax:

```
ClRcT clCorObjectHandleToTypeGet(
                            CL_IN   ClCorObjecthandleT pHandle,
                            CL_OUT  ClCorObjTypesT* type);
```

Parameters:

pHandle: (in) Object handle.

type: (out) It can be set to `CL_COR_OBJ_TYPE_MO` or `CL_COR_OBJ_TYPE_MSO`.

Return values:

CL_OK: The API is executed successfully.

CL_COR_ERR_NULL_PTR: `type` is a NULL pointer.

CL_COR_UTILS_ERR_INVALID_KEY: Invalid value of object handle.

CL_COR_ERR_INVALID_PARAM: On passing an invalid parameter.

CL_COR_ERR_INVALID_DEPTH: Invalid depth of `MoId`.

CL_COR_ERR_CLASS_INVALID_PATH: Qualifiers are invalid.

CL_COR_SVC_ERR_INVALID_ID: Service ID of `MoId` is invalid.

CL_COR_ERR_VERSION_UNSUPPORTED: Version is not supported.

Description:
This API is used to retrieve the object type given a object handle. The type of an object indicates if it is an MO or an MSO object.

Library File:
ClCorClient

Related Function(s):
clCorObjecthandleGet, clCorObjectHandleToMoIdGet.

3.7.9 clCorObjecthandleToMoIdGet

clCorObjectHandleToMoIdGet

clCorObjecthandleToMoIdGet

Synopsis:
Returns the MOID corresponding to compressed MO handle.

Header File:
clCorApi.h

Syntax:

```
ClRcT clCorObjecthandleToMoIdGet(
                        CL_IN ClCorObjecthandleT objhandle,
                        CL_OUT ClCorMOIdPtrT moId,
                        CL_OUT ClCorServiceIdT *srvcId);
```

Parameters:

objhandle: (in) Object handle.

MoId: (out) MOID of the object.

srvcID: (out) Service ID of the object.

Return values:

CL_OK: The API executed successfully.

CL_COR_ERR_NULL_PTR: `moid` or `srvcId` is a NULL pointer.

CL_COR_UTILS_ERR_INVALID_KEY: Invalid value of object handle passed.

CL_COR_ERR_INVALID_PARAM: On passing an invalid parameter.

CL_COR_ERR_VERSION_UNSUPPORTED: Version is not supported.

Description:
This API returns the `MoId` for a given object handle. The service ID of the object is returned in `srvcId`.

Library File:
ClCorClient

Related Function(s):
clCorObjecthandleGet.

3.7.10 clCorTxnSessionCommit

clCorTxnSessionCommit

Synopsis:
Commits an active transaction session.

Header File:
clCorTxnApi.h

Syntax:

```
ClRcT    clCorTxnSessionCommit(
                        CL_IN ClCorTxnSessionIdT txnSessionId);
```

Parameters:
txnSessionID: (in) The session ID is a unique ID which identifies a COMPLEX transaction obtained while calling object create, attribute set, or object delete APIs.

Return values:
CL_OK: The API executed successfully.

CL_COR_TXN_ERR_ZERO_JOBS: There are no jobs to commit.

Description:
The transaction session ID is obtained using `clCorObjectCreate()/clCorObjectAttributeSet()/clCorObjectDelete()` APIs. When this API is invoked, the transaction request is sent to the COR server for processing. The server contacts the OIs for MOs in the transaction job and completes the transaction in a two phase commit manner. This is a synchronous operation so the thread calling this API is blocked until a response is received.

Library Name:
ClCorClient

Related Function(s):
clCorTxnSessionCancel.

3.7.11 clCorTxnSessionCancel

clCorTxnSessionCancel

Synopsis:
Cancels a transaction session.

Header File:
clCorTxnApi.h

Syntax:

```
ClRcT    clCorTxnSessionCancel(
                        CL_IN    ClCorTxnSessionIdT  txnSessionId);
```

Parameters:
txnSessionID: (in) Transaction session ID.

Return values:
CL_OK: The API executed successfully.
CL_ERR_INVALID_HANDLE: The `txnSessionId` is invalid.

Description:
This API is used to cancel the active transaction sessions whose session ID is sent to this API.

Library Name:
ClCorClient

Note
This functionality is not supported in the current release.

Related Function(s):
clCorTxnSessionCommit.

3.7.12 clCorTxnSessionFinalize

clCorTxnSessionFinalize

Synopsis:
Finalizes a COR transaction session.

Header File:
clCorTxnApi.h

Syntax:

```
       ClRcT clCorTxnSessionFinalize(
CL_IN ClCorTxnSessionIdT txnSessionId);
```

Parameters:
txnSessionID: (in) Transaction session ID.

Return values:
CL_OK: The API executed successfully.

CL_ERR_INVALID_HANDLE: The `txnSessionId` is invalid.

Description:
This API is used to free all the resources that are allocated for a COMPLEX transaction only when the transaction fails. It frees the allocated resources which include the client specific data and the failed jobs list. This function should be called to free the resources used by the transaction session.

Library Name:
ClCorClient

Related Function(s):
clCorTxnSessionCommit.

3.7.13 clCorTxnFailedJobGet

clCorTxnFailedJobGet

Synopsis:
Retrieves the information about the failed transaction job for a particular transaction ID.

Header File:
clCorTxnApi.h

Syntax:

```
ClRcT clCorTxnFailedJobGet(
 CL_IN  ClCorTxnSessionIdT txnSessionId,
                CL_IN  ClCorTxnInfoT *pPrevTxnInfo,
                CL_OUT ClCorTxnInfoT *pNextTxnInfo)
```

Parameters:

txnSessionID: (in) The transaction session ID for which the transaction failed.

pPrevTxnInfo: (in) If this is set to NULL, the first failed job information is sent in the second parameter. Otherwise, the next entry with respect to `pPrevTxnInfo` is sent.

pNextTxnInfo: (out) The information of the failed job in the transaction is populated in this parameter.

Return values:

CL_OK: The API executed successfully.

CL_COR_TXN_ERR_FAILED_JOB_GET: For this transaction ID, there is no failed job information available.

Description:
Applications that initiate COMPLEX transactions to perform MO `CREATE/DELETE` and attribute `SET` operations, need to use this API to obtain information about failures. Any component that participates in a transaction can report failures through their agent callback APIs. This API retrieves all such reported failures.
In case of multiple failures (on different components), there would be more than one error entries added in COR for a particular transaction. All the error entries are obtained through the following mechanism:

1. Obtain First Entry: Call the API with the parameter `pPrevTxnInfo` specified as NULL. This API returns the first error record in `pNextTxnInfo` parameter.
2. Obtain SubSequent Entries: Copy the content of `pNextTxnInfo` obtained in previous invocation of this API, and assign it to `pPrevTxnInfo` and call `clCorTxnFailedJobGet()`.

Note:
After receiving failed jobs, the `clCorTxnSessionFinalize()` API must be called to free the memory.

Library Name:
ClCorClient

Related Function(s):
clCorTxnSessionCancel.

3.7.14 clCorBundleObjectGet

clCorBundleObjectGet

Synopsis:
Populates a bundle with read-jobs.

Header File:
clCorUtilityApi.h

Syntax:

```
   ClRcT  clCorBundleObjectGet (
CL_IN    ClCorBundleT bundleHandle,
CL_IN    ClCorObjHandleT  *objHandle,
CL_INOUT   ClCorAttrValueDescriptorListPtrT          pAttrDescList
    );
```

Parameters:

bundleHandle: (in) This parameter identifies the bundle.

objHandle: (in) This parameter contains the description of the job that needs to be operated.

pAttrDescList: (in/out) This parameter describes the list of attributes on which read operation needs to be performed. `ObjHandle` and `attrDescList` together form a read-job.

Return values:

CL_OK: The API executed successfully. The Job was successfully queued in the bundle.

CL_COR_ERR_NO_MEM: This job is not added to the bundle. Bundle unsuccessful due to insufficient memory.

CL_COR_ERR_NULL_PTR: `pAttrDescList` or `objHandle` is NULL or the data buffer corresponding to any of the attribute is NULL. This job is not added into the bundle.

CL_COR_ERR_INVALID_PARAM: `pAttrDescList->numOfDescriptor` is NULL.

CL_COR_ERR_INVALID_HANDLE: Invalid bundle handle. This job is not added to the bundle.

CL_COR_ERR_BUNDLE_IN_EXECUTION: The bundle specified by `<bundlehandle>` is executing at the server while a new read-job is being added at the client side.

Description:
This API populates a bundle with read-jobs. It can be called repeatedly to queue all the required read-jobs into a bundle. The API returns after queuing the jobs in the bundle. If there is a failure encountered (as indicated by the return value of this API), the jobs are not added into the bundle. The status and the data associated with each attribute (specified in attribute value descriptor) can be accessed only when the bundle execution is completed.

Library File:
ClCorClient

Related Function(s):
clCorBundleInitialize, clCorBundleApply, clCorBundleApplyAsync, clCorBundleFinalize.

3.7.15 clCorBundleApply

clCorBundleApply

Synopsis:
Submits a bundle to the COR server for execution.

Header File:
clCorUtilityApi.h

Syntax:

```
ClRcT clCorBundleApply(
CL_IN  ClCorBundleHandleT bundleHandle);
```

Parameters:
bundleHandle: (in) This parameter identifies the bundle.

Return values:
CL_OK: The API executed successfully. The Job was successfully queued in the bundle.

CL_COR_ERR_NO_MEM: This job is not added to the bundle. Bundle unsuccessful due to insufficient memory.

CL_COR_ERR_NULL_PTR: `pAttrDescList` or `objHandle` is NULL or the data buffer corresponding to any of the attribute is NULL. This job is not added into the bundle.

CL_COR_ERR_INVALID_PARAM: `pAttrDescList->numOfDescriptor` is NULL.

CL_COR_ERR_INVALID_HANDLE: Invalid bundle handle. This job is not added to the bundle.

CL_COR_ERR_BUNDLE_IN_EXECUTION: The bundle specified by `<bundlehandle>` is executing at the server while a new read-job is being added at the client side.

Description:
The API operates synchronously. This API submits the bundle to the server for execution. The application blocks till a timeout or the bundle is successfully executed. After the bundle is executed successfully, the attribute value descriptor corresponding to Read-Jobs in the bundle can be accessed. The application can then free the attribute value descriptor. The bundle cannot be applied, if this API returns any errors described in the *Return values* section.

Library File:
ClCorClient

Related Function(s):
clCorBundleObjectGet, clCorBundleApplyAsync, clCorBundleFinalize.

3.8 Object Addressing APIs

3.8.1 clCorMoIdInitialize

clCorMoIdInitialize

Synopsis:
Initializes a MOID or resets the content of an existing MOID.

Header File:
clCorUtilityApi.h

Syntax:

```
ClRcT clCorMoIdInitialize(
           CL_INOUT ClCorMOIdPtrT pMoId);
```

Parameters:
pMoId: (in/out) Pointer to an existing MOID structure.

Return values:
CL_OK: The API executed successfully.

CL_COR_ERR_NULL_PTR: `pMoid` is a NULL pointer.

Description:
This API is used to initialize the `ClCorMoId` structure. It resets the path information, if present, and initializes it to make it an empty path. It can be applied on MOIDs that have not been initialized and used before, and also on MOIDs containing the path. The empty `MoIds` can be manipulated using `clCorMoIdSet()` and `clCorMoIdAppend()`.
This function can also be called when the MoId needs to be reset and begin a fresh operation.

Note:
This API need not be called after invoking the `clCorMoIdAlloc()` API, since the latter initializes the MOID.

Library File:
ClCorClient

Related Function(s):
clCorMoIdAlloc, clCorMoIdFree, clCorMoIdTruncate, clCorMoIdSet, clCorMoIdAppend, clCorMoIdDepthGet, clCorMoIdShow, clCorMoIdToMoClassGet, clCorMoIdNameToMoIdGet, clCorMoIdToMoIdNameGet, clCorMoIdFirstInstanceGet, clCorMoIdNextSiblingGet, clCorMoIdToInstanceGet, clCorMoIdToMoClassPathGet, clCorMoIdClone, clCorMoIdCompare.

3.8.2 clCorMoIdAlloc

clCorMoIdAlloc

Synopsis:
Creates a MOID.

Header File:
clCorUtilityApi.h

Syntax:

```
ClRcT clCorMoIdAlloc(
            CL_OUT ClCorMOIdPtrT *pMoId);
```

Parameters:
pMoId: (out) Handle of the new MOID.

Return values:
CL_OK: The API executed successfully.

CL_COR_ERR_NO_MEM: Memory allocation failure.

CL_COR_ERR_NULL_PTR: `pMoid` is a NULL pointer.

Description:
This API is used as a constructor for *ClCorMOId* as it creates a MOID. It initializes the memory and returns an empty *ClCorMOId*.
By default, the values of both the instance ID and class ID is `-1`. The default depth for the MOID is 20. This value is incremented dynamically when a new entry is added. This API allocates memory and initializes the MOID structure.

Library File:
ClCorClient

Note:
The memory allocated by this API must be freed after usage, to avoid memory leaks. This memory can be freed by calling the `clCorMoIdFree()` API.

Related Function(s):
clCorMoIdInitialize, clCorMoIdAlloc, clCorMoIdFree, clCorMoIdTruncate, clCorMoIdSet, clCorMoIdAppend, clCorMoIdDepthGet, clCorMoIdShow, clCorMoIdToMoClassGet, clCorMoIdNameToMoIdGet, clCorMoIdToMoIdNameGet, clCorMoIdFirstInstanceGet, clCorMoIdNextSiblingGet, clCorMoIdToInstanceGet, clCorMoIdToMoClassPathGet, clCorMoIdClone, clCorMoIdCompare.

3.8.3 clCorMoIdFree

clCorMoIdFree

Synopsis:
Deletes the *ClCorMOId* handle.

Header File:
clCorUtilityApi.h

Syntax:

```
ClRcT clCorMoIdFree(
            CL_OUT ClCorMOIdPtrT pMoId);
```

Parameters:
pMoId: (out) Handle of *ClCorMOId*.

Return values:
CL_OK: The API executed successfully.

CL_COR_ERR_NULL_PTR: `pMoid` is a NULL pointer.

Description:
This API is used as destructor for `ClCorMOId` as it deletes the handle of the MOID. It removes the handle and frees the memory associated with it. To avoid memory leaks you must free the memory using `clCorMoIdFree()` API whenever you invoke the `clCorMoIdAlloc()` API.

Library File:
ClCorClient

Related Function(s):
clCor`MoId`Initialize, clCorMoIdAlloc, clCorMoIdTruncate, clCorMoIdSet, clCorMoIdAppend, clCorMoIdAppend, clCorMoIdDepthGet, clCorMoIdShow, clCorMoIdToMoClassGet, clCorMoIdNameToMoIdGet, clCorMoIdToMoIdNameGet, clCorMoIdFirstInstanceGet, clCorMoIdNextSiblingGet, clCorMoIdToInstanceGet, clCorMoIdToMoClassPathGet, clCorMoIdClone, clCorMoIdCompare.

3.8.4 clCorMoIdTruncate

clCorMoIdTruncate

Synopsis:
Removes the node after the specified level.

Header File:
clCorUtilityApi.h

Syntax:

```
ClRcT clCorMoIdTruncate(
            CL_INOUT ClCorMOIdPtrT pMoId,
            CL_IN ClInt16T level);
```

Parameters:
pMoId: (in/out) Handle of the MOID.

level: (in) Level to which the MOID needs to be truncated.

Return values:
CL_OK: The API executed successfully.

CL_COR_ERR_INVALID_DEPTH: The level specified is invalid.

CL_COR_ERR_NULL_PTR: `pMoid` is a NULL pointer.

Description:
This API is used to remove all the nodes and reset the MOID until a specified level is reached. The level is specified based on the depth to which it is required to return so that the operation can continue on the other sibling tree node.

Library File:
ClCorClient

Related Function(s):
clCorMoIdInitialize, clCorMoIdAlloc, clCorMoIdFree, clCorMoIdTruncate, clCorMoIdSet, clCorMoIdAppend, clCorMoIdDepthGet, clCorMoIdShow, clCorMoIdToMoClassGet, clCorMoIdNameToMoIdGet, clCorMoIdToMoIdNameGet, clCorMoIdFirstInstanceGet, clCorMoIdNextSiblingGet, clCorMoIdToInstanceGet, clCorMoIdToMoClassPathGet, clCorMoIdClone, clCorMoIdCompare.

3.8.5 clCorMoIdSet

clCorMoIdSet

Synopsis:
Sets the class type and `instanceId` at a given node or level.

Header File:
clCorUtilityApi.h

Syntax:

```
ClRcT clCorMoIdSet(
            CL_INOUT ClCorMOIdPtrT pMoId,
            CL_IN ClUint16T level,
            CL_IN ClCorClassTypeT type,
            CL_IN ClCorInstanceIdT instance);
```

Parameters:

pMoId: (in/out) Handle of MOID path.

level: (in) Level of the node.

type: (in) Class type to be set.

instance: (in) `InstanceId` to be set.

Return values:

CL_OK: The API executed successfully.

CL_COR_ERR_INVALID_DEPTH: If the level specified is invalid.

CL_COR_ERR_NULL_PTR: MOID is a NULL pointer.

CL_COR_ERR_INVALID_CLASS: Invalid class type to set.

Description:
This API is used to set the class type and the `instanceId` properties at a given node or level. This level should be less than the current depth of the MOID. This feature can be used to set the MOID for a level when it is required to operate on that part of the instance tree.

Library File:
ClCorClient

Note:
The second parameter level should be always lesser than the current depth of the MOID.

Related Function(s):
clCorMoIdInitialize, clCorMoIdAlloc, clCorMoIdFree, clCorMoIdTruncate, clCorMoIdSet, clCorMoIdAppend, clCorMoIdDepthGet, clCorMoIdShow, clCorMoIdToMoClassGet, clCorMoIdNameToMoIdGet, clCorMoIdToMoIdNameGet, clCorMoIdFirstInstanceGet, clCorMoIdNextSiblingGet, clCorMoIdToInstanceGet, clCorMoIdToMoClassPathGet, clCorMoIdClone, clCorMoIdCompare.

3.8.6 clCorMoIdAppend

clCorMoIdAppend

Synopsis:
Adds an entry to the MOID.

Header File:
clCorUtilityApi.h

Syntax:

```
ClRcT clCorMoIdAppend(
            CL_INOUT ClCorMOIdPtrT pMoId,
            CL_IN ClCorClassTypeT type,
            CL_IN ClCorInstanceIdT instance);
```

Parameters:

pMoId: (in/out) Handle of the MOID.

type: (in) Node type.

instance: (in) ID of the node instance.

Return values:

CL_OK: The API executed successfully.

CL_COR_ERR_MAX_DEPTH: The depth exceeded the maximum limit.

CL_COR_ERR_INVALID_CLASS: Invalid class type for append.

CL_COR_ERR_INVALID_PARAM: Invalid parameter is passed.

Description:
This API is used to add an entry to `ClCorMOId` containing the type and the instance. The `classId` and instance ID are appended at the end of the current MOID and the depth of the MOID is incremented.

Library File:
ClCorClient

Related Function(s):
clCorMoIdInitialize, clCorMoIdAlloc, clCorMoIdFree, clCorMoIdTruncate, clCorMoIdSet, clCorMoIdAppend, clCorMoIdDepthGet, clCorMoIdShow, clCorMoIdToMoClassGet, clCorMoIdNameToMoIdGet, clCorMoIdToMoIdNameGet, clCorMoIdFirstInstanceGet, clCorMoIdNextSiblingGet, clCorMoIdToInstanceGet, clCorMoIdToMoClassPathGet, clCorMoIdClone, clCorMoIdCompare.

3.8.7 clCorMoIdDepthGet

clCorMoIdDepthGet

Synopsis:
Returns the node depth of the COR `MoId`.

Header File:
clCorUtilityApi.h

Syntax:

```
ClInt16T clCorMoIdDepthGet(
           CL_IN ClCorMOIdPtrT pMoId);
```

Parameters:
pMoId: (in) handle of the MOID.

Parameters:
ClInt16T: The number of elements.

Description:
This API is used to return the number of nodes in the hierarchy within the COR MOID.

Library File:
ClCorClient

Related Function(s):
cl Cor MoId Initialize, clCorMoIdAlloc, clCorMoIdFree, clCorMoIdTruncate, clCorMoIdSet, clCorMoIdAppend, clCorMoIdShow clCorMoIdToMoClassGet, clCorMoIdNameToMoIdGet, clCorMoIdToMoIdNameGet, clCorMoIdFirstInstanceGet, clCorMoIdNextSiblingGet, clCorMoIdToInstanceGet, clCorMoIdToMoClassPathGet, clCorMoIdClone, clCorMoIdCompare.

3.8.8 clCorMoIdShow

clCorMoIdShow

Synopsis:
Displays the *ClCorMOId* handle.

Syntax:

```
void clCorMoIdShow(
            CL_IN ClCorMOIdPtrT pMoId);
```

Parameters:
pMoId: (in) Handle of the MOID.

Return values:
None.

Description:
This API is used to display all the entries within the COR MOID. This function displays the current active nodes (classId:instanceId format) in the MOID and its service IDs.

Related Function(s):
clCorMoIdInitialize, clCorMoIdAlloc, clCorMoIdFree, clCorMoIdTruncate, clCorMoIdSet, clCorMoIdAppend, clCorMoIdDepthGet, clCorMoIdShow, clCorMoIdToMoClassGet, clCorMoIdNameToMoIdGet, clCorMoIdToMoIdNameGet, clCorMoIdFirstInstanceGet, clCorMoIdNextSiblingGet, clCorMoIdToInstanceGet, clCorMoIdToMoClassPathGet, clCorMoIdClone, clCorMoIdCompare.

3.8.9 clCorMoIdClone

clCorMoIdClone

Synopsis:
Clones a particular MOID.

Header File:
clCorUtilityApi.h

Syntax:

```
ClRcT clCorMoIdClone(
                        CL_IN ClCorMOIdPtrT pMoId,
                        CL_OUT ClCorMOIdPtrT* newH);
```

Parameters:

pMoId: (in) Handle of the `MoId`.

newH: (out) Handle of the new clone.

Return values:

CL_OK: The API executed successfully.

CL_COR_ERR_NULL_PTR: `pMoId` or `newH` is a NULL pointer.

CL_COR_ERR_NO_MEM: Memory allocation failure.

Description:
This API is used to clone a particular MOID. It allocates and copies the contents of the given MOID to a new MOID.

Library File:
ClCorClient

Related Function(s):
clCorMoIdInitialize, clCorMoIdAlloc, clCorMoIdFree, clCorMoIdTruncate, clCorMoIdSet, clCorMoIdAppend, clCorMoIdDepthGet, clCorMoIdShow, clCorMoIdToMoClassGet, clCorMoIdNameToMoIdGet, clCorMoIdToMoIdNameGet, clCorMoIdFirstInstanceGet, clCorMoIdNextSiblingGet, clCorMoIdToInstanceGet, clCorMoIdToMoClassPathGet, clCorMoIdClone, clCorMoIdCompare.

3.8.10 clCorMoIdCompare

clCorMoIdCompare

Synopsis:
Compares two `MoIds` and verifies if they are equal.

Header File:
clCorUtilityApi.h

Syntax:

```
ClInt32T clCorMoIdCompare(
                    CL_IN ClCorMOIdPtrT pMoId,
                    CL_IN ClCorMOIdPtrT cmp);
```

Parameters:
pMoId: (in) Handle of the first `ClCorMOId`.

cmp: (in) Handle of the second `ClCorMOId`.

Return values:
0: Both COR MOIDs match each other.

-1: MOIDs do not match.

1: The wildcards of MOIDs match each other.

Description:
This API is used to compare two `ClCorMOId` and verify if they are equal. This comparison is performed till the depth specified in the MOID is reached.

Library File:
ClCorClient

Related Function(s):
clCorMoIdInitialize, clCorMoIdAlloc, clCorMoIdFree, clCorMoIdTruncate, clCorMoIdSet, clCorMoIdAppend, clCorMoIdDepthGet, clCorMoIdShow, clCorMoIdToMoClassGet, clCorMoIdNameToMoIdGet, clCorMoIdToMoIdNameGet, clCorMoIdFirstInstanceGet, clCorMoIdNextSiblingGet, clCorMoIdToInstanceGet, clCorMoIdToMoClassPathGet, clCorMoIdClone, clCorMoIdCompare.

3.9 Object Search APIs

3.9.1 clCorMoIdFirstInstanceGet

clCorMoIdFirstInstanceGet

Synopsis:
Returns the first child instance.

Header File:
clCorUtilityApi.h

Syntax:

```
ClRcT clCorMoIdFirstInstanceGet(
                        CL_INOUT ClCorMOIdPtrT pMoId);
```

Parameters:
pMoId: (in/out) The updated `ClCorMOId` is returned. The MOID pointed by the `pMoId` should be allocated before calling this API.

Return values:
CL_OK: The API executed successfully.

CL_COR_ERR_NULL_PTR: pMoId is a NULL pointer.

CL_COR_ERR_INVALID_PARAM: pMoId contains an invalid service ID.

CL_COR_MO_TREE_ERR_NODE_NOT_FOUND: The node is not found in the MO tree.

CL_COR_INST_ERR_NODE_NOT_FOUND: The node is not found in the object instance tree.

Description:
This API is used to retrieve the first child instance from the MOID tree. On successful execution, it updates the COR MOID.

Library File:
ClCorClient

Note:
The last node class tag must be completed and `instanceId` must be left at zero which will be updated by this API.

Related Function(s):
clCorMoIdInitialize, clCorMoIdAlloc, clCorMoIdFree, clCorMoIdTruncate, clCorMoIdSet, clCorMoIdAppend, clCorMoIdDepthGet, clCorMoIdShow, clCorMoIdToMoClassGet, clCorMoIdNameToMoIdGet, clCorMoIdToMoIdNameGet, clCorMoIdFirstInstanceGet, clCorMoIdNextSiblingGet, clCorMoIdToInstanceGet, clCorMoIdToMoClassPathGet, clCorMoIdClone, clCorMoIdCompare.

3.9.2 clCorMoIdNextSiblingGet

clCorMoIdNextSiblingGet

Synopsis:
Returns the next sibling.

Header File:
clCorUtilityApi.h

Syntax:

```
ClRcT clCorMoIdNextSiblingGet(
                        CL_INOUT ClCorMOIdPtrT pMoId);
```

Parameters:

pMoId: (in/out) The updated `ClCorMOId` is returned. You must allocate the memory for the MOID.

Return values:

CL_OK: The API executed successfully.

CL_COR_ERR_NULL_PTR: `pMoId` is a NULL pointer.

CL_COR_ERR_INVALID_PARAM: `pMoId` contains an invalid service ID.

CL_COR_MO_TREE_ERR_NODE_NOT_FOUND: The node is not found in the MO tree.

CL_COR_INST_ERR_NODE_NOT_FOUND: The node is not found in the object instance tree.

Description:
This API is used to return the next sibling from the MOID tree. For a given MOID, the COR server finds next sibling of the MO, and returns the class and instance of the MO.

Library File:
ClCorClient

Related Function(s):
clCorMoIdInitialize, clCorMoIdAlloc, clCorMoIdFree, clCorMoIdTruncate, clCorMoIdSet, clCorMoIdAppend, clCorMoIdDepthGet, clCorMoIdShow, clCorMoIdToMoClassGet, clCorMoIdNameToMoIdGet, clCorMoIdToMoIdNameGet, clCorMoIdFirstInstanceGet, clCorMoIdNextSiblingGet, clCorMoIdToInstanceGet, clCorMoIdToMoClassPathGet, clCorMoIdClone, clCorMoIdCompare.

3.9.3 clCorObjectWalk

clCorObjectWalk

Synopsis:
Walks through the object tree.

Header File:
clCorApi.h

Syntax:

```
ClRcT clCorObjectWalk(
                          CL_IN ClCorMOIdPtrT moIdRoot,
                          CL_IN ClCorMOIdPtrT moIdFilter,
                          CL_IN ClCorObjectWalkFunT fp,
                          CL_IN ClCorObjWalkFlagsT flags,
                          CL_IN void * cookie);
```

Parameters:

moIdRoot: (in) MOID from where the walk operation begins.

moIdFilter: (in) MOID with wild card entry. This filter is used to refine the scope of the search. For example, if the `moIdRoot` is `/chassis:0/blade:1/`, `moIdFilter` can have a value such as `/ chassis:0 / blade:1 / port:* / channel:* /`

fp: (in) The callback API invoked for every object found in COR.

flags: (in) Flags indicating walk related definitions. This parameter accepts the following values:

- `CL_COR_MO_WALK` - The walk is performed on the MO objects of the object tree.
- `CL_COR_MSO_WALK` - The walk is performed on the MSO nodes of the object tree.
- `CL_COR_MO_SUBTREE_WALK` - The walk is performed on the MO objects of the object subtree.

cookie: (in) Pointer to the user-defined data. This value is passed as a parameter to the user callback API.

Return values:

CL_OK: The API executed successfully.

CL_COR_ERR_NULL_PTR: `moIdRoot` or `moIdFilter` is a NULL pointer.

CL_COR_SVC_ERR_INVALID_ID: Service ID is invalid.

CL_COR_ERR_VERSION_UNSUPPORTED: Version is not supported.

CL_COR_ERR_NO_MEM: Memory allocation failed.

Description:

This API is used to perform a walk on the COR objects in the object tree. The first parameter is the MOID of the root, from where the object walk begins. If the walk has to be performed on the entire tree, this parameter must be set to NULL.
`moIdFilter`, contains the filter for the search. This parameter can contain a wild card entry or a MOID. It must be set to NULL if a filter is not required for the object walk.
fp is a callback API that has two parameters: `data` and `cookie`. `data` contains the handle to the object. `cookie` contains the cookie passed in the object walk API.
flags indicate the type of object walk that is required.
The final parameter is the user-argument cookie.

Library File:
ClCorClient

Note:
The cookie parameter can be used to pass the value used in the callback API. If `moIdRoot` and `moIdFilter` are NULL, all the MO/MSO objects are walked through irrespective of any specification in the flag value.

Related Function(s):
clCorObjectAttributeWalk, clCorMoIdAppend.

3.9.4 clCorObjectAttributeWalk

clCorObjectAttributeWalk

Synopsis:
Walk is performed on the attributes of the object.

Header File:
clCorApi.h

Syntax:

```
   ClRcT clCorObjectAttributeWalk(
CL_IN ClCorObjecthandleT objH,
CL_IN ClCorObjAttrWalkFilterT *pFilter,
CL_IN ClCorObjAttrWalkFuncT fp,
CL_IN void * cookie);
```

Parameters:

objH: (in) handle of the object.

pFilter: (in) Pointer to the attribute filter. You have to provide the values for this structure.

fp: (in) User callback API called for every attribute found.

cookie :(in) The user data passed to the user callback API.

Return values:

CL_OK: The API executed successfully.

CL_COR_ERR_INVALID_PARAM: A parameter is invalid.

CL_COR_ERR_NULL_PTR: `pFilter` or `cookie` is a NULL pointer.

CL_COR_ERR_NO_MEM: Failed to allocate memory.

CL_COR_ERR_NOT_SUPPORTED: `cmp_flag` is not invalid.

CL_COR_ERR_VERSION_UNSUPPORTED: Client version not supported.

CL_COR_UTILS_ERR_INVALID_KEY: On passing an invalid parameter.

CL_COR_INST_ERR_INVALID_MOID: MoId is invalid.

pFilter Usage:

pFilter is used to apply filters to the walk criteria. It can have the following values:

- `pFilter = NULL`: No filter is applied and it walks through all the attributes of the object.
- `pFilter>baseAttrWalk = CL_TRUE`: Walks through the attributes of the base object (native attributes of the object pointed by objH).
- `pFilter>baseAttrWalk = CL_FALSE`: Does not walk through the native attributes.
- `pFilter>contAttrWalk`: For the current implementation, this must be set to `CL_FALSE`.
- `pFilter>pAttrPath`: This can be set to a valid attribute path or NULL. For the current implementation, this must be set to NULL.
- `pFilter>cmpFlag`: This is used to compare the `attrId` with a specified value. Following are the various comparison flags.
  - `CL_COR_ATTR_CMP_FLAG_VALUE_EQUAL_TO`: The attributes whose value is equal to the specified value are matched.

  - CL_COR_ATTR_CMP_FLAG_VALUE_LESS_THAN: The attributes whose value is greater than the specified value are matched.
  - CL_COR_ATTR_CMP_FLAG_VALUE_LESS_OR_EQUALS: The attributes whose value is greater than or equal to the specified value are matched.
  - CL_COR_ATTR_CMP_FLAG_VALUE_GREATER_THAN: The attributes whose value is less than the specified value are matched.
  - CL_COR_ATTR_CMP_FLAG_VALUE_GREATER_OR_EQUALS: The attributes whose value is less than or equal to the specified value are matched.
- pFilter>attrWalkOption This can have one of the following values:
  - CL_COR_ATTR_WALK_ALL_ATTR: All the attributes of contained (and/or base object, depending on baseAttrWalk parameter) object are considered for walk, provided the pFilter->cmpFlag condition is true.
  - CL_COR_ATTR_WALK_ONLY_MATCHED_ATTR: Only the attributes matching with the filter criteria are walked provided the pFilter->cmpFlag condition goes true.
- pFilter>pValue: Pointer to the value of the attribute
- pFilter>size: The size of the value to be compared.
- pFilter>index: For a SIMPLE attribute, index ID is set to CL_COR_INVALID_ATTR_IDX.

Description:

This API is used to perform a walk on attributes of the object based on the filter. This API takes the object handle as the first parameter. The handle is obtained from the MOID using the clCorObjecthandleGet() API.
The second parameter is the filter for the attribute walk. If the filter is NULL, the walk is performed on all attributes of the MO. The first element of the filter baseAttrWalk, must always be set to CL_TRUE.
The parameter contAttrWalk must be set to CL_TRUE. pAttrPath must be set to NULL. The callback function should be specified when the object walk API is invoked. This callback function is called for every attribute found during the walk. The cookie (user-data) is passed as the parameter to the callback function.

Library File:

ClCorClient

Note:

The last parameter can be used to pass the value used by the callback API. If the first and second parameters are NULL, all the MO/MSO objects are walked through irrespective of any specification in the flag.

Related Function(s):

clCorObjecthandleGet, clCorMoIdCompare.

3.10 Moid Manipulation APIs

3.10.1 clCorMoIdToClassGet

clCorMoIdToClassGet

Synopsis:
Returns the class type.

Header File:
clCorUtilityApi.h

Syntax:

```
ClRcT clCorMoIdToClassGet(
                        CL_IN ClCorMOIdPtrT pMoId,
                        CL_IN ClCorMoIdClassGetFlagsT flag,
                        CL_OUT ClCorClassTypeT *pClassId);
```

Parameters:

pMoId: (in) Handle of the MOID.

flag: (in) Specifies the type of MO class to be returned. The value of the flag can be `CL_COR_MO_CLASS_GET` or `CL_COR_MSO_CLASS_GET`.

pClassID: (out) The `classId` of the class.

Return values:

CL_OK: The API executed successfully.

CL_COR_ERR_NULL_PTR: `pMoId` or `pClassId` is a NULL pointer.

CL_COR_ERR_CLASS_INVALID_PATH: The MO-path specified in the `pMoId` is invalid.

CL_COR_SVC_ERR_INVALID_ID: The service ID specified in the `pMoId` is invalid.

CL_COR_ERR_INVALID_PARAM: The flag specified is invalid.

Description:
This API is used to return the class type within the COR MOID. It refers to the class type at the end of the hierarchy.

Library File:
ClCorClient

Related Function(s):
clCorMoIdInitialize, clCorMoIdAlloc, clCorMoIdFree, clCorMoIdTruncate, clCorMoIdSet, clCorMoIdAppend, clCorMoIdDepthGet, clCorMoIdShow, clCorMoIdToMoClassGet, clCorMoIdNameToMoIdGet, clCorMoIdToMoIdNameGet, clCorMoIdFirstInstanceGet, clCorMoIdNextSiblingGet, clCorMoIdToInstanceGet, clCorMoIdToMoClassPathGet, clCorMoIdClone, clCorMoIdCompare.

3.10.2 clCorMoIdNameToMoIdGet

clCorMoIdNameToMoIdGet

Synopsis:

Retrieves the MOID in `ClCorMOIdT` format, when MOID is provided in `ClNameT` format.

Syntax:

```
ClRcT clCorMoIdNameToMoIdGet
                        CL_IN ClNameT *moIdName,
                        CL_OUT ClCorMOIdT *moId);
```

Parameters:

MoIdName: (in) MOID in string format.

MoId: (out) Returns the MOID. It has to be allocated by the user.

Return values:

CL_OK: The API executed successfully.

CL_COR_ERR_NULL_PTR: `moIdName` or `moId` is a NULL pointer.

Description:

This API is used to retrieve the MOID in `ClCorMOIdT` format, when the MOID is provided in `ClNameT` format.

Related Function(s):

clCorMoIdInitialize, clCorMoIdAlloc, clCorMoIdFree, clCorMoIdTruncate, clCorMoIdSet, clCorMoIdAppend, clCorMoIdDepthGet, clCorMoIdShow, clCorMoIdToMoClassGet, clCorMoIdNameToMoIdGet, clCorMoIdToMoIdNameGet, clCorMoIdFirstInstanceGet, clCorMoIdNextSiblingGet, clCorMoIdToInstanceGet, clCorMoIdToMoClassPathGet, clCorMoIdClone, clCorMoIdCompare.

3.10.3 clCorMoIdToMoIdNameGet

clCorMoIdToMoIdNameGet

Synopsis:
Retrieves `MoId` in `ClNameT` format, when `MoId` is provided in `ClCorMOIdT` format.

Header File:
clCorUtilityApi.h

Syntax:

```
ClRcT clCorMoIdToMoIdNameGet(
                        CL_IN ClCorMOIdT *moId,
                        CL_OUT ClNameT *moIdName);
```

Parameters:
MoId: (in) Structure of the MOID.

MoIdName: (out) MOID in string format.

Return values:
CL_OK: The API executed successfully.

CL_COR_ERR_NULL_PTR: `moIdName` or `moId` is a NULL pointer.

Description:
This API is used to retrieve MOID in `ClNameT` format, when MOID is provided in `ClCorMOIdT` format.

Library File:
ClCorClient

Related Function(s):
clCorMoIdInitialize, clCorMoIdAlloc, clCorMoIdFree, clCorMoIdTruncate, clCorMoIdSet, clCorMoIdAppend, clCorMoIdDepthGet, clCorMoIdShow, clCorMoIdToMoClassGet, clCorMoIdNameToMoIdGet, clCorMoIdToMoIdNameGet, clCorMoIdFirstInstanceGet, clCorMoIdNextSiblingGet, clCorMoIdToInstanceGet, clCorMoIdToMoClassPathGet, clCorMoIdClone, clCorMoIdCompare.

3.10.4 clCorMoIdToInstanceGet

clCorMoIdToInstanceGet

Synopsis:
Returns the instance.

Header File:
clCorUtilityApi.h

Syntax:

```
ClCorInstanceIdT clCorMoIdToInstanceGet(
                        CL_IN ClCorMOIdPtrT pMoId);
```

Parameters:
pMoId: (in) Handle of the MOID.

Return values:
ClCorInstanceIDT: Associated instance ID.

Description:
This API is used to return the instance that is queried. It refers to the class type and instance ID at the end of the hierarchy.

Library File:
ClCorClient

Related Function(s):
clCorMoIdInitialize, clCorMoIdAlloc, clCorMoIdFree, clCorMoIdTruncate, clCorMoIdSet, clCorMoIdAppend, clCorMoIdDepthGet, clCorMoIdShow, clCorMoIdToMoClassGet, clCorMoIdNameToMoIdGet, clCorMoIdToMoIdNameGet, clCorMoIdFirstInstanceGet, clCorMoIdNextSiblingGet, clCorMoIdToInstanceGet, clCorMoIdToMoClassPathGet, clCorMoIdClone, clCorMoIdCompare.

3.10.5 clCorMoIdToMoClassPathGet

clCorMoIdToMoClassPathGet

Synopsis:
Derives the COR path from a given MOID.

Header File:
clCorUtilityApi.h

Syntax:

```
ClRcT clCorMoIdToMoClassPathGet(
                CL_IN ClCorMOIdPtrT moIdh,
                CL_OUT ClCorMOClassPathPtrT corIdh);
```

Parameters:
MoIdh: (in) Handle of the MOID.

corIdh: (out) Handle of the updated COR path.

Return values:
CL_OK: The API executed successfully.

CL_COR_ERR_NULL_PTR: `moIdh` or `corIdh` is a NULL pointer.

Description:
This function is used to obtain the MO-path from the MOID. You are required to allocate memory for both MOID and COR path.

Library File:
ClCorClient

Related Function(s):
clCorMoIdInitialize, clCorMoIdAlloc, clCorMoIdFree, clCorMoIdTruncate, clCorMoIdSet, clCorMoIdAppend, clCorMoIdDepthGet, clCorMoIdShow, clCorMoIdToMoClassGet, clCorMoIdNameToMoIdGet, clCorMoIdToMoIdNameGet, clCorMoIdFirstInstanceGet, clCorMoIdNextSiblingGet, clCorMoIdToInstanceGet, clCorMoIdToMoClassPathGet, clCorMoIdClone, clCorMoIdCompare.

3.10.6 clCorMoIdServiceGet

clCorMoIdServiceGet

Synopsis:
Returns the service ID.

Header File:
clCorUtilityApi.h

Syntax:

```
ClCorMOServiceIdT clCorMoIdServiceGet(
                  CL_IN ClCorMOIdPtrT pMoId);
```

Parameters:
pMoId: (in) Handle of the MOID.

Return Values:
It returns the service ID through `ClCorMOServiceIDT`.

Description:
This API is used to return the service ID associated with the COR MOID.

Library File:
ClCorClient

Related Function(s):
clCorMoIdServiceSet.

3.10.7 clCorMoIdServiceSet

clCorMoIdServiceSet

Synopsis:
Sets the service ID.

Header File:
clCorUtilityApi.h

Syntax:

```
ClRcT clCorMoIdServiceSet(
                        CL_INOUT ClCorMOIdPtrT pMoId,
                        CL_IN ClCorMOServiceIdT svc);
```

Parameters:
pMoId: (in/out) Handle of the MOID.

svc: (in) Service ID to be set.

Return values:
CL_OK: The API executed successfully.

CL_COR_ERR_NULL_PTR: `pMoId` is a NULL pointer.

CL_COR_ERR_INVALID_MSP_ID: The service ID is invalid.

Description:
This API is used to set the service ID for a particular COR MOID.

Library File:
ClCorClient

Related Function(s):
clCorMoIdServiceGet, clCorServiceIDValidate.

3.10.8 clCorMoIdInstanceSet

clCorMoIdInstanceSet

Synopsis:
Sets the instance of the MOID.

Header File:
clCorUtilityApi.h

Syntax:

```
ClRcT clCorMoIdInstanceSet(
                        CL_INOUT ClCorMOIdPtrT pMoId,
                        CL_IN ClUint16T ndepth,
                        CL_IN ClCorInstanceIdT newInstance);
```

Parameters:

pMoId: (in/out) Handle of the MOID.

ndepth: (in) Depth at which the instance is to be set.

newInstance: (in) The instance of the MOID that needs to be set.

Return values:

CL_OK: The API executed successfully.

CL_COR_INST_ERR_INVALID_MOID: The `ndepth` value is greater than the depth of the MOID.

Description:
This API is used to set the instance field to a specified depth of the `MoId`.

Library File:
ClCorClient

Related Function(s):
clCorMoIdFirstInstanceGet.

3.10.9 clCorMoIdConcatenate

clCorMoIdConcatenate

Synopsis:
Concatenates a MOID to another MOID.

Header File:
clCorUtilityApi.h

Syntax:

```
ClRcT clCorMoIdConcatenate(
                        CL_INOUT ClCorMOIdPtrT moid1,
                        CL_INOUT ClCorMOIdPtrT moid2,
                        CL_IN ClInt32T copyWhere);
```

Parameters:

moid1: (in/out) First MOID.

moid2: (in/out) Second MOID.

copyWhere: (in) Indicates where the concatenated MOID is to be copied.

If 0, *moid2* is concatenated with *moid1* and the result is stored in *moid1*.

If 1, *moid1* is concatenated with *moid2* and the result is stored in *moid2*.

Return values:

CL_OK: The API executed successfully.

CL_COR_ERR_MAX_DEPTH: The depth exceeds the maximum limit.

CL_COR_ERR_NULL_PTR: `moId1` or `moId2` is a NULL pointer.

CL_COR_ERR_MAX_DEPTH: The depth of `(moId1 + moId2)` exceeds the maximum depth of the MOID.

Description:
This API is used to concatenate two MOIDs.

Library File:
ClCorClient

Related Function(s):
clCorMoIdAppend, clCorMoIdTruncate.

3.11 COR-Event APIs

3.11.1 clCorEventSubscribe

clCorEventSubscribe

Synopsis:
Subscribes for notifications when a change occurs in an attribute.

Header File:
clCorNotifyApi.h

Syntax:

```
ClRcT clCorEventSubscribe(
                          CL_IN ClEvtChannelhandleT channelhandle,
                          CL_IN ClCorMOIdPtr changedObj,
                          CL_IN ClCorAttrPathPtrT  pAttrPath,
                          CL_IN ClCorAttrListPtr attrList,
                          CL_IN ClCorOpsT    flags,
                          CL_IN void * cookie,
                          CL_IN ClEvtSubscriptionIdT subscriptionId);
```

Parameters:

channelhandle: (in) Handle of the COR channel. You are required to allocate the memory for this parameter.

changedObj: (in) This is the complete path to the object. Wildcards can be used to specify a *class* or *subtree* of objects.

pAttrPath: (in) To subscribe for the attributes of the contained objects, the `attrPath` identifying the contained object must be specified. It must be set to NULL for the current implementation and will be supported in future releases.

attrList: (in) If the subscriber is interested in receiving notifications when certain attribute(s) of an object change, the list of these attribute IDs can be specified here. If it is set to NULL, the subscriber receives notifications when a change occurs in any of the attributes of the specified MO.

Following are the critical usage restrictions for this parameter:

- This parameter is interpreted only if the parameter flags contains one or more `_SET_` operations. Refer to 3.5.5 for the possible operations types.
- Specifying the `attrList` implies that you are interested in the changes on the attribute level.
- The service ID must not contain wildcard entries.
- The class value in `changedObj` (which is a MOID) must not contain wildcard entries.
- Wildcard entries can be used to specify instance value in `changedObj`.

flags: (in) This parameter contains the operations that the user can subscribe for. The operations are:

`CREATE`

`DELETE`

`SET`

cookie: (in) This contains the user-data which is passed as the parameter to the notification callback function.

subscriptionID: (in) You are required to provide the subscription ID. This subscription ID has to be used while unsubscribing.

Return values:

CL_OK: The API executed successfully.

CL_COR_ERR_NULL_PTR: `changedObj` is a NULL pointer.

CL_COR_NOTIFY_ERR_INVALID_OP: The operation specified for the subscription is not valid.

CL_EVENT_ERR_BAD_HANDLE: `channelhandle` is zero.

Description:

This API is used to subscribe for CREATE, SET, and DELETE operations on a particular MO, specified by changedObject. When these operations are performed on the MO, the notification callback of the subscriber is invoked. The event handle is passed as the parameter to the notification callback. The clCorEventHandleToCorTxnIdGet() API can be used to retrieve the transaction-ID specific to COR. This transaction-ID can be used to obtain information about the jobs by using the clCorTxnJobMoIdGet(), clCorTxnJobSetParamsGet(), clCorTxnJobOperationGet() and clCorTxnJobWalk() APIs.

Library File:

ClCorClient

Related Function(s):

clCorEventUnsubscribe.

3.11.2 clCorEventUnsubscribe

clCorEventUnsubscribe

Synopsis:
Un subscribes for attribute change notification.

Header File:
clCorNotifyApi.h

Syntax:

```
ClRcT clCorEventUnsubscribe(
                            CL_IN ClEvtChannelhandleT channelhandle,
                            CL_IN ClEvtSubscriptionIdT subscriptionId);
```

Parameters:

channelhandle: (in) Channel handle obtained when the COR channel was opened by the application.

subscriptionID: (in) The Subscription ID obtained from COR while *subscribing*.

Return values:

Return values:

CL_OK: The API executed successfully.

CL_EVENT_ERR_INIT_NOT_DONE: Event library is not initialized.

CL_EVENT_ERR_BAD_HANDLE: The handle is invalid.

CL_EVENT_INTERNAL_ERROR: An unexpected problem occurred with the Event Manager.

CL_EVENT_ERR_INVALID_PARAM: On passing an invalid parameter.

CL_EVENT_ERR_NO_MEM: Memory allocation failure.

Description:
This API is used to unsubscribe an event subscribed previously. The subscription ID obtained while subscribing for the event must be used to unsubscribe the event.

Library File:
ClCorClient

Related Function(s):
clCorEventSubscribe.

3.11.3 clCorEventHandleToCorTxnIdGet

clCorEventHandleToCorTxnIdGet

Synopsis:
Obtains the transaction ID from the event handle.

Header File:
clCorUtilityApi.h

Syntax:

```
   ClRcT clCorEventHandleToCorTxnIdGet(
CL_IN  ClEventHandleT  evtH,
CL_IN  ClSizeT  size,
CL_OUT ClCorTxnIdT* corTxnId);
```

Parameters:

evtH: (in) Handle to the event .

size: (in) Size of the event data.

corTxnId: (out) COR transaction Id extracted from the event.

Return values:

CL_OK: The API executed successfully.

CL_COR_ERR_INVALID_PARAM: Invalid parameter passed.

CL_COR_ERR_NO_MEM: Memory allocation failure.

CL_COR_TXN_ERR_ZERO_JOBS: No jobs found in the transaction.

Description:
This function is used to GET the transaction ID from the event handle that is passed as a parameter to the Event Deliver Callback function. The user can subscribe for the changes on a MO using the function `clCorEventSubscribe()`. The Event Deliver Callback function is called when a change occurs on that MO. The transaction ID extracted from the event handle can be used to retrieve the information about the jobs in that transaction using the `clCorTxnJobMoIdGet() / clCorTxnJobSetParamsGet() / clCorTxnJobOperationGet()` functions.

Library File:
ClCorClient

Note:

This function allocates memory for `corTxnId` when it is successfully executed. You must use the function `clCorTxnIdTxnFree()` with `corTxnId` as the parameter to free this memory.

Related Function(s):
clCorEventSubscribe, clCorTxnIdTxnFree, clCorTxnJobMoIdGet, clCorTxnJobSetParamsGet, clCorTxnJobOperationGet, clCorTxnJobWalk.

3.11.4 clCorTxnIdTxnFree

clCorTxnIdTxnFree

Synopsis:
Frees the data from the transaction ID.

Header File:
clCorUtilityApi.h

Syntax:

```
   ClRcT clCorTxnIdTxnFree(
CL_IN    ClCorTxnIdT    corTxnId);
```

Parameters:
corTxnId: (in) COR transaction ID.

Return values:
CL_OK: The API executed successfully.

CL_COR_ERR_NULL_PTR: `corTxnId` contains a NULL pointer.

CL_COR_TXN_ERR_ZERO_JOBS: No jobs found in the transaction.

Description:
This function is used to free the data corresponding to the COR Transaction ID, `corTxnId`. This function must be called after the transaction ID is extracted from the event handle using `clCorEventHandleToCorTxnIdGet()`.

Library File:
ClCorClient

Related Function(s):
clCorEventHandleToCorTxnIdGet.

3.11.5 clCorTxnJobWalk

clCorTxnJobWalk

Synopsis:
Performs the walk operation through the transaction jobs.

Header File:
clCorUtilityApi.h

Syntax:

```
   ClRcT   clCorTxnJobWalk(
CL_IN ClCorTxnIdT  txnId,
CL_IN ClCorTxnFuncT funcPtr,
CL_IN void  *cookie);
```

Parameters:

txnId: (in) COR transaction ID.

funcPtr: (in) Callback function that is called for each job found during the walk operation.

****cookie:*** (in) User-data that is passed as a parameter to the callback function.

Return values:

CL_OK: The API executed successfully.

CL_COR_TXN_ERR_ZERO_JOBS: No jobs found during the walk operation.

CL_COR_ERR_NULL_PTR: `funcPtr` or `txnId` contains a NULL pointer.

Description:
This function is used to perform the walk operation through the transaction jobs. The callback function is called every job found in the transaction. The transaction ID, Job ID and the user argument, cookie, are passed as parameters to the callback function.

Library File:
ClCorClient

Related Function(s):
clCorTxnJobMoIdGet, clCorTxnJobSetParamsGet, clCorTxnJobOperationGet, pagecor205.

3.11.6 clCorTxnJobMoIdGet

clCorTxnJobMoIdGet

Synopsis:
Retrieves the MOID from the transaction job.

Header File:
clCorUtilityApi.h

Syntax:

```
   ClRcT  clCorTxnJobMoIdGet(
CL_IN     ClCorTxnIdT  txnId,
CL_OUT    ClCorMOIdT  *pMOId);
```

Parameters:

txnId: (in) COR transaction ID.

pMOId: (out) MoId in the transaction.

Return values:

CL_OK: The API executed successfully.

CL_COR_ERR_NULL_PTR: `pMOId` contains a NULL pointer.

CL_COR_TXN_ERR_ZERO_JOBS: No jobs found in the transaction.

Description:
This function is used to GET the MOID from the transaction job. The parameter `txnId` contains the ID for the transaction. The MOID is returned in the OUT parameter `pMOId`.

Library File:
ClCorClient

Related Function(s):
clCorTxnJobSetParamsGet, clCorTxnJobOperationGet, clCorTxnJobWalk.

3.11.7 clCorTxnJobSetParamsGet

clCorTxnJobSetParamsGet

Synopsis:
Retrieves the information of the SET job in the transaction.

Header File:
clCorUtilityApi.h

Syntax:

```
   ClRcT clCorTxnJobSetParamsGet(
CL_IN   ClCorTxnIdT          txnId,
CL_IN   ClCorTxnJobIdT       jobId,
CL_OUT  ClCorAttrIdT        *pAttrId,
CL_OUT  ClInt32T            *pIndex,
CL_OUT  void               **pValue,
CL_OUT  ClUint32T           *pSize);
```

Parameters:

txnId: (in) COR transaction ID.

jobId: (in) Transaction job ID.

****pAttrId:*** (out) Attribute ID.

****pIndex:*** (out) Index of the attribute.

****pValue:*** (out) Value of the attribute.

****pSize:*** (out) Size of the attribute.

Return values:

CL_OK: The API executed successfully.

CL_COR_ERR_NULL_PTR: `pAttrId`, `pIndex`, `pValue`, `pSize` contains a NULL pointer.

CL_COR_ERR_INVALID_PARAM: `pMOId` is invalid.

CL_COR_TXN_ERR_INVALID_JOB_ID: `jobId` is invalid.

Description:
This function is used to extract the parameters of the attribute from the transaction SET job. The Transaction ID and Job ID should be passed as the parameters to the function. The OUT parameter 'pIndex' contains the index of the attribute. This value will be CL_COR_INVALID_ATTR_IDX in case of SIMPLE attributes.

Library File:
ClCorClient

Note:
This function can be used only when the operation type of the job is SET.

Related Function(s):
clCorTxnJobMoIdGet, clCorTxnJobOperationGet, clCorTxnJobWalk.

3.11.8 clCorTxnJobOperationGet

clCorTxnJobOperationSet

Synopsis:
Retrieves the operation type of the Job.

Header File:
clCorUtilityApi.h

Syntax:

```
   ClRcT  clCorTxnJobOperationGet(
CL_IN     ClCorTxnIdT    txnId,
CL_IN     ClCorTxnJobIdT jobId,
CL_OUT    ClCorOpsT      *op);
```

Parameters:

txnId: (in) COR transaction ID.

jobId: (in) Transaction job ID.

****op:*** (out) Type of the operation. It can be CL_COR_OP_CREATE, CL_COR_OP_SET, or CL_COR_OP_DELETE.

Return values:

CL_OK: The API executed successfully.

CL_COR_ERR_NULL_PTR: `op` contains a NULL pointer.

CL_COR_ERR_INVALID_PARAM:

Description:
This function is used to extract the operation type from the transaction job. The transaction ID and job ID should be passed as the parameters to this function.

Library File:
ClCorClient

Related Function(s):
clCorTxnJobMoIdGet, clCorTxnJobSetParamsGet.

3.12 OI Related APIs

3.12.1 clCorOIRegister

clCorOIRegister

Synopsis:
A component can register itself as an OI through this API.

Header File:
clCorApi.h

Syntax:

```
          ClRcT clCorOIRegister(
     CL_IN const ClCorMOIdPtrT pMoId,
CL_IN const ClCorAddrPtrT  pCompAddr);
```

Parameters:
pMoId: (in) This is pointer to MOID of the MO.

pCompAddr: (in) IOC address of OI.

Return values:

Return values:
CL_OK: The API executed successfully.

CL_ERR_NO_MEM: Event library is not initialized.

CL_ERR_TIMED_OUT: The server failed to send the response within the time limit.

CL_ERR_NULL_PTR: `pMoid` or `pCompAddr` is NULL.

Description:
A component (specified by `compAddr`) can register itself as an OI for an MO (pointed by `*pMoId`) through this API. The MOID can be a qualified MOID or a wild card MOID. This API is synchronous in nature.

Library File:
ClCorClient

Related Function(s):
clCorOIUnregister, clCorPrimaryOIGet, clCorPrimaryOISet.

3.12.2 clCorOIUnregister

clCorOIUnregister

Synopsis:
De-register the component acting as the OI.

Header File:
clCorApi.h

Syntax:

```
  ClRcT clCorOIUnRegister (
CL_IN ClCorMOIdPtrT pMoid,
CL_IN ClCorAddrPtrT pCompAddr);
```

Parameters:

pMoId: (in) This is pointer to MOID of the MO.

pCompAddr: (in) The component that needs to be de-registered as the OI. The MOID can be a qualified MOID or a wild card MOID.

Return values:

CL_OK: If the OI un-registration operation is successful.

CL_ERR_OI_NOT_REGISTERED: Component was not registered as OI for this MO.

CL_ERR_NULL_PTR: `pCompAddr` or `pMoId` is a NULL pointer.

CL_ERR_TIMED_OUT: The server failed to send the response within the time limit.

Description:
If this API is executed successfully, the component pointed by `pCompAddr` discontinues to be the OI. If this component is the primary OI, it discontinues to act as the primary OI.

Library File:
ClCorClient

Related Function(s):
clCorOIRegister.

3.12.3 ClCorPrimaryOISet

clCorPrimaryOISet

Synopsis:
Sets a component as the primary OI.

Header File:
clCorApi.h

Syntax:

```
        ClRcT clCorPrimaryOISet(
CL_IN const ClCorMOIdPtrT pMoid,
     CL_IN const ClCorAddrPtrT pCompAddr);
```

Parameters:

pMoId: (in) Pointer to MOID of the MO. It can be a qualified MOID or wild card MOID.

pCompAddr: (in) This is the pointer to the component address structure.

Return values:

CL_OK: If the call is successful, the OI provided by the component address becomes the Primary OI for this MO.

CL_ERR_OI_NOT_REGISTERED: An OI can be a primary OI only if is registered as an OI. This error indicates that the OI is attempting to be a primary OI without registering as an OI.

CL_ERR_OI_ALREADY_SET: A Primary OI for the MO pointed by `pMoId` exists.

CL_ERR_NULL_PTR: `pMoid`, `pCompAddr`, or `pMoId` is a NULL pointer.

CL_ERR_TIMED_OUT: Server failed to send the response within the time limit.

Description:
This API sets the OI specified by `pCompAddr` as a primaryOI. A component can become a primary OI only if it has registered itself as OI using `clCorOIRegister()` API. This is a synchronous API.

Library File:
ClCorClient

Related Function(s):
clCorOIRegister, clCorPrimaryOIGet, clCorPrimaryOIUnset.

3.12.4 ClCorPrimaryOIGet

clCorPrimaryOIGet

Synopsis:
Obtains the primary OI for a given MO.

Header File:
clCorApi.h

Syntax:

```
 ClRcT clCorPrimaryOIGet(
CL_IN const ClCorMOIdPtrT pMoid,
CL_OUT ClCorAddrPtrT pCompAddr);
```

Parameters:

pMoId: (in) Pointer to MOID of the MO.

pCompAddr: (out) Pointer to the component address structure.

Return values:

CL_OK: The call is successful and the address of the component is populated in the second parameter

CL_ERR_OI_NOT_REGISTERED: There is no Primary OI registered for this MO.

CL_ERR_NULL_PTR: `pCompAddr` or `pMoId` is NULL.

CL_ERR_TIMED_OUT: Server failed to send the response within the time limit.

Description:
If this API is executed successfully, it obtains the primary OI for the MO pointed by `pMoid`. The component address of the primary OI is returned in `pCompAddr`.

Library File:
ClCorClient

Related Function(s):
clCorOIRegister, clCorPrimaryOIGet, clCorPrimaryOIUnset.

3.12.5 clCorPrimaryOIUnset

clCorPrimaryOIUnset

Synopsis:
De-register the component that is acting as the primary OI.

Header File:
clCorApi.h

Syntax:

```
        ClRcT clCorPrimaryOIUnset(
CL_IN const ClCorMOIdPtrT pMoid,
          CL_IN const ClCorAddrPtrT pCompAddr);
```

Parameters:

pMoId: (in) Pointer to MOID of the MO.

pCompAddr: (in) Pointer to the component address structure.

Return values:

CL_OK: The call is successful and the component's address is populated in the second parameter.

CL_ERR_OI_NOT_REGISTERED: This OI is not the Primary OI.

CL_ERR_NULL_PTR: `pCompAddr` or `pMoId` is NULL.

CL_ERR_TIMED_OUT: Server failed to send the response within the time limit.

Description:
This API cancels the registration of the component as the primary OI. If this API is successfully executed, the OI pointed by `pCompAddr` discontinues to act as the primary OI for the MO (pointed by the `pMoId`). This is a synchronous call.

Library File:
ClCorClient

Related Function(s):
clCorOIRegister, clCorPrimaryOIGet.

Chapter 4

Service Management Information Model

TBD

Chapter 5

Service Notifications

TBD

Chapter 6

Bundle Specific CLIs

This chapter provides descriptions for each debug CLI command relevant to COR.

6.1 clCorBundleInitialize

Parameters:

1: For transactional bundle.

2: For non-transactional bundle.

Description:

This command creates a bundle and returns a bundle handle. It takes a parameter to create a transactional or a non-transactional bundle. Transactional bundle is not supported in this release. This handle is unique for a given bundle. It should be used to add jobs into the bundle.

Output:

- The bundle ID is displayed on successful bundle initialization.
- An error code is displayed if a failure in bundle initialization occurs. The error codes are based on the error returned by `clCorBundleInitialize()` API.

6.2 clCorBundleGetJobAdd

Parameters:

bundleHandle: Handle of the COR bundle obtained from the `clCorBundleInitialize` CLI.

MoID: MOID for the object. For example,
Chassis:0
SysController:0 or
0x10001:0
0x10002:0

SvcID: Service ID of the MSO. `SvcId` must have the value `3` for PROV MSO and `2` for alarm MSO

attrPath: Attribute path must be set to NULL for PROV MSOs.

AttrID: Attribute ID of the attribute for which get is performed.

Index: In case of array attributes this specifies the index of the array from where the read operation must occur.

Description:

This command adds `MoID+attribute` to a queue. `clCorBundleApply()` should to be called after adding all the jobs into the bundle,

Output:

- No message is printed on successful addition of jobs.
- An error code is displayed, if a failure occurs. The error codes are based on the error returned by `clCorBundleJobAdd` API.

6.3 clCorBundleApply

Parameters:

bundleHandle: Handle to the bundle obtained from `clCorBundleInitialize` CLI.

Description:

This command applies the bundle containing jobs to the server for processing. This command operates synchronously. When the command is returned, an appropriate message is printed on the console. If an error occurs, the message is printed along with the error message.

Output:

- The data obtained is displayed on the console in the form of attribute ID and value. If it is an array attribute, the values for all its indexes are displayed. For other operations, only the job (along with error code,) which caused failure of the bundle, is displayed.
- If any error occurs while processing a job, the `MoID` and `attributeID` of that job with the error codes is displayed. This error code contains the value of `jobStatus` parameter of `clCorBundleJob`.

6.4 clCorBundleFinalize

Parameters:

bundleHandle: Handle to the bundle obtained from `clCorBundleInitialize` CLI.

Description:

This command finalizes a bundle and the handle becomes invalid.

Output:

- Error is not displayed, if the command is executed successfully.
- An error code is displayed, if a failure occurs. The error codes are based on the error returned by `clCorBundleFinalize()` API

6.5 objectShow

Parameters:

MoID: MoID of the object. Ex:

Chassis:0
SysController:0 or
0x10001:0
0x10002:0

SvcID: Service ID of the MSO or -1 if it is a MO.

attrPath: Attribute path of the MO , if it is a contained MO. Ex:
attr1:0
attr2:2 or
0x1001:0
0x1002:1 otherwise it is NULL.

Description:
The command provides the values of run-time and configuration attributes.

Output:
- The values of all the attributes are displayed when the command is successfully executed.
- - An error code is displayed if a failure occurs. The error codes are based on the `jobStatus` parameter returned by `clCorBundleJobAdd()` API.

6.6 rmShow

Parameters:
None.

Description:
This command displays the MOs and its registered OIs (IOC address of the component). There are specific flags that are displayed with IOC address. They are as follows:
STATUS - This flag provides the current status of the OI. Following are the values this flag can take and its significance:
- *1* - OI is enabled and ready to serve the request.
- *2* - OI is disabled. The OI is running but temporarily not serving the request.
- *3* - OI is deleted. The OI has failed and cannot serve the request.

STN - This flag provides the IOC address of the component. *PRIMARY* - This status indicates if this OI is acting as the primary OI for the MO.
- *0* - The OI is not a primary OI.
- *1* - The OI is a primary OI.

6.7 dmShow

Parameters:
ClassID: Class Identifier.

Description:
This command provides the details of the class and its attributes. If no argument is specified it displays all classes contained in COR.
Following attributes of the a `ClassID` are displayed: attribute ID, offset, size, type and user flags.
The user flags displayed for each attribute contains the following combination of characters that provide information about its type:

- CF - Configuration attribute
- RT - Runtime attribute
- OP - Operational attribute
- R-O - Read only attribute
- R-W - Read Write attribute
- C$ - Cached attribute
- N-C$ - Non cached attribute
- PERS - Persistent attribute
- N-PERS - Non-persistent attribute
- WRONCE - Writable
- INITED - Initialized

Glossary

Glossary of COR Service Terms:

Runtime Attribute Runtime attributes are also called transient attributes and are read-only. They are owned by the Object Implementors. To read a runtime attribute the OI needs to be contacted. The run-time objects and attributes form the descriptive part of the Information Model.

Object Implementer An ASP service or component that implements an object.

Job A job is an entity that specifies a CREATE, DELETE, SET, or GET operation on a MO. A job that specifies MO creation is called Create-Job. A job that specifies MO deletion is called Delete-Job. A job that represents a SET operation on the MO is called Set-Job and a job that represents a GET operation is called a Get-Job. A Get-Job has an MO and a list of attributes that need to be read.

Bundle A bundle is a group of jobs. A bundle enables efficient communication between the COR client, COR server, and OI by limiting the number of RMD calls between them. A bundle can either be operated as a transactional or a non transactional bundle. COR service ensures that all jobs in a transactional bundle are either successful or unsuccessful (failures). A non transactional bundle does not contain such restrictions and can contain unsuccessfully executed jobs due to error conditions.

Index