OpenClovis
Software Development Kit (SDK)
Service Description and API Reference for Name Service

For OpenClovis SDK Release 2.3 V0.4

Document Revision Date: March 27, 2007

Contents

Chapter 1

Functional Overview

The OpenClovis Name service provides a mechanism that allows an object or a service to be referred by its name instead of the Object Reference (Object Reference). This friendly name is agnostic to the topology and location.

The Object Reference can be the logical address, resource ID, and so on. Object Reference is opaque to Name Service. Name service returns the logical address when the friendly name is provided to it. Hence, Name Service provides location transparency. Name Service maintains a mapping table between objects and the Object Reference (logical address) associated with the object.

An object consists of an object name, an object type, and object attributes. The object names are strings and are meaningful when considered along with the object type. Examples of object names are print service, file service, functions, and so on. Examples of object types include services, nodes, or any other user-defined type. An object can have a number of attributes (limited by a configurable maximum number.) An object attribute consists of a `<attribute type, attribute value>` pair of strings. For example, `<version, 2.5>`, `<status, active>`, and so on.

The Name Service provides functions to register/de-register object names, types, attributes, and associated addresses. A process can register multiple services with a single address or multiple processes can register with a single service name and attributes. Some object names, attributes, and addresses can also be statically defined. Each registration is associated with the priority of the process providing the service. It is the responsibility of the process that registers an object, to remove the mapping when the object becomes invalid. However, a process can terminate abnormally. Name Service subscribes to notifications that can be received when a process dies from the Component Manager (CPM) and deletes the entries of the process in the database associated with the component.

The Name Service provides the ability to query the name database in various ways. Clients can provide the object name and object type to the Name Service and query it to retrieve the address and attributes of that object. Alternatively, clients can also provide attributes and query for object names that satisfy those attributes. Wild cards can also be used in the query. If two components provide the same service and have the same priority, the entry that is read first from the database is returned. Different components can provide different variants of a single service name with different attributes. For example, `comp1` provides car wash (manual) and `comp2` provides car wash (automatic). Manual and automatic are attributes of `comp1` and `comp2`, respectively. If the query is specified as `name = carwash` and `attribute = manual`, `comp1` is returned. If the attribute, manual, is not specified, the Object Reference of the component with the highest priority is returned.

Name Service supports different sets of name to Object Reference mappings (also called as Context of the mapping table). They are:

- User-defined set: 150 Applications can choose to be a part of a specific set (or Context). This requires for the Context to be created. For example, the OpenClovis ASP related services can be part of a separate Context.
- Default set: 150 Name Service supports a default Context for services that are not part of a specific Context. Name is unique in a Context.

The Name Service to be registered can have two scopes:

- Nodelocal scope: The scope is local to the node. The service provider and the user should co-exist on the same blade.
- Global or cluster-wide scope: The service is available to the user applications running on any of the blades. Any component residing anywhere in the cluster can access it. When a component registers with Name Service, it has to provide the Context and the scope of its service.

Chapter 2

Service Model

2.1 Usage Model

Name Service is based on the database model. Service providers can register their service name to Object Reference mapping with Name Service. This registration can be performed for all the services provided by the service provider. The service users query the Name Service for the Object Reference of the service providers. When a service is unavailable, the service provider de-registers the entry with Name Service. When a service provider dies ungracefully, all the services provided by it are de-registered.

A service can be registered with global scope or local scope. If it is registered with global scope, the service is available to any service user within the cluster. If the scope is local, the access to the services is limited to the Node. To support global and local scope, Name Service provides two types of Contexts:

- Global Context
- Local Context

The service provider decides the Context in which service entry needs to be registered. The service users must know the Context in which the service entry needs to be queried.

Name Service is used when the address of the service provider is dynamic. For example, if the service A is running on only one Node at a known port, the service users can use the physical address to contact service A, and Name Service is not required.

Name Service is used with Transparency Layer(TL) service provided by Intelligence Object Communication (IOC). Name Service contains Name to Object Reference Mapping and TL contains Object Reference to physical address mapping.

Name Service not only stores the name to logical address mapping but also stores any mapping between name and other Object References. For example, semname to semid, name to message queue ID, and so on.

2.2 Functional Description

The purpose of the Name Service is to provide location transparency. The service providers can register the mapping between services (exported by them) and Object Reference. To use the

services of Name Service, the user application has to initialize the Name Service by invoking the `clNameLibInitialize()` function. When this service is not required, the association with Name Client can be closed by invoking the `clNameLibFinalize()` function. Service users can create their own Context or use the default Context. Service users can create their own Context using the `clNameContextCreate()` function. `clNameContextCreate()` returns a `ContextId` that can be used in `clNameRegister()` function to identify the Context. Users can register their services with their unique name using the `clNameRegister()` function. A service can be registered by multiple components. As part of `clNameRegister()`, the service provider can request Name Service to generate an Object Reference or they can set the Object Reference to a particular name. When the service provider fails, it can de-register the service using the `clNameComponentDeregister()` function.

Multiple components can register with a single name. The last component can de-register its service, using the `clNameServiceDeregister()` function. A service provider can delete a Context using the `clNameContextDelete()` function. The default Context cannot be deleted. Name Service deletes the default Context when the service shuts down gracefully.

Every Object Name is characterized by its name, attribute count, and a set of attributes. These attributes are specified when the service is registered through the `ClNameSvcAttrEntryT` structure.

The service users can query the Name Service for Object Reference using the `clNameToObjectReferenceGet()` function. The service users have to provide the following:

- Name
- Name and Attributes
- Attributes

The service users can also query the entire mapping of service Name to Object Reference using the `clNameToObjectMappingGet()` function. This function returns all the matching records. `clNameToObjectMappingGet()` returns the following:

- Name
- Object Reference
- Attributes (if specified)

The mapping returned can be freed using the `clNameObjectMappingCleanup()` function.

Chapter 3

Service Functions

3.1 Type Definitions

3.1.1 ClNameSvcRegisterT

typedef struct {
ClNameT name;
ClUint32T compId;
ClNameSvcPriorityT priority;
ClUint32T attrCount;
ClNameSvcAttrEntryT attr[1];
} ClNameSvcRegisterT;

This `ServiceRegisterInfo` structure is provided by a process to the Name Service to register a name. `ClNameSvcRegisterT` contains the name service registration information. The attributes of this structure are:

- *name* - Name of the service entry to be registered.
- *compId* - ID of the component invoking this function.
- *priority* - Priority of the service given by this component. The priority can be set to `CL_NS_PRIORITY_LOW`, `CL_NS_PRIORITY_MEDIUM`, `CL_NS_PRIORITY_HIGH`.
- *attrCount* - Number of attributes associated with this entry by the component.
- *attr* - List of attributes of this service.

3.1.2 ClNameSvcContextT

typedef enum{
CL_NS_USER_NODELOCAL,
CL_NS_USER_GLOBAL
} ClNameSvcContextT;

This enumeration is used to specify the scope of the Context. The Context can be local to the node (Local NameContext) or global to the cluster (Global NameContext).

3.1.3 ClNameSvcAttrEntryT

typedef struct{
ClUint8T type [CL_Name service_MAX_STR_LENGTH];
ClUint8T value [CL_Name service_MAX_STR_LENGTH];
} ClNameSvcAttrEntryT;

The structure, `ClNameSvcAttrEntryT`, is provided by the process when its name is being registered. The attributes of this structure are:

- *type* - Type of the attribute.
- *value* - Value of the attribute.

3.1.4 ClNameSvcEntryPtrT

typedef struct{
ClNameT name;
ClUint64T objReference;
ClUint32T refCount;
ClNameSvcCompListT compId;
ClUint32T attrCount;
ClNameSvcAttrEntryT attr[1];
} ClNameSvcEntryT;

typedef ClNameSvcEntryT ClNameSvcEntryPtrT;*

This structure, `ClNameSvcEntryT`, contains the Name Service entry. It contains attributes associated with a Name entry. They are:

- *name* - Name of the service entry to be registered.
- *objReference* - Unique reference to this object.
- *refCount* - Number of components providing this service.
- *compId* - List of components providing this service and their information such as `compId`, `priority`, and so on.
- *attrCount* - Number of attributes associated with the entry.
- *attr[1]* - List of attributes of this service.

3.1.5 ClNameSvcConfigT

typedef struct{
ClUint32T nsMaxNoEntries;
ClUint32T nsMaxNoGlobalContexts;
ClUint32T nsMaxNoLocalContexts;
} ClNameSvcConfigT;

The structure, `ClNameSvcConfigT`, contains the Name Service configuration information.

3.1.6 ClNameSvcCompListT

```
typedef struct{
        ClUint32T dsId;
        ClUint32T compId;
        ClNameSvcPriority priority;
        struct ClNameSvcCompListT *pNext;
} ClNameSvcCompListT;
```

The structure, `ClNameSvcCompListT`, provides information about the component that is registering its name with Name Service. The attributes of this structure are:

- *dsId* - `dataSetId` in which the information must be checkpointed. This value will be assigned dynamically by Name Service.
- *compId* - ID of the component providing this service.
- *priority* - Priority of the service.
- *pNext* - Pointer to the next component providing the same service. This should always be set to NULL.

3.1.7 ClNameSvcOpT

```
typedef enum{
        CL_NS_COMPONENT_DEREGISTER,
        CL_NS_SERVICE_DEREGISTER
} ClNameSvcOpT;
```

The enumeration, `ClNameSvcOpT`, is used to specify the operation code. This operation code can be set to component de-register or service de-register.

3.1.8 ClNameSvcEventInfoT

```
typedef struct{
        ClNameT name;
        ClUint64T objReference;
        ClNameSvcOpT operation;
        ClUint32T ContextMapCookie;
} ClNameSvcEventInfoT;
```

The structure, `ClNameSvcEventInfoT`, contains the payload (event data) of an event. This event is published when the service de-registers or a component of the service de-registers. The attributes of this structure are:

- *name* - Name of the service to be de-registered.
- *objReference* - Object Reference of this name.
- *operation* - Operation code that specifies if it is a component de-register or service de-register operation.
- *contexMapCookie* - Context in which this service exists.

3.1.9 ClNameSvcAttrSearchT

```
typedef struct{
        ClNameSvcAttrEntryT attrList[CL_Name service_MAX_NO_ATTR];
        ClUint32T attrCount;
        ClUint32T opCode[CL_Name service_MAX_NO_ATTR -1];
} ClNameSvcAttrSearchT;
```

When the Name Service searches for an entry in the database, it uses this structure to compare the attributes of the entry with the entries of the service provider. The attributes of this structure are:

- *attrList* - List of attributes of the service.
- *attrCount* - Number of attributes associated with the service .

3.2 Library Life Cycle APIs

3.2.1 clNameLibInitialize

clNameLibInitialize

Synopsis:
This function is used to initialize Name Service for the calling process.

Header File:
clNameApi.h

Syntax:

```
ClRcT clNameLibInitialize(void);
```

Parameters:
None.

Return values:
CL_OK: The function executed successfully.

CL_ERR_ALREADY_INITIALIZED: The library is already initialized.

Description:
This function is used to initialize Name Service for the calling process. This function creates an association with the Name Service and its client. This function must be invoked before any other function of Name Service function is used.

Library File:
ClNameClient

Related Function(s):
clNameLibFinalize

3.2.2 clNameLibFinalize

clNameLibFinalize

Synopsis:
This function is used to finalize the Name Service for the calling process.

Header File:
clNameApi.h

Syntax:

```
ClRcT clNameLibFinalize(void);
```

Parameters:
None.

Return values:
CL_OK: The function executed successfully.

CL_ERR_NOT_INITIALIZED: The library is not initialized.

Description:
This function is used to finalize the Name Service for the calling process. This function deletes the association with the Name Service and its client. After this function is executed successfully, any operation on Name Service is invalid.

Library File:
ClNameClient

Related Function(s):
clNameLibInitialize

3.2.3 clNameInitialize

clNameInitialize

Synopsis:
This function is used to initialize the Name Service component.

Header File:
clNameConfigApi.h

Syntax:

```
ClRcT clNameInitialize (void);
```

Parameters:
pConfig: Pointer to the Name Service configuration data as defined in `ClNameSvcConfigT`.

Return values:
CL_OK: The function executed successfully.

CL_ERR_NULL_POINTER: pConfig contains a NULL pointer.

CL_Name Service_ERR_DUPLICATE: An instance of Name Service is running on a blade where this component is running.

CL_ERR_NO_MEMObject ReferenceY: Memory allocation failure.

Description:
This function initializes the Name Service server component and allocates resources to it. This function must be invoked before Name Service can be used.

Library File:
libClNameClient

Related Function(s):
clNameFinalize.

3.2.4 clNameFinalize

clNameFinalize

Synopsis:
This function is used to free the resources acquired when the Name Service component is initialized.

Header File:
clNameConfigApi.h

Syntax:
```
ClRcT clNameFinalize(void);
```

Parameters:
None.

Return values:
CL_OK: The function executed successfully.

Description:
This function is used to free the resources acquired when the Name Service component is initialized. This must be called when the services of the Name Service component are not required.

Library File:
libClNameClient

Related Function(s):
clNameInitialize.

3.3 Functional APIs

3.3.1 clNameRegister

clNameRegister

Synopsis:
This function is used to register a name to Object Reference mapping.

Header File:
clNameApi.h

Syntax:

```
ClRcT clNameRegister(
                CL_IN ClUint32T  ContextId,
                CL_IN ClNameSvcRegisterT  * pName serviceRegisInfo,
                CL_INOUT ClUint64T  *pObjReference);
```

Parameters:

ContextId: (in) Context in which the service is to be provided. Before a service is registered in the user-defined Context, the Context must be created using `clNameContextCreate` function. It can have the following values:

- `CL_NS_DEFT_GLOBAL_CONTEXT`: For registering in global Context.
- `CL_NS_DEFT_LOCAL_CONTEXT`: For registering in node local Context.
- ID returned by `clNameContextCreate()` for user-defined Contexts.

pName serviceRegisInfo: (in) Pointer to registration information as defined in `ClNameSvcRegisterT`.

pObjReference: (in/out) This contains the Object Reference. If Object Reference is known, `pObjReference` contains the known value. If Object Reference is unknown, `pObjReference` must be set to `CL_NS_GET_OBJ_REF`. The allocated Object Reference is returned in `pObjReference`.

Return values:

CL_OK: The function executed successfully.

CL_ERR_NULL_POINTER: `pName`, `serviceRegisInfo`, or `pObjReference` contains a NULL pointer.

CL_Name Service_ERR_CONTEXT_NOT_CREATED: Cannot register, de-register, or query a Context that does not exist.

CL_Name Service_ERR_LIMIT_EXCEEDED: Cannot create or register Contexts and entries greater than the maximum allowed.

CL_ERR_NO_MEMObject ReferenceY: Memory allocation failure.

CL_ERR_NOT_INITIALIZED: Name Service library is not initialized.

Description:
A service provider can register a name to Object Reference mapping with Name Service using this function. It can register the name using the default `ContextId` or its own `ContextId`. To register using its own `ContextId`, the service provider must first create the Context. The service can be registered in global Context or local Context. The service is

accessible throughout the cluster if the service is registered in global Context. If the service is registered in local Context, the access is limited to the node only.
`pObjRef` is an in/out (input/output) variable. A component can provide its logical address and Object Reference, or it can request Name Service to generate an Object Reference by setting the `CL_NS_GET_OBJ_REF` flag.

Library File:
ClNameClient

Related Function(s):
clNameComponentDeregister, clNameserviceDeregister.

3.3.2 clNameComponentDeregister

clNameComponentDeregister

Synopsis:
This function is used to de-register a component with Name Service.

Header File:
clNameApi.h

Syntax:

```
   ClRcT clNameComponentDeregister(
CL_IN ClUint32T compId);
```

Parameters:
compId: (in) ID of the component.

Return values:
CL_OK: The function executed successfully.

CL_ERR_NOT_INITIALIZED: Name Service library is not initialized.

Description:
A service provider can de-register all its services, when it shuts down gracefully using this function. On successful completion of this function, all the services registered by this component are de-registered.

Library File:
ClNameClient

Related Function(s):
clNameRegister, clNameserviceDeregister.

3.3.3 clNameServiceDeregister

clNameserviceDeregister

Synopsis:
This function is used to de-register a service provided by a component.

Header File:
clNameApi.h

Syntax:

```
ClRcT clNameServiceDeregister(
                        CL_IN ClUint32T ContextId,
                        CL_IN ClUint32T compId,
                        CL_IN ClNameT* serviceName);
```

Parameters:

ContextId: (in) Context in which the service is to be provided. Before a service is registered in the user-defined Context, the Context must be created using `clNameContextCreate` function. It can have the following values:

- `CL_Name Service_DEFT_GLOBAL_CONTEXT`: For registering in global Context.
- `CL_Name Service_DEFT_LOCAL_CONTEXT`: For registering in node local Context.
- ID returned by `clNameContextCreate()` for user-defined Contexts.

compId: (in) ID of the component.

serviceName: (in) Pointer to the name of the service being de-registered.

Return values:

CL_OK: The function executed successfully.

CL_ERR_NULL_POINTER: serviceName contains a NULL pointer.

CL_Name Service_ERR_CONTEXT_NOT_CREATED: Cannot register, de-register, or query a Context that does not exist.

CL_Name Service_ERR_service_NOT_REGISTERED: Cannot de-register a service that is not registered.

CL_ERR_NOT_INITIALIZED: Name Service library is not initialized.

Description:
This function is used to de-register a service provided by a component in a specified Context. The de-registered service cannot be accessed and a query to this service generates an error. This service can be de-registered only if it is previously registered by a component. Name Service frees the resources acquired by this service registration.

Library File:
ClNameClient

Related Function(s):
clNameRegister, clNameServiceDeregister, clNameComponentDeregister.

3.3.4 clNameContextCreate

clNameContextCreate

Synopsis:
Creates a Context.

Header File:
clNameApi.h

Syntax:

```
   ClRcT clNameContextCreate(
CL_IN ClNameSvcContextT ContextType,
CL_IN ClUint32T ContextMapCookie,
CL_OUT ClUint32T *ContextId);
```

Parameters:

ContextType: (in) Indicates if the scope is global or node local.

ContextMapCookie: (in) This is used during look-up. Look-up refers to a search operation for a name. There is a one-to-one mapping between `ContextMapCookie` and `ContextId` that is returned by this function. This parameter can accept the following values:

- `CL_NS_DEFT_GLOBAL_MAP_COOKIE`: For querying the default global Context.
- `CL_NS_DEFT_LOCAL_MAP_COOKIE`: For querying the default local Context.

ContextId: (out) Pointer to the ID of the created Context.

Return values:

CL_OK: The function executed successfully.

CL_Name Service_ERR_LIMIT_EXCEEDED: Cannot create or register Contexts and entries more than the maximum allowed.

CL_ERR_NO_MEMObject ReferenceY: Memory allocation failure.

CL_ERR_INVALID_PARAMETER: An invalid parameter is passed to the function.

CL_ERR_NULL_POINTER: `ContextId` contains a NULL pointer.

CL_Name Service_ERR_CONTEXT_ALREADY_CREATED: Cannot create a Context with a `ContextMapCookie` that is already in use.

CL_ERR_NOT_INITIALIZED: Name Service library is not initialized.

Description:

This function creates a user-defined Context (both global scope and node local scope). The service providers have to maintain a separate name space for their set of services. They can create their own Context using this function and this creates an internal namespace for the Context. The scope of the Context to be created can be specified. If the scope is global, the services registered in this Context are reachable within the cluster.
When this function is successfully executed, the corresponding Context ID is returned to the calling process. The service-users/providers should use this Context ID for all other Context related operations. Every Context has a `ContextMapCookie`, that uniquely identifies the Context. While querying, the service user must specify the `ContextMapCookie` associated with the Context while performing a look-up.

Library File:
ClNameClient

Related Function(s):
clNameContextDelete.

3.3.5 clNameContextDelete

clNameContextDelete

Synopsis:
This function is used to delete a Context.

Header File:
clNameApi.h

Syntax:

```
   ClRcT clNameContextDelete(
CL_IN ClUint32T ContextId);
```

Parameters:
ContextId: (in) Context to be deleted.

Return values:
CL_OK: The function executed successfully.

CL_Name Service_ERR_CONTEXT_NOT_CREATED: Cannot register, de-register, or query a Context that does not exist.

CL_Name Service_ERR_OPERATION_NOT_PERMITTED: Cannot delete default Contexts.

CL_ERR_NOT_INITIALIZED: Name Service library is not initialized.

Description:
This function deletes the user-defined Context (both global scope and node local scope). This function frees the resources acquired when the Context was created and frees the Name Service entries registered in the Context. The contextId, acquired from `clNameContextCreate()`, becomes invalid. The registered names to Object Reference mapping in this ContextId are de-registered by Name Service. The default predefined Context cannot be deleted using this function.

Library File:
ClNameClient

Related Function(s):
clNameContextCreate.

3.3.6 clNameToObjectReferenceGet

clNameToObjectReferenceGet

Synopsis:
This function is used to return the Object Reference for a service.

Header File:
clNameApi.h

Syntax:

```
   ClRcT clNameToObjectReferenceGet(
CL_IN ClNameT* pName,
CL_IN ClUint32T attrCount,
CL_IN ClNameSvcAttrEntryT *pAttr,
CL_IN ClUint32T ContextMapCookie,
CL_OUT ClUint64T* pObjReference);
```

Parameters:

pName: (in) Pointer to the service name.

attrCount: (in) Number of attributes passed in the query. If the number of attributes is unknown, the value can be set to `CL_NS_DEFT_ATTR_LIST` and the Object Reference of the component with the highest priority is returned.

pAttr: (in) Pointer to the list of attributes. If `attrCount=CL_NS_DEFT_ATTR_LIST`, `pAttr` must be set to NULL.

ContextMapCookie: (in) Cookie to find the Context. There is one-to-one mapping between `ContextMapCookie` and Context. This parameter can accept the following values:

- `CL_NS_DEFT_GLOBAL_MAP_COOKIE`: For querying the default global Context.
- `CL_NS_DEFT_LOCAL_MAP_COOKIE`: For querying the default local Context.

pObjReference: (out) Pointer to the Object Reference associated with the service.

Return values:

CL_OK: The function executed successfully.

CL_Name Service_ERR_CONTEXT_NOT_CREATED: Cannot register, de-register, or query a Context that does not exist.

CL_Name Service_ERR_ENTRY_NOT_FOUND: Cannot query an entry that does not exist.

CL_ERR_NULL_POINTER: `pName`, `pAttr`, or `pObjReference` contains a NULL pointer.

CL_ERR_NOT_INITIALIZED: Name Service library is not initialized.

Description:

This function is used to query and retrieve the Object Reference for a given service name. The service users can access any service that is registered in Name Service. They can query the Object Reference to access a service irrespective of its location. The service user must know either the service name or service attributes, to retrieve the Object Reference of a name.

When this function is successfully executed, the Object Reference is returned. Using this Object Reference, the users can access a service. If the same service is registered by multiple components, and the service user needs to retrieve the service with the highest priority, attrCount must be set to `CL_NS_DEFT_ATTR_LIST`.

Library File:
ClNameClient

Related Function(s):
clNameToObjectMappingGet.

3.3.7 clNameToObjectMappingGet

clNameToObjectMappingGet

Synopsis:

This function is used to return the Name Service entry of the service.

Header File:

clNameApi.h

Syntax:

```
   ClRcT clNameToObjectMappingGet( CL_IN ClNameT* pName,
CL_IN ClUint32T attrCount,
CL_IN ClNameSvcAttrEntryT *pAttr,
CL_IN ClUint32T ContextMapCookie,
CL_OUT ClNameSvcEntryPtrT* pOutBuff);
```

Parameters:

pName: (in) Pointer to the service name.

attrCount: (in) Number of attributes passed in the query. If the number of attributes is unknown, the value can be set to `CL_Name Service_DEFT_ATTR_LIST` and the Object Reference of the component with the highest priority is returned.

pAttr: (in) Pointer to list of attributes. This must be set to NULL if `attrCount=CL_NS_DEFT_ATTR_LIST`.

ContextMapCookie: (in) Cookie to find the Context. There is one-to-one mapping between `ContextMapCookie` and Context. This parameter can accept the following values:

- `CL_NS_DEFT_GLOBAL_MAP_COOKIE`: For querying the default global Context.
- `CL_NS_DEFT_LOCAL_MAP_COOKIE`: For querying the default local Context.

pOutBuff: (out) Contains the entry. The calling process/component must free the memory after successful execution of this function.

Return values:

CL_OK: The function executed successfully.

CL_Name Service_ERR_CONTEXT_NOT_CREATED: Cannot register, de-register, or query a Context that does not exist.

CL_Name Service_ERR_ENTRY_NOT_FOUND: Cannot query an entry that does not exist.

CL_ERR_NULL_POINTER: `pName`, `pAttr`, `pOutBuff` contains a NULL pointer.

CL_ERR_NOT_INITIALIZED: Name Service library is not initialized.

Description:

This function provides the entire Name Service entry for the service Name provided. Details about a service can be retrieved using this function. This function returns the following information:

- Name of the service
- Corresponding Object Reference
- Associated attributes

If a single same service is registered by multiple service providers, Name Service returns all matched entries. The service users can query Name Service using the service name, attributes, or a combination of both. The information provided in `ClNameSvcRegisterT`

(during registration) can be retrieved. If multiple components have registered with the same name, the information about all such components can be retrieved. On successful completion of this function, `pOutBuf` is allocated and populated by Name Service. It is the responsibility of the calling component/process to free the mapping information. This can be performed using the `clNameObjectMappingCleanup()` function.

Library File:
ClNameClient

Related Function(s):
clNameObjectMappingCleanup, clNameToObjectReferenceGet.

3.3.8 clNameObjectMappingCleanup

clNameObjectMappingCleanup

Synopsis:
This function is used to free the object mapping entry.

Header File:
clNameApi.h

Syntax:

```
   ClRcT clNameObjectMappingCleanup(
CL_IN ClNameSvcEntryPtrT pObjMapping);
```

Parameters:
pObjMapping: (in) Pointer to the object mapping being deleted.

Return values:
CL_OK: The function executed successfully.

CL_ERR_NULL_POINTER: `pObjMapping` contains a NULL pointer.

Description:
This function is used to free the memory the object mapping returned during the look-up or search. It frees the memory allocated for `clNameToObjectMappingGet()`. To avoid memory leaks, every `clNameToObjectMappingGet()` must be followed by a `clNameObjectMappingCleanup()`.

Library File:
ClNameClient

Related Function(s):
clNameToObjectMappingGet.

3.3.9 clNameLibVersionVerify

clNameLibVersionVerify

Synopsis:
This function is used to verify the version with the Name Service library.

Header File:
clNameApi.h

Syntax:

```
ClRcT clNameLibVersionVerify(
                        CL_INOUT ClVersionT *pVersion)
```

Parameters:

pVersion: (in/out) iPointer to the version of Name Service Library required by the calling component. If the required version does not match the version of the Name Service library, the function returns the version supported by the Name Service library.

Return values:

CL_OK: The function executed successfully.

CL_ERR_NULL_POINTER: `pVersion` contains a NULL pointer.

CL_ERR_VERSION_MISMATCH: Version passed to the client is incorrect.

Description:

This function is used to verify the required version of the calling component with the implementation version of the Name Service. If the implementation supports the required releaseCode (a member of `pVersion`) and its majorVersion is greater than or equal to the required `majorVersion`, the functions returns `CL_OK` and pVersion is set to:

- `releaseCode` = required `releaseCode`.
- `majorVersion` = highest major version supported for required `releaseCode`.
- `minorVersion` = highest minor version supported for the returned `releaseCode` and `majorVersion`.

If this condition is not met, the function returns `CL_ERR_VERSION_MISMATCH` and `pVersion` is set to:

- releaseCode:
  - required `releaseCode`, if it is supported.
  - Lowest `releaseCode` higher than required releaseCode, if the required `releaseCode` is lower than any supported `releaseCode`.
  - Highest `releaseCode` lower than required releaseCode, if the required `releaseCode` is higher than any supported `releaseCode`.
- `majorVersion` = highest major version supported for returned `releaseCode`.
- `minorVersion` = highest minor version supported for returned `releaseCode` and `majorVersion`.

Library File:
ClNameClient

Related Function(s):
None.

Chapter 4

Service Management Information Model

TBD

Chapter 5

Service Notifications

TBD

Chapter 6

Debug CLIs

TBD

Glossary

Glossary of Name Service Terms:

Name Service(NS) ASP Service that stores the Name to Object Reference mapping and provides the facility to query the same.

Service provider

Service user

Object Object can be any service or resource.

Object Name A string that is one of the unique identifiers for an object.

Object Reference A reference to a particular object. It can be the logical address, resource ID, and so on.

Context A set of Name Service entries.

Local Context A set of mappings of node local services.

Global Context A set of mappings of global services.

Name Service Client Part of the Name Service linked to the user application. This provides Name Service interface to the users.

NS Server Server part of Name Service that performs the actual processing. The Name Service elements with Name clients form the Name Service.

Index