# OpenClovis Software Development Kit (SDK) Service Description and API Reference for Component Management (CPM) Service

For OpenClovis SDK Release 2.3 V0.4

Document Revision Date: March 27, 2007

# Contents

Chapter 1

# Functional Overview

The Availability Management Framework (AMF) functions are collectively implemented in the Availability Management Service (AMS) and the Component Manager (CPM) software components. AMS is the primary decision making entity in the system with respect to service availability. AMS and CPM together maintain a view of the system model that describes the various software and hardware components in the system, their attributes, dependencies on each other, rules for grouping them together for providing services availability, current state and policies to be applied in the event of failure. The CPM is the operational manager entity and is responsible for managing the various Service Units/components through their life cycle, based on the configured policy and directions from AMS. Component management is achieved in a hierarchical manner and is implemented in two ways:

1. CPM/L (available on All nodes)
2. CPM/G (available on the node where AMS exists) CPM/L takes on the additional responsibility of managing the overall system].

Component Manager provides following functionalities:

- Life cycle management of the components.
- Boot Management functionality.
- Health check for the components available in the system. Component failure, detected by Heartbeat mechanism, is notified by CPM.
- Instantiate/Terminate/Cleanup/Restart of the component.

CPM also exposes all the SAF defined interfaces, which can be used by various applications/components.

- Library Life Cycle
  - clCpmClientInitialize
  - clCpmClientFinalize
  - clCpmSelectionObjectGet
  - clCpmDispatch
- Component Registration and Unregistration

  - clCpmComponentRegister
  - clCpmComponentUnregister
  - clCpmComponentNameGet
- Passive Monitoring of processes of components
  - Not Supported
- Component Health Monitoring
  - Not Supported
- Component Service Instance Management [in HA only]
  - clCpmHAStateGet
  - clCpmCSIQuiescingComplete
  - ClCpmCSISetCallbackT
  - ClCpmCSIRmvCallbackT
- Component Life Cycle
  - ClCpmTerminateCallbackT
  - ClCpmProxiedComponentInstantiateCallbackT
  - ClCpmProxiedComponentCleanupCallbackT
- Protection Group Management [in HA only]
  - clCpmProtectionGroupTrack
  - clCpmProtectionGroupTrackStop
  - ClCpmProtectionGroupTrackCallbackT
- Error Reporting [in HA case only]
  - clCpmComponentFailureReport
  - clCpmComponentFailureClear [Not supported]
- Component Response to framework requests
  - clCpmResponse

Along with this CPM also exposes the following APIs:

- Boot Management
  - clCpmBootLevelGet
  - clCpmBootLevelSet
  - clCpmBootLevelMax
- Component Management
  - clCpmComponentIdGet
  - clCpmComponentAddressGet
  - clCpmComponentStatusGet
  - clCpmComponentInstantiate

    - clCpmComponentTerminate
    - clCpmComponentCleanup clCpmComponentRestart

- EO Management
    - clCpmFuncEOWalk
    - clCpmExecutionObjectStateSet

- Other APIs
    - clCpmMasterAddressGet
    - clCpmIsMaster
    - clCpmNodeShutDown
    - clCpmLocalNodeNameGet

CPM also interacts with other OpenClovis ASP Infrastructure components, mainly event services for publishing node arrival or departure events and component failure events.

# Chapter 2

# Service Model

TBD

# Chapter 3

# Service APIs

## 3.1 Defines

**#define CL_CPM_DEFAULT_MAX_FREQ 32000**

Maximum frequency at which Component Manager checks the heartbeats of the component. This value is in milliseconds. It can be overwritten in the Component Manager configuration file.

**#define CL_CPM_DEFAULT_MIN_FREQ 2000**

Default frequency at which Component Manager checks the heartbeats of the component. This value is in milliseconds. It can be overwritten in the Component Manager configuration file.

**#define CL_CPM_EO_ALIVE (ClUint32T)0**

This define is related to EO state management, in the context of heartbeat success or failure.

**#define CL_CPM_EO_DEAD (ClUint32T)0xFFFFFFF**

This define is related to EO state management, in the context of heartbeat success or failure.

**#define CL_CPM_EO_VERSION_NO 0x0100**

Component Manager-supported version of EO library.

**#define CL_CPM_ERR_BAD_OPERATION (CL_ERR_COMMON_MAX + 8)**

Component Manager returns this error when it receives an invalid request.

**#define CL_CPM_ERR_EO_UNREACHABLE (CL_ERR_COMMON_MAX + 2)**

EO is not reachable.

**#define CL_CPM_ERR_EXIST (CL_ERR_COMMON_MAX + 9)**

Component Manager returns this error when the same registration request is requested multiple times.

**#define CL_CPM_ERR_FORWARDING_FAILED (CL_ERR_COMMON_MAX + 5)**

Component Manager is unable to forward the request to the required node.

**#define CL_CPM_ERR_INIT (CL_ERR_COMMON_MAX + 7)**

Component Manager returns this error if the initialization was not performed correctly and other functions are being accessed.

**#define CL_CPM_ERR_INVALID_ARGUMENTS (CL_ERR_COMMON_MAX + 3)**

One of the parameters is not set correctly.

**#define CL_CPM_ERR_OPER_ABANDONED (CL_ERR_COMMON_MAX + 10)**

Component Manager returns this error when processing of the request is abandoned due to a high priority request.

**#define CL_CPM_ERR_OPERATION_FAILED (CL_ERR_COMMON_MAX + 4)**

Requested operation could not be performed by Component Manager.

**#define CL_CPM_ERR_OPERATION_IN_PROGRESS (CL_ERR_COMMON_MAX + 6)**

Component Manager cannot handle this request, as it is currently performing similar operations.

**#define CL_CPM_ERR_OPERATION_NOT_ALLOWED (CL_ERR_COMMON_MAX + 1)**

Requested operation is not allowed.

**#define CL_CPM_EVENT_CHANNEL_NAME "CPM_EVENT_CHANNEL"**

Event Channel information related to Component failure. Component Manager publishes an event for the component failures using event filters. The event can be subscribed to using the IOC address.

**#define CL_CPM_IOC_ADDRESS_CHASSIS_GET(address, myCh) myCh = (address >> CL_CPM_IOC_SLOT_BITS); \**

Macro for getting the `chsssisId` from IOC address.

**#define CL_CPM_IOC_ADDRESS_GET(myCh, mySl, address)**

**Value:**

```
{\
    address = (myCh << CL_CPM_IOC_SLOT_BITS); \
    address |= mySl; \
    }
```

Macro for generating the IOC address based on the `chassidId` and `slotId`.

**#define CL_CPM_IOC_ADDRESS_SLOT_GET (address, mySl) mySl = address & CL_-CPM_IOC_SLOT_BITS_HEX; \**

Macro for getting the `slotId` from IOC address.

**#define CL_CPM_IOC_SLOT_BITS 16**

**#define CL_CPM_IOC_SLOT_BITS_HEX 0x0000ffff**

**#define CL_CPM_MAJOR_VERSION 0x01**

**#define CL_CPM_MINOR_VERSION 0x01**

**#define CL_CPM_NODE_EVENT_CHANNEL_NAME "CPM_NODE_EVENT_CHANNEL"**

The node arrival and departure related Event channel information.

**#define CL_CPM_RC(ERROR_CODE) CL_RC(CL_CID_CPM, (ERROR_CODE))**

**#define CL_CPM_RELEASE_CODE 'B'**

Component Manager-supported version for SA Forum.

**#define ClCpmSchedFeedBackT ClEoSchedFeedBackT**

Feedback sent by the software component that is polled in response to heartbeat (is-Alive).

## 3.2 Type Definitions

### 3.2.1 ClCpmCompProcessRelT

*typedef enum clCpmCompProcessRel ClCpmCompProcessRelT;*

The enumeration, `ClCpmCompProcessRelT`, indicates the relationship between the component and the process.

### 3.2.2 ClCpmCSIRmvCallbackT

*typedef ClRcT ( *ClCpmCSIRmvCallbackT ) (*
*CL_IN ClInvocationT invocation,*
*CL_IN const ClNameT *pCompName,*
*CL_IN const ClNameT *pCsiName,*
*CL_IN ClAmsCSIFlagsT csiFlags );*

Removes one or all the assigned CSIs. Component Manager can request the component, `compName`, to remove a particular CSI or all the CSIs using this function. This is an asynchronous function.

- *invocation* - Invocation of this callback function.
- *pCompName* - Pointer to the name of the component to which a CSI needs to be assigned.
- *pCsiName* - Pointer to the `csiName` that is to be removed from the component, `compName`.
- *csiFlags* - Flags that indicate if one or more CSI is affected.

### 3.2.3 ClCpmCSISetCallbackT

*typedef ClRcT (*ClCpmCSISetCallbackT) (*
*CL_IN ClInvocationT invocation,*
*CL_IN const ClNameT *pCompName,*
*CL_IN ClAmsHAStateT haState,*
*CL_IN ClAmsCSIDescriptorT csiDescriptor );*

Component Manager requests the component, `compName`, to assume a particular HA state, specified by `haState`, for one or all the CSIs. This is an asynchronous function.

- *invocation* - Invocation of this callback function.
- *pCompName* - Pointer to the name of the component to which a CSI needs to be assigned.
- *haState* - HA state to be assigned for the CSI, `csiDescriptor`, or for all CSIs, supported by the component, (if `CSI_TARGET_ALL` is set in `csiFlags` of `csiDescriptor`.)
- *csiDescriptor* - Information about the CSI.

### 3.2.4 ClCpmHandleT

typedef ClHandleT ClCpmHandleT;

### 3.2.5 ClCpmProtectionGroupTrackCallbackT

*typedef void(ClCpmProtectionGroupTrackCallbackT)(*
*CL_IN const ClNameT pCsiName,*
*CL_IN ClAmsPGNotificationBufferT pNotificationBuffer,*

*CL_IN ClUint32T numberOfMembers,*
*CL_IN ClUint32T error);*

Requested information is delivered to `notificationBuffer` using this callback function. The type of information returned in this buffer depends on the `trackFlag` parameter of the `clAMSProtectionGroupTrack` function. This is an asynchronous function.

*pCsiName:* Pointer to `csiName` that is to be removed from the component, `compName`.

*pNotificationBuffer:* Pointer to a notification buffer, that contains the requested information.

*numberOfMembers:* Number of components that belong to protection group associated with the CSI, `csiName`.

*error:* Indicates if Component Manager performed the operation.

### 3.2.6 ClCpmProxiedComponentCleanupCallbackT

*typedef ClRcT( ClCpmProxiedComponentCleanupCallbackT)(*
*CL_IN ClInvocationT invocation,*
*CL_IN const ClNameT pProxiedCompName);*

Component Manager requests the proxy component to free the component, `proxiedCompName`, using this callback function. This is an asynchronous function.

*invocation:* Invocation of this callback function.

*pProxiedCompName:* Pointer to the name of the component that needs to be freed.

### 3.2.7 ClCpmProxiedComponentInstantiateCallbackT

*typedef ClRcT(ClCpmProxiedComponentInstantiateCallbackT)(*
*CL_IN ClInvocationT invocation,*
*CL_IN const ClNameT pProxiedCompName)*

Component Manager requests the proxy component to instantiate the component, `proxiedCompName`, using this callback function. This is an asynchronous function.

*invocation:* Invocation of this callback function.

*pProxiedCompName:* Pointer to the name of the component that needs to be instantiated.

### 3.2.8 ClCpmTerminateCallbackT

*typedef ClRcT( ClCpmTerminateCallbackT)(*
*CL_IN ClInvocationT invocation,*
*CL_IN const ClNameT ∗pCompName)*

Component Manager requests the component, `compName`, to gracefully shutdown using this callback function. This is an asynchronous function with reply.

*invocation:* Invocation of this callback function.

*pCompName:* Pointer to the name of the component that must gracefully terminate.

### 3.2.9 ClCpmAttrIdsT

```
typedef enum {
        CL_CPM_COMP_NAME = 1,
        CL_CPM_COMP_OPERATIONAL_STATE,
        CL_CPM_COMP_PRESENCE_STATE,
        CL_CPM_SU_NAME,
        CL_CPM_SU_OPERATIONAL_STATE,
        CL_CPM_SU_PRESENCE_STATE,
        CL_CPM_NODE_NAME
} ClCpmAttrIdsT;
```

The enumeration, `ClCpmAttrIdsT`, contains the attributes of a component.

**Enumeration values:**

***CL_CPM_COMP_NAME*** - Component class attribute for name of the component.

***CL_CPM_COMP_OPERATIONAL_STATE*** - Component class attribute for its operational state.

***CL_CPM_COMP_PRESENCE_STATE*** - Component class attribute for its presence state.

***CL_CPM_SU_NAME*** - Service Unit class attribute for its name.

***CL_CPM_SU_OPERATIONAL_STATE*** - Service Unit class attribute for its operational state.

***CL_CPM_SU_PRESENCE_STATE*** - Service Unit class attribute for its presence state.

***CL_CPM_NODE_NAME*** - Node class attribute for name.

### 3.2.10 ClCpmClassIdT

```
typedef enum {
        CL_CPM_CLASS_UNKNOWN = 0,
        CL_CPM_CLASS_CLUSTER = 0x7d02,
        CL_CPM_CLASS_NODE = 0x7d03,
        CL_CPM_CLASS_SU = 0x7d04,
        CL_CPM_CLASS_COMP = 0x7d05
} ClCpmClassIdT;
```

The enumeration, `ClCpmClassIdT`, contains the COR Class ID for the SA Forum information model.

**Enumeration values:**

***CL_CPM_CLASS_UNKNOWN***

***CL_CPM_CLASS_CLUSTER*** - Cluster MO Class ID.

***CL_CPM_CLASS_NODE*** - Node MO Class ID.

***CL_CPM_CLASS_SU*** - Service Unit MO Class ID.

***CL_CPM_CLASS_COMP*** - Component MO Class ID.

### 3.2.11 clCpmCompProcessRel

```
typedef enum clCpmCompProcessRel {
        CL_CPM_COMP_NONE = 0,
        CL_CPM_COMP_MULTI_PROCESS = 1,
        CL_CPM_COMP_SINGLE_PROCESS = 2,
        CL_CPM_COMP_THREADED = 3
} ClCpmCompProcessRelT;
```

The enumeration, `ClCpmCompProcessRelT`, represents the relationship between the component and the process.

**Enumeration values:**

***CL_CPM_COMP_NONE*** - This indicates that the component does not have any execution context.

***CL_CPM_COMP_MULTI_PROCESS*** - This indicates that the component consists of multiple processes.

***CL_CPM_COMP_SINGLE_PROCESS*** - This indicates the that component consists of a single process.

***CL_CPM_COMP_THREADED*** - This indicates that the component consists of multiple threads, but the process does not belong to the component.

### 3.2.12 ClCpmCompRequestTypeT

```
typedef enum {
        CL_CPM_HEALTHCHECK = 1,
        CL_CPM_TERMINATE = 2,
        CL_CPM_PROXIED_INSTANTIATE = 3,
        CL_CPM_PROXIED_CLEANUP = 4,
        CL_CPM_EXTN_HEALTHCHECK = 5,
        CL_CPM_INSTANTIATE = 6,
        CL_CPM_CLEANUP = 7,
        CL_CPM_RESTART = 8
} ClCpmCompRequestTypeT;
```

The enumeration, `ClCpmCompRequestTypeT`, defines the request types executed by the Component Manager, using the registered function callback.

**Enumeration values:**

***CL_CPM_HEALTHCHECK*** - SA Forum compliant health check request.

***CL_CPM_TERMINATE*** - SA Forum and Proxied component terminate request.

***CL_CPM_PROXIED_INSTANTIATE*** - Proxied component instantiation request.

***CL_CPM_PROXIED_CLEANUP*** - Proxied component cleanup request.

***CL_CPM_EXTN_HEALTHCHECK*** - Component extensive healthcheck request.

***CL_CPM_INSTANTIATE*** - SA Forum ot Non-proxied component instantiation request.

***CL_CPM_CLEANUP*** - Component cleanup request.

***CL_CPM_RESTART*** - Component restart request.

### 3.2.13 ClCpmCSISetCallbackT

```
typedef enum {
        CL_CPM_COMP_EVENT,
        CL_CPM_NODE_EVENT
} ClCpmEventTypeT;
```

The enumeration, `ClCpmEventTypeT`, contains the types of the events published by the Component Manager.

**Enumeration values:**

***CL_CPM_COMP_EVENT*** - Event that is published when a component fails.

***CL_CPM_NODE_EVENT*** - Event that is published when a node arrives/departs.

### 3.2.14 ClCpmSlotInfoFieldIdT

```
typedef enum {
        CL_CPM_SLOT_ID,
        CL_CPM_IOC_ADDRESS,
        CL_CPM_NODE_MOID,
        CL_CPM_NODENAME
} ClCpmSlotInfoFieldIdT;
```

The enumeration, `ClCpmSlotInfoFieldIdT`, indicates which field of `ClCpmSlotInfoT` is set and other information related to it.

**Enumeration values:**

***CL_CPM_SLOT_ID***

***CL_CPM_IOC_ADDRESS***

***CL_CPM_NODE_MOID***

***CL_CPM_NODENAME***

### 3.2.15 ClEoFunctionsIdT

```
typedef enum {
        CL_EO_INITIALIZE_COMMON_FN_ID = 0,
        CL_EO_GET_STATE_COMMON_FN_ID =
            CL_EO_GET_FULL_FN_NUM(CL_EO_EO_MGR_CLIENT_TABLE_ID, 0),
        CL_EO_SET_STATE_COMMON_FN_ID =
            CL_EO_GET_FULL_FN_NUM(CL_EO_EO_MGR_CLIENT_TABLE_ID, 1),
        CL_EO_IS_ALIVE_COMMON_FN_ID =
            CL_EO_GET_FULL_FN_NUM(CL_EO_EO_MGR_CLIENT_TABLE_ID, 2),
        CL_EO_GET_RMD_STATS =
            CL_EO_GET_FULL_FN_NUM(CL_EO_EO_MGR_CLIENT_TABLE_ID, 3),
        CL_EO_SET_PRIORITY =
            CL_EO_GET_FULL_FN_NUM(CL_EO_EO_MGR_CLIENT_TABLE_ID, 4),
        CL_EO_LOG_LEVEL_SET =
            CL_EO_GET_FULL_FN_NUM(CL_EO_EO_MGR_CLIENT_TABLE_ID, 5),
        CL_EO_LOG_LEVEL_GET =
                CL_EO_GET_FULL_FN_NUM(CL_EO_EO_MGR_CLIENT_TABLE_ID, 6),
        CL_EO_LAST_COMMON_FN_ID
} ClEoFunctionsIdT;
```

The enumeration, `ClEoFunctionsIdT`, defines the RMD number for the Component Manager EO management client installed on each EO.

**Enumeration values:**

***CL_EO_INITIALIZE_COMMON_FN_ID*** - EO initialization function.

***CL_EO_GET_STATE_COMMON_FN_ID*** - EO Get State.

***CL_EO_SET_STATE_COMMON_FN_ID*** - EO Set State.

***CL_EO_IS_ALIVE_COMMON_FN_ID*** - Pings an EO.

***CL_EO_GET_RMD_STATS*** - Returns RMD state of an EO.

***CL_EO_SET_PRIORITY*** - Sets the thread priority of EO Threads.

***CL_EO_LOG_LEVEL_SET*** - Sets the log level of the EO.

***CL_EO_LOG_LEVEL_GET*** - Retrieves the log level of the EO.

***CL_EO_LAST_COMMON_FN_ID*** - End of EO Function.

## 3.3 Functional APIs

### 3.3.1 clCkptLibraryCkptCreate

#### clCpmFuncEOWalk

**Synopsis:**
Performs a walk operation through all the EOs.

**Header File:**
clCpmApi.h

**Syntax:**

```
extern ClRcT clCpmFuncEOWalk(
                CL_IN ClCpmFuncWalkT *pWalk);
```

**Parameters:**
***pWalk:*** Contains information regarding the walk.

**Return values:**
***CL_OK:*** The function executed successfully.

***CL_CPM_ERR_NULL_PTR:*** `pWalk` is a NULL pointer.

***CL_ERR_VERSION_MISMATCH:*** The version of the client and server is incompatible.

**Description:**
This function is used to perform a walk operation on all the components registered with the Component Manager. It invokes the function with arguments, such as the function number and input buffer in `pWalk`, on all the components registered with the component manager.

**Note:** This is a synchronous function.

**Library Files:**
ClAmfClient

**Related API(s):**
clCpmExecutionObjectStateSet

## 3.3.2 clCkptLibraryCkptCreate

**clCpmExecutionObjectStateSet**

**Synopsis:**
Sets the state of an EO.

**Header File:**
clCpmApi.h

**Syntax:**
```
        extern ClRcT clCpmExecutionObjectStateSet(
                    CL_IN ClIocNodeAddressT destAddr,
                    CL_IN ClUint32T eoId,
                    CL_IN ClEoStateT state);
```

**Parameters:**
***destAddr:*** Address of the Component Manager.

***eoId:*** ID of the EO.

***state:*** New state being assigned to the EO.

**Return values:**
***CL_OK:*** The function executed successfully.

**Description:**
This function is used to set the state of an EO, as specified by the parameter `state`. After calling this function, the component, identified by `eoId`, on the node having the address `destAddr`, is set to the state, `state`.

**Note:** This is a synchronous function.

**Library Files:**
ClAmfClient

**Related API(s):**
None.

## 3.3.3 clCkptLibraryCkptCreate

**clCpmBootLevelGet**

**Synopsis:**
Retrieves the current boot-level of a node.

**Header File:**
clCpmApi.h

**Syntax:**

```
extern ClRcT clCpmBootLevelGet(
            CL_IN ClNameT     *pNodeName,
            CL_OUT ClUint32T  *pBootLevel);
```

**Parameters:**

***pNodeName:*** Pointer to the name of the node whose boot-level is to be returned.

***pBootLevel:*** (out) Boot level of the node, `pNodeName`.

**Return values:**

***CL_OK:*** The function executed successfully.

***CL_ERR_NULL_POINTER:*** `pNodeName` or `pBootLevel` is a NULL pointer.

**Description:**
This function is used to return the current boot-level of the node. The boot-level returned by this function indicates the level upto which the node, `pNodeName`, has been re-booted.

**Note:** This is a synchronous function.

**Library Files:**
ClAmfClient

**Related API(s):**
clCpmBootLevelSet, clCpmBootLevelMax

### 3.3.4 clCkptLibraryCkptCreate

#### clCpmBootLevelSet

**Synopsis:**
Sets the boot-level of a node.

**Header File:**
clCpmApi.h

**Syntax:**

```
extern ClRcT clCpmBootLevelSet(
            CL_IN ClNameT            *pNodeName,
            CL_IN ClCpmLcmReplyT     *pSrcInfo,
            CL_IN ClUint32T          bootLevel);
```

**Parameters:**

***pNodeName:*** Pointer to the name of the node whose boot-level is to be set.

***pSrcInfo:*** Pointer to the source information, where the reply to the requested operation is expected. If a response is not required, this can be set to NULL.

***bootLevel:*** Boot-level to be set for the node.

**Return values:**

***CL_OK:*** The function executed successfully.

***CL_ERR_NULL_POINTER:*** `pNodeName` or `pSrcInfo` is a NULL pointer.

**Description:**
This function is used to set the boot-level of the node. If `bootLevel` is greater than the current boot-level, CPM boots all the service units mentioned in subsequent boot-levels up to the required boot-level. If `bootLevel` is lesser than the current boot-level, CPM terminates all the service units listed above the `bootLevel`.

**Note:** This is an asynchronous function.

**Library Files:**
ClAmfClient

**Related API(s):**
clCpmBootLevelGet , clCpmBootLevelMax

### 3.3.5 clCkptLibraryCkptCreate

#### clCpmBootLevelMax

**Synopsis:**
Returns the maximum boot-level of a node.

**Header File:**
clCpmApi.h

**Syntax:**

```
        extern ClRcT clCpmBootLevelMax(
                    CL_IN ClNameT     *pNodeName,
                    CL_OUT ClUint32T  *pBootLevel);
```

**Parameters:**

***pNodeName:*** Pointer to the name of the node whose boot-level is returned.

***pBootLevel:*** (out) Maximum boot-level of the node `pNodeName`.

**Return values:**

***CL_OK:*** The function executed successfully.

***CL_ERR_NULL_POINTER:*** `pNodeName` or `pBootLevel` contains a NULL pointer.

**Description:**
This function is used to return the maximum possible boot-level of the node. This indicates the maximum level upto which the node, `pNodeName`, can be rebooted.

**Note:** This is a synchronous function.

**Library Files:**
ClAmfClient

**Related API(s):**
clCpmBootLevelGet, clCpmBootLevelSet

### 3.3.6 clCkptLibraryCkptCreate

#### clCpmClientInitialize

**Synopsis:**
Initializes the client Component Manager library.

**Header File:**
clCpmApi.h

**Syntax:**

```
extern ClRcT clCpmClientInitialize(
            CL_IN     ClCpmHandleT            *pCpmHandle,
            CL_IN     const ClCpmCallbacksT   *pCallback,
            CL_INOUT  ClVersionT              *pVersion)
```

**Parameters:**

***pCpmHandle:*** Handle of the component.

***pCallback:*** Callback functions provided by the application to the Component Manager.

***pVersion:*** Required version number.

**Return values:**

***CL_OK:*** The function executed successfully.

***CL_ERR_NULL_POINTER:*** `pCpmHandle`, `pCallback`, or `pVersion` contains a NULL pointer.

**Description:**
Before invoking this function, the EO must be created. This function is used to update the EO client table internally based on `ClCpmCallbacksT`. It also ensures that the version of Component Manager client and Server Libraries are compatible.

**Note:**
This function is equivalent to `saAmfInitialize`, which makes a synchronous call.

**Library Files:**
ClAmfClient

**Related API(s):**
clCpmClientFinalize

### 3.3.7 clCkptLibraryCkptCreate

#### clCpmClientFinalize

**Synopsis:**
Frees the client Component Manager library.

**Header File:**
clCpmApi.h

**Syntax:**
```
extern ClRcT clCpmClientFinalize(
            CL_IN ClCpmHandleT cpmHandle);
```

**Parameters:**
***cpmHandle:*** Handle returned by `clCpmClientInitialize()` function.

**Return values:**
***CL_OK:*** The function executed successfully.

**Description:**
This function is used to free the client Component Manager library. It releases all the resources acquired when `clCpmClientInitialize()` function was called. After this function is executed, `cpmHandle` becomes invalid.

**Note:**
This function is equivalent to `saAmfFinalize`, which makes an asynchronous call.

**Library Files:**
ClAmfClient

**Related API(s):**
clCpmClientInitialize

### 3.3.8 clCkptLibraryCkptCreate

#### clCpmSelectionObjectGet

**Synopsis:**
Returns an operating system handle for detecting pending callbacks.

**Header File:**
clCpmApi.h

**Syntax:**

```
extern ClRcT clCpmSelectionObjectGet(
            CL_IN     ClCpmHandleT        cpmHandle,
            CL_OUT    ClSelectionObjectT  *pSelectionObject);
```

**Parameters:**

***cpmHandle:*** Handle returned by `clCpmClientInitialize()` function.

***pSelectionObject:*** Pointer to the operating system handle that the process can use to detect pending callbacks.

**Return values:**

***CL_OK:*** The function executed successfully.

**Description:**
This function returns the operating system handle, `pSelectionObject`, associated with the handle, `cpmHandle`. The process can use this operating system handle to detect pending callbacks, instead of repeatedly invoking `clCpmDispatch()` for this purpose.
The operating system handle returned by this function is a file descriptor that is used with `poll()` or `select()` systems call to detect incoming callbacks.
The `pSelectionObject`, returned by this function, is valid until `clCpmClient-Finalize()` is invoked on the same handle `cpmHandle`.

**Library Files:**
ClAmfClient

**Related API(s):**
clCpmClientInitialize, clCpmDispatch

### 3.3.9 clCkptLibraryCkptCreate

#### clCpmDispatch

**Synopsis:**

Invokes pending callbacks.

**Header File:**

clCpmApi.h

**Syntax:**

```
extern ClRcT clCpmDispatch(
            CL_IN      ClCpmHandleT        cpmHandle,
            CL_OUT     ClDispatchFlagsT    dispatchFlags);
```

**Parameters:**

***cpmHandle:*** Handle returned by `clCpmClientInitialize()` function.

***dispatchFlags:*** Flags that specify the callback execution behaviour of this function.

**Return values:**

***CL_OK:*** The function executed successfully.

**Description:**

This function invokes, in the context of the calling EO, pending callbacks for the handle, `cpmHandle`, as specified by the `dispatchFlags` parameter.

**Library Files:**

ClAmfClient

**Related API(s):**

clCpmClientInitialize , clCpmSelectionObjectGet

### 3.3.10 clCkptLibraryCkptCreate

#### clCpmComponentRegister

**Synopsis:**
Registers a component with CPM.

**Header File:**
clCpmApi.h

**Syntax:**

```
extern ClRcT clCpmComponentRegister(
            CL_IN ClCpmHandleT    cpmHandle,
            CL_IN const ClNameT   *pCompName,
            CL_IN const ClNameT   *pProxyCompName);
```

**Parameters:**

***cpmHandle:*** Handle returned by the `clCpmClientInitialize()` function.

***pCompName:*** Name of the component.

***pProxyCompName:*** Name of the component that is a proxy for `pCompName`.

**Return values:**

***CL_OK:*** The function executed successfully.

***CL_ERR_NULL_POINTER:*** `pCompName` or `pProxyCompName` contains a NULL pointer.

***CL_CPM_ERR_INIT:*** The callbacks are provided during initialization is not correct.

***CL_CPM_ERR_EXIST:*** The component has already been registered.

**Description:**
This function is used to register a component with CPM. It can also be used by a proxy component to register a proxied component. By calling this function, the component indicates that it is up and running, and is ready to provide service.

**Note:**
This function is equivalent to `saAmfComponentRegister`, which makes a synchronous call.

**Library Files:**
ClAmfClient

**Related API(s):**
clCpmComponentUnregister

### 3.3.11 clCkptLibraryCkptCreate

**clCpmComponentUnregister**

**Synopsis:**
De-registers a component.

**Header File:**
clCpmApi.h

**Syntax:**

```
extern ClRcT clCpmComponentUnregister(
            CL_IN ClCpmHandleT    cpmHandle,
            CL_IN const ClNameT   *pCompName,
            CL_IN const ClNameT   *pProxyCompName);
```

**Parameters:**

***cpmHandle:*** Handle returned by the `clCpmClientInitialize()` function.

***pCompName:*** Name of the component.

***pProxyCompName:*** Name of the component proxied by the component, `pCompName`.

**Return values:**

***CL_OK:*** The function executed successfully.

***CL_ERR_NULL_POINTER:*** `pCompName` or `pProxyCompName` contains a NULL pointer.

***CL_CPM_ERR_BAD_OPERATION:*** The component, `pCompName`, has not de-registered all the proxied components or `proxyComp` is not the proxy for `pCompName`.

**Description:**
This function is used for two purposes: A proxy component can de-register one of its proxied components or a component can de-register itself. The `cpmHandle` in this function must be same as that used in the corresponding `clCpmComponentRegister()` call.

**Note:**
This function is equivalent to `saAmfComponentUnregister()`, which makes an asynchronous call.

**Library Files:**
ClAmfClient

**Related API(s):**
clCpmClientInitialize, clCpmComponentRegister

### 3.3.12 clCkptLibraryCkptCreate

**clCpmComponentNameGet**

**Synopsis:**
Retrieves the component name.

**Header File:**
clCpmApi.h

**Syntax:**

```
extern ClRcT clCpmComponentNameGet(
            CL_IN ClCpmHandleT cpmHandle,
            CL_OUT ClNameT     *pCompName);
```

**Parameters:**

***cpmHandle:*** Handle returned by the `clCpmClientInitialize()` function.

***pCompName:*** Name of the component.

**Return values:**

***CL_OK:*** The function executed successfully.

***CL_ERR_NULL_POINTER:*** `pCompName` contains a NULL pointer.

**Description:**
This function returns the name of the component that the calling process belongs to. This function can be invoked by the process before its component is registered with CPM. The name of the component provided by this function should be used by a process, when it registers its local component.

**Note:**
This function is equivalent to `saAmfComponentNameGet()`.

**Library Files:**
ClAmfClient

**Related API(s):**
clCpmClientInitialize, clCpmComponentRegister

### 3.3.13 clCkptLibraryCkptCreate

#### clCpmResponse

**Synopsis:**
Returns the response with the corresponding handle and `invocation`.

**Header File:**
clCpmApi.h

**Syntax:**

```
extern ClRcT clCpmResponse(
            CL_IN ClCpmHandleT    cpmHandle,
            CL_IN ClInvocationT   invocation
            CL_IN CLRcT           rc);
```

**Parameters:**

***cpmHandle:*** Handle returned by the `clCpmClientInitialize()` function.

***invocation:*** Associates an invocation of this response function with a particular invocation of a callback function by CPM.

**Return values:**

***CL_OK:*** The function executed successfully.

**Description:**
This function is used to provide a response about the success or failure of the callback invoked by CPM. The component responds to CPM with the result of its execution of a particular request of the CPM, identified by `invocation`.

**Note:**
This function is equivalent to `saAmfResponse()`.

**Library Files:**
ClAmfClient

**Related API(s):**
clCpmClientInitialize, clCpmComponentRegister

### 3.3.14 clCkptLibraryCkptCreate

#### clCpmHAStateGet

**Synopsis:**
Returns the HA state of the component.

**Header File:**
clCpmApi.h

**Syntax:**

```
extern ClRcT clCpmHAStateGet(
              CL_IN ClCpmHandleT    cpmHandle,
              CL_IN   ClNameT          *compName,
              CL_IN   ClNameT          *csiName,
              CL_OUT  ClAmsHAStateT    *haState);
```

**Parameters:**

***cpmHandle:*** Handle returned by the `clCpmClientInitialize()` API.

***compName:*** Pointer to the name of the component for which the information is requested.

***csiName:*** Pointer to the name of the component service instance for which the information is requested.

***haState:*** Pointer to the HA state that the AMS is assigning to the component, `compName`, on behalf of the component service instance, `csiName`.

**Return values:**

***CL_OK:*** The function executed successfully.

***CL_ERR_NULL_POINTER:*** `compName`, `csiName`, or `haState` contains a NULL pointer.

***CL_ERR_DOESNT_EXIST:*** CPM library is not able to retrieve the nodeName.

**Description:**
This function returns the HA state of a component, `compName`, on behalf of the component service instance, `csiName`. The HA state of the component indicates if it is currently responsible to provide service characterized by the component service instance assigned to it, if it is a standby, or if it is in the quiesced state.

**Library Files:**
libClAmfClient

**Related API(s):**
clCpmClientInitialize

### 3.3.15 clCkptLibraryCkptCreate

#### clCpmCSIQuiescingComplete

**Synopsis:**
Returns the status of HA state quiescing with the corresponding handle and `invocation`.

**Header File:**
clCpmApi.h

**Syntax:**

```
extern ClRcT clCpmCSIQuiescingComplete(
            CL_IN ClCpmHandleT    cpmHandle,
            CL_IN ClInvocationT   invocation
            CL_IN CLRcT           rc);
```

**Parameters:**

***cpmHandle:*** Handle returned by the `clCpmClientInitialize()` function.

***invocation:*** Associates the `invocation` of this function with a particular invocation of a callback function by CPM.

**Return values:**

***CL_OK:*** The function executed successfully.

**Description:**
This function is used to respond with the success and failure information of the callback invoked by CPM, for quiescing the HA state of a given CSI. The AMS requests to enter the `CL_AMS_HA_STATE_QUIESCING` HA state for that particular component service instance or to all component service instances. Using this call, a component can notify CPM that it has successfully stopped its activity related to a particular component service instance or to all component service instances assigned to it.

**Note:**
This function is equivalent to `saAmfCSIQuiescingComplete()`.

**Library Files:**
ClAmfClient

**Related API(s):**
clCpmClientInitialize clCpmComponentRegister clCpmResponse

### 3.3.16 clCpmComponentFailureReport

**clCpmComponentFailureReport**

**Synopsis:**
Notifies about the failed component.

**Header File:**
clCpmApi.h

**Syntax:**

```
extern ClRcT clCpmComponentFailureReport(
            CL_IN ClCpmHandleT                 cpmHandle,
            CL_IN const ClNameT                *pCompName,
            CL_IN ClTimeT                      errorDetectionTime,
            CL_IN ClAmsRecommendedRecoveryT    recommendedRecovery,
            CL_IN ClUint32T                    alarmHandle)
```

**Parameters:**

***cpmHandle:*** Handle returned by `clCpmClientInitialize()` function.

***pCompName:*** Pointer to the name of the component for which the failure is notified.

***errorDetectionTime:*** Time when the error is detected.

***recommondedRecovery:*** Recommonded recovery to be performed by AMF.

***alarmHandle:*** Key returned by the `clAlarmRaise()` function.

**Return values:**

***CL_OK:*** The function executed successfully.

***CL_ERR_NULL_POINTER:*** `pCompName` contains a NULL pointer.

**Description:**
This function is used to notify the Component Manager about the failure of a component. It reports an error and provides a recovery recommendation to the AMF. AMF responds to the error report and performs a recovery operation to retain the availability of component service instances supported by the erroneous component. AMF does not perform a recovery action other than that recommended by this function, but it can decide to escalate to the higher recovery level.

**Note:**
This function is equivalent to `saAmfComponentFailureReport()`, which makes a synchronous call.

**Library Files:**
ClAmfClient

**Related API(s):**
clCpmClientInitialize, clCpmComponentFailureClear

### 3.3.17 clCpmComponentFailureClear

**clCpmComponentFailureClear**

**Synopsis:**
Notifies about the restoration of the failed component.

**Header File:**
clCpmApi.h

**Syntax:**

```
extern ClRcT clCpmComponentFailureClear(
            CL_IN ClCpmHandleT    cpmHandle,
            CL_IN ClNameT         *pCompName)
```

**Parameters:**
***cpmHandle:*** Handle returned by the `clCpmClientInitialize()` function.

***pCompName:*** Pointer to the name of the component for which the failure is notified.

**Return values:**
***CL_OK:*** The function executed successfully.

***CL_ERR_NULL_POINTER:*** `pCompName` contains a NULL pointer.

**Description:**
This function is used to notify the Component Manager that the failed component is restored. It cancels the failure notification made by the `clCpmComponentFailureReport()` function.

**Note:**
This function is equivalent to `saAmfComponentFailureClear()`, which makes a synchronous call.

**Library Files:**
ClAmfClient

**Related API(s):**
clCpmClientInitialize, clCpmComponentFailureReport

### 3.3.18 clCpmProtectionGroupTrack

**clCpmProtectionGroupTrack**

**Synopsis:**
Tracks the protection group for the given CSI.

**Header File:**
clCpmApi.h

**Syntax:**

```
extern ClRcT clCpmProtectionGroupTrack(
                  CL_IN    ClCpmHandleT              cpmHandle,
                  CL_IN    ClNameT                   *csiName,
                  CL_IN    ClUint8T                  trackFlags,
                  CL_INOUT  ClAmsPGNotificationT    *notificationBuffer);
```

**Parameters:**

***cpmHandle:*** Handle returned by `clCpmClientInitialize()` function.

***pCsiName:*** Pointer to the CSI for which the protection group needs to be tracked.

***trackFlags:*** The type of tracking that is requested. This is bitwise OR of one or more of the flags `CL_AMS_PG_TRACK_CURRENT`, `CL_AMS_PG_TRACK_CHANGES`, or `CL_-AMS_PG_TRACK_CHANGES_ONLY`.

***pNotificationBuffer:*** Pointer to a buffer of type `ClAmsPGNotificationT`. This parameter is ignored, if `CL_AMS_PG_TRACK_CURRENT` is not set in `trackFlags`; otherwise, if `notificationBuffer` is not NULL, the buffer will contain information about all components in the protection group when `clCpmProtectionGroupTrack()` returns.

**Return values:**

***CL_OK:*** The function executed successfully.

***CL_ERR_NULL_POINTER:*** `csiName` or `notificationBuffer` contains a NULL pointer.

**Description:**
This function is used to request AMF to start tracking changes in the protection group associated with the component service instance, `csiName`, or changes to attributes of any component in the protection group. These changes are notified through the invocation of the `ClCpmProtectionGroupTrackCallbackT()` callback function, obtained using the `clCpmClientInitialize()` function.

**Note:**
This function is equivalent to `saAmfProtectionGroupTrack()`, which makes a synchronous call.

**Library Files:**
ClAmfClient

**Related API(s):**
clCpmClientInitialize, clCpmProtectionGroupTrackStop

### 3.3.19 clCpmProtectionGroupTrackStop

#### clCpmProtectionGroupTrackStop

**Synopsis:**
Stops tracking the protection group for the given CSI.

**Header File:**
clCpmApi.h

**Syntax:**

```
extern ClRcT clCpmProtectionGroupTrackStop(
                  CL_IN   ClCpmHandleT             cpmHandle,
                  CL_IN   ClNameT                  *pCsiName)
```

**Parameters:**

***cpmHandle:*** Handle returned by `clCpmClientInitialize()` function.

***pCsiName:*** Pointer to the CSI for which protection group needs to be tracked.

**Return values:**

***CL_OK:*** The function executed successfully.

***CL_ERR_NULL_POINTER:*** `pCsiName` contains a NULL pointer.

**Description:**
The invoking process requests Availability Management Framework to stop tracking protection group changes for the component service instance, `pCsiName`.

**Note:**
This function is equivalent to `saAmfProtectionGroupTrackStop()`, which makes a synchronous call.

**Library Files:**
ClAmfClient

**Related API(s):**
clCpmClientInitialize, clCpmProtectionGroupTrack

### 3.3.20 clCpmComponentInstantiate

**clCpmComponentInstantiate**

**Synopsis:**
Instantiates a component.

**Header File:**
clCpmApi.h

**Syntax:**

```
extern ClRcT clCpmComponentInstantiate(
            CL_IN ClNameT            *pCompName,
            CL_IN ClNameT            *pNodeName,
            CL_IN ClCpmLcmReplyT     *pSrcInfo);
```

**Parameters:**

***pCompName:*** Name of the component being instantiated.

***pNodeName:*** Pointer to the name of the node where `pCompName` exists.

***pSrcInfo:*** Pointer to the source information, where the reply to the requested operation is expected. If a response is not required, this can be set to NULL.

**Return values:**

***CL_OK:*** The function executed successfully.

***CL_ERR_NULL_POINTER:*** `pCompName`, `pNodeName`, or `pScrInfo` contains a NULL pointer.

**Description:**
This function is used to instantiate the component, `pCompName`, on the node, `pNodeName`. If `pSrcInfo` is NULL, the reply regarding the success or failure in instantiating the component, is not sent. Otherwise, the reply is sent to the requested port through an RMD call, as specified by the attributes of the structure `pSrcInfo`.

**Note:** This is an asynchronous function.

**Library Files:**
ClAmfClient

**Related API(s):**
clCpmComponentTerminate, clCpmComponentCleanup, clCpmComponentRestart

### 3.3.21 clCpmComponentTerminate

**clCpmComponentTerminate**

**Synopsis:**
Terminates the component.

**Header File:**
clCpmApi.h

**Syntax:**

```
extern ClRcT clCpmComponentTerminate(
            CL_IN ClNameT               *pCompName,
            CL_IN ClNameT               *pNodeName,
            CL_IN ClCpmLcmReplyT        *pSrcInfo);
```

**Parameters:**

***pCompName:*** Name of the component being terminated.

***pNodeName:*** Pointer to the name of the node, where `pCompName` exist.

***pSrcInfo:*** Pointer to the source information, where the reply of the requested operation is expected. If a response is not required, this can be set to NULL.

**Return values:**

***CL_OK:*** The function executed successfully.

***CL_ERR_NULL_POINTER:*** `pCompName`, `pNodeName`, or `pScrInfo` contains a NULL pointer.

**Description:**
This function is used to terminate (graceful shut-down) the component, `pCompName`, on the node `pNodeName`. If `pSrcInfo` is NULL, the reply regarding the success or failure in terminating the component is not sent. Otherwise, a reply is sent to the requested port through an RMD call, as specified by the attrbutes of the structure, `pSrcInfo`.

**Note:** This is an asynchronous function.

**Library Files:**
ClAmfClient

**Related API(s):**
clCpmComponentInstantiate, clCpmComponentCleanup, clCpmComponentRestart

### 3.3.22 clCpmComponentCleanup

**clCpmComponentCleanup**

**Synopsis:**
Frees a component.

**Header File:**
clCpmApi.h

**Syntax:**

```
extern ClRcT clCpmComponentCleanup(
            CL_IN ClNameT              *pCompName,
            CL_IN ClNameT              *pNodeName,
            CL_IN ClCpmLcmReplyT       *pSrcInfo);
```

**Parameters:**

***pCompName:*** Name of the component to be freed.

***pNodeName:*** Pointer to the name of the node where `pCompName` exists.

***pSrcInfo:*** Pointer to the source information, where the reply of the requested operation is expected. If a response is not required, this can be set to NULL.

**Return values:**

***CL_OK:*** The function executed successfully.

***CL_ERR_NULL_POINTER:*** `pCompName`, `pNodeName`, or `pScrInfo` contains a NULL pointer.

**Description:**
This function is used to free (abrupt shutdown) the component, `pCompName`, on the node, `pNodeName`. If `pSrcInfo` is NULL, the reply regarding the success or failure in freeing the component is not sent. Otherwise, a reply is sent to the requested port through an RMD call, as specified by the attributes of the structure, `pSrcInfo`.

**Note:** This is an asynchronous function.

**Library Files:**
ClAmfClient

**Related API(s):**
clCpmComponentInstantiate, clCpmComponentTerminate, clCpmComponentRestart

### 3.3.23 clCpmComponentRestart

**clCpmComponentRestart**

**Synopsis:**
Restarts the component.

**Header File:**
clCpmApi.h

**Syntax:**

```
extern ClRcT clCpmComponentRestart(
            ClNameT                  *pCompName,
            CL_IN ClNameT            *pNodeName,
            CL_IN ClCpmLcmReplyT     *pSrcInfo);
```

**Parameters:**

***pCompName:*** Name of the component that needs to be restarted.

***pNodeName:*** Pointer to the name of the node where `pCompName` exists.

***pSrcInfo:*** Pointer to the source information, where the reply of the requested operation is expected. If a response is not required, this can be set to NULL.

**Return values:**

***CL_OK:*** The function executed successfully.

***CL_ERR_NULL_POINTER:*** `pCompName`, `pNodeName`, or `pScrInfo` contains a NULL pointer.

**Description:**
This function is used to restart a given component, `pCompName`, on node `pNodeName`. If `pSrcInfo` is NULL, the reply regarding the success or failure in restarting the component is not sent. Otherwise, a reply is sent to the requested port through an RMD call, as specified by the attributes of the structure, `pSrcInfo`. The complete restart process includes, termination of the component, freeing it and instantiating the component back to the earlier state.

**Note:** This is an asynchronous function.

**Warning:**
The cleanup process will get executed only when (graceful) termination is failed. In this case, termination and cleanup will occur without deleting the communication port.

**Library Files:**
ClAmfClient

**Related API(s):**
clCpmComponentInstantiate, clCpmComponentTerminate, clCpmComponentCleanup

### 3.3.24 clCpmComponentIdGet

**clCpmComponentIdGet**

**Synopsis:**
Returns the ID of the component.

**Header File:**
clCpmApi.h

**Syntax:**

```
extern ClRcT clCpmComponentIdGet(
            CL_IN ClCpmHandleT     cpmHandle,
            CL_IN ClNameT          *pCompName,
            CL_OUT ClUint32T       *pCompId);
```

**Parameters:**

***cpmHandle:*** Handle returned by `clCpmClientInitialize()` function.

***pCompName:*** Name of the component.

***pCompId:*** (out) Unique ID of the component.

**Return values:**

***CL_OK:*** The function executed successfully.

***CL_ERR_NULL_POINTER:*** `pCompName` or `pCompId` contains a NULL pointer.

**Description:**
This function is used to return the component ID for a component, `pCompName`. This unique component ID is internal to the node and is not known to outside entities. Every time a component is instantiated, a new component ID is assigned to it.

**Note:** This is a synchronous function.

**Library Files:**
ClAmfClient

**Related API(s):**
None

### 3.3.25 clCpmComponentAddressGet

**clCpmComponentAddressGet**

**Synopsis:**
Returns the IOC address of a component.

**Header File:**
clCpmApi.h

**Syntax:**

```
extern ClRcT clCpmComponentAddressGet(
            CL_IN  ClIocNodeAddressT      nodeAddress,
            CL_IN  ClNameT                *pCompName,
            CL_OUT ClIocAddressT          *pCompAddress);
```

**Parameters:**
***nodeAddress:*** IOC address of the node.

***pCompName:*** Name of the component.

***pCompAddress:*** IOC address of the component including port information.

**Return values:**
***CL_OK:*** The function executed successfully.

***CL_ERR_NULL_POINTER:*** `pCompName` or `pCompAddress` contains a NULL pointer.

**Description:**
This function is used to return the IOC address of the component, `pCompName`. This is the physical address of the node where CPM is running.

**Note:** This is a synchronous function.

**Library Files:**
ClAmfClient

**Related API(s):**
clCpmComponentStatusGet

### 3.3.26 clCpmComponentStatusGet

**clCpmComponentStatusGet**

**Synopsis:**

Returns the component presence and operational state.

**Header File:**

clCpmApi.h

**Syntax:**

```
extern ClRcT clCpmComponentStatusGet(
               CL_IN   ClNameT                 *pCompName,
               CL_IN   ClNameT                 *pNodeName,
               CL_OUT  ClAmsPresenceStateT     *pPresenceState,
               CL_OUT  ClAmsOperationalStateT  *pOperationalState);
```

**Parameters:**

***pCompName:*** Name of the component.

***pNodeName:*** IOC address of the component.

***pPresenceState:*** Presence state of the component.

***pOperationalState:*** Operational state of the component.

**Return values:**

***CL_OK:*** The function executed successfully.

***CL_ERR_NULL_POINTER:*** `pCompName`, `pNodeName`, `pPresenceState` or `pOperationalState` contains a NULL pointer.

**Description:**

This function is used to return the presence and operational state of the component, `pCompName`. The presence state of the component reflects the component life cycle and the operational state of the component is used by AMF to determine if a component is capable of handling component service instance assignments.

**Note:** This is a synchronous function.

**Library Files:**

ClAmfClient

**Related API(s):**

clCpmComponentAddressGet

## 3.3.27 clCpmSUInstantiate

**clCpmSUInstantiate**

**Synopsis:**
Instantiates a given Service Unit (SU).

**Header File:**
clCpmApi.h

**Syntax:**

```
extern ClRcT clCpmSUInstantiate(
            CL_IN ClNameT               *pSuName,
            CL_IN ClNameT               *pNodeName,
            CL_IN ClCpmLcmReplyT        *pSrcInfo);
```

**Parameters:**

***pSuName:*** Pointer to the name of the SU being instantiated.

***pNodeName:*** Pointer to the name of the node where pSuName exists.

***pSrcInfo:*** Pointer to the source information, where the reply of the requested operation is expected. If no response is required, pass this as NULL.

**Return values:**

***CL_OK:*** The function executed successfully.

***CL_ERR_NULL_POINTER:*** `pSuName`, `pNodeName`, or `pScrInfo` contains a NULL pointer.

**Description:**
This function is used to instantiate the given SU. This is equivalent to instantiating all the components in a pre-defined order for a given SU.

**Note:** This is an asynchronous function.

**Library Files:**
ClAmfClient

**Related API(s):**
clCpmSUTerminate

### 3.3.28 clCpmSUTerminate

**clCpmSUTerminate**

**Synopsis:**
Terminates a given Service Unit (SU).

**Header File:**
clCpmApi.h

**Syntax:**

```
extern ClRcT clCpmSUTerminate(
            CL_IN ClNameT             *pSuName,
            CL_IN ClNameT             *pNodeName,
            CL_IN ClCpmLcmReplyT      *pSrcInfo);
```

**Parameters:**

***pSuName:*** Pointer to the name of the SU being terminated.

***pNodeName:*** Pointer to the name of the node where `pSuName` exist.

***pSrcInfo:*** Pointer to the source information, where the reply of the requested operation is expected. If no response is required, pass this as NULL.

**Return values:**

***CL_OK:*** The function executed successfully.

***CL_ERR_NULL_POINTER:*** `pSuName`, `pNodeName`, or `pScrInfo` contains a NULL pointer.

**Description:**
This function is used to terminate a given SU. This is quivalent to terminating all the components in a pre-defined order for a given SU.

**Note:** This is an asynchronous function.

**Library Files:**
ClAmfClient

**Related API(s):**
clCpmSUInstantiate

### 3.3.29 clCpmMasterAddressGet

**clCpmMasterAddressGet**

**Synopsis:**
Returns the IOC address of the master.

**Header File:**
clCpmApi.h

**Syntax:**

```
        extern ClRcT clCpmMasterAddressGet(
                    CL_OUT ClIocNodeAddressT *pIocAddress);
```

**Parameters:**
***pIocAddress:*** (out) IOC address of the master.

**Return values:**
***CL_OK:*** The function executed successfully.

***CL_ERR_NULL_POINTER:*** `pIocAddress` contains a NULL pointer.

**Description:**
This function is used to retrieve the IOC address of the master. The address returned by this function is the physical address of the node where the CPM/G is running.

**Library Files:**
ClAmfClient

**Related API(s):**
clCpmIsMaster

### 3.3.30 clCpmIsMaster

**clCpmIsMaster**

**Synopsis:**
Informs if the node is the master of the cluster.

**Header File:**
clCpmApi.h

**Syntax:**
```
        extern ClUint32T clCpmIsMaster();
```

**Parameters:**
None

**Return values:**
***CL_TRUE:*** The node is the master.

***CL_FALSE:*** The node is not the master.

**Description:**
This function indicates if the node is master of the cluster. This function can be used by the component to determine if it is running on the CPM master (CPM/G (global) or CPM/L (local)).

**Library Files:**
ClAmfClient

**Related API(s):**
clCpmMasterAddressGet

### 3.3.31 clCpmNodeShutDown

**clCpmNodeShutDown**

**Synopsis:**
Shuts down the node.

**Header File:**
clCpmApi.h

**Syntax:**

```
extern ClRcT clCpmNodeShutDown(
              CL_IN ClIocNodeAddressT iocNodeAddress);
```

**Parameters:**
***iocNodeAddress:*** IOC address of the node.

**Return values:**
***CL_OK:*** The function executed successfully.

**Description:**
This function is used to shut down the node, corresponding to the given IOC Address. This function terminates all services running on that node.

**Note:** This is an asynchronous function.

**Library Files:**
ClAmfClient

**Related API(s):**
None.

### 3.3.32 clCpmLocalNodeNameGet

**clCpmSlotInfoGet**

**Synopsis:**
Returns the name of the local node.

**Header File:**
clCpmApi.h

**Syntax:**

```
extern ClRcT clCpmLocalNodeNameGet(
                   CL_IN ClNameT *nodeName);
```

**Parameters:**
***nodeName:*** Local node name.

**Return values:**
***CL_OK:*** The function executed successfully.

***CL_ERR_NULL_POINTER:*** `nodeName` contains a NULL pointer.

***CL_ERR_DOESNT_EXIST:*** CPM library is not able to retrieve the `nodeName`.

**Description:**
This function provides the local node name, in the buffer provided by the user. This function can be used by a component to determine the name of the node on which it is running.

**Library Files:**
ClAmfClient

**Related API(s):**
None.

### 3.3.33 clCpmComponentMoIdGet

**clCpmComponentMoIdGet**

**Synopsis:**
Retrieves the MoID of the component.

**Header File:**
clCpmExtApi.h

**Syntax:**
```
extern ClRcT clCpmComponentMoIdGet(
                   CL_IN ClNameT compName
                   CL_OUT ClCorMOIdT *compMoId);
```

**Parameters:**
***nodeName:*** Local node name.

**Return values:**
***CL_OK:*** The function executed successfully.

**Description:**
This function provides the `MoId` of the user component. `MoId` of the component is a unique identifier that represents the position of the component in the tree heirarchy maintained by the COR.

**Library Files:**
ClAmfClient

**Related API(s):**
None.

### 3.3.34 clCpmEventPayLoadExtract

**clCpmEventPayLoadExtract**

**Synopsis:**
Extracts the event payload data.

**Header File:**
clCpmApi.h

**Syntax:**

```
extern clCpmEventPayLoadExtract(
            CL_IN   ClEventHandleT eventHandle,
            CL_IN   ClSizeT eventDataSize,
            CL_IN   ClCpmEventTypeT cpmEventType,
            CL_OUT  void *payLoad)
```

**Parameters:**

***eventHandle:*** Event handle.

***eventDataSize:*** Size of the event data.

***cpmEventType:*** Type of the event published.

***payLoad:*** (out) The actual payload.

**Return values:**

***CL_OK:*** The function executed successfully.

***CL_ERR_NULL_POINTER:*** `payLoad` contains a NULL pointer.

**Description:**
This function extracts the payload data from the flat buffer returned by the event delivery callback for the events published by the CPM. `eventHandle` and `eventDataSize` are the obtained from the event delivery callback. This function must be called only in the event subscriber's context of event delivery callback, that is invoked when the event is published by CPM.

**Library Files:**
ClAmfClient

**Related API(s):**
None.

### 3.3.35 clCpmSlotInfoGet

**clCpmSlotInfoGet**

**Synopsis:**

Returns the Slot related information such as node moID, nodeName, and locAddress. Provides the mapping between slot ID, IOC address, MoID, and name of the given node depending on the flag that is passed to it.

**Header File:**

clCpmApi.h

**Syntax:**

```
extern ClRcT clCpmSlotInfoGet(
                   CL_IN ClCpmSlotInfoFieldIdT flag,
                   CL_OUT ClCpmSlotInfoT *slotInfo);
```

**Parameters:**

***flag:*** Flag that indicates the field of the structure,`slotInfo` that is populated by the user.

***slotInfo:*** The structure, populated by this function, will contain other information about node corresponding to the field provided by the user.

**Return values:**

***CL_OK:*** The function executed successfully.

***CL_ERR_DOESNT_EXIST:*** `flag` is invalid or a parameter is not set correctly.

**Description:**

This function provides the mapping between the attributes that are unique to the node namely, slot ID, IOC address, MOID, and node name. Depending on the flag provided by the user, this function assumes that the corresponding field has been given a correct value by the user and populates the remaining fields of the structure, corresponding to the information in this field. For example, if the user needs to know the MOID of a given slot ID, the flag `CL_CPM_SLOT_ID` should be provided (which are exposed in `clCpmApi.h`) and the `slotId` field of `slotInfo` structure should be set to a valid slot number.

**Library Files:**

libClAmfClient

**Related APIs:**

None.

### 3.3.36 clCpmIocAddressForNodeGet

**clCpmIocAddressForNodeGet**

**Synopsis:**
Retrieves the IOC address for the given node.

**Header File:**
clCpmApi.h

**Syntax:**

```
extern ClRcT clCpmIocAddressForNodeGet(
                    CL_IN ClNameT nodeName,
                    CL_OUT ClIocAddressT *pIocAddress);
```

**Parameters:**

***nodeName:*** The name of the node whose `IocAddress` is required.

***pIocAddress:*** IOC address of a given node, `nodeName`.

**Return values:**

***CL_OK:*** The function executed successfully.

***CL_ERR_DOESNT_EXIST:*** CPM library is unable to retrieve the `nodeName`.

**Description:**
This function returns the physical IOC address of the node, which is passed as an argument.

**Library Files:**
libClAmfClient

**Related APIs:**
None.

### 3.3.37 clCpmIsCompRestarted

**clCpmIsCompRestarted**

**Synopsis:**
Checks if the given component has been restarted.

**Header File:**
clCpmExtApi.h

**Syntax:**
```
    extern ClBoolT clCpmIsCompRestarted(
                        CL_IN ClNameT compName);
```

**Parameters:**
***compName*** : The name of the component.

**Return values:**
***CL_OK:*** The function executed successfully.

***CL_ERR_DOESNT_EXIST:*** CPM library is unable to retrieve the name of the component.

**Description:**
This function returns TRUE, if the component has been restarted before.

**Library Files:**
libClAmfClient

**Related APIs:**
None.

# Chapter 4

# Service Management Information Model

TBD

# Chapter 5

# Service Notifications

TBD

# Chapter 6

# Debug CLIs

TBD

# Index