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

For OpenClovis SDK Release 2.3 V0.4

Document Revision Date: March 20, 2007

Contents

Chapter 1

Functional Overview

The OpenClovis Fault Manager (Fault Manager) infrastructure provides a hierarchical scheme for managing faults in a system and initiating actions as configured during the design time. It can handle various user-defined run-time faults, including hardware and software faults, and can prioritize faults to ensure that critical faults are addressed before the normal (low priority) faults.

Alarms are notified by the Fault Manager client library to the Fault Manager server located on the same node. The actions to be taken on receiving a fault are controlled by the Fault Manager policy associated with the faults. The Fault Manager interacts with the Availability Management Framework (AMF) while handling faults. The AMF performs recovery action and the Fault Manager performs the fault repairs. Both AMF and Fault Manager entities are interdependent while taking actions in a system during recovery and repair.

In a high availability enabled system, central fault management is taken by the AMF framework whereas, in a high availability disabled system, Fault Manager is the primary decision-maker for performing both recovery and repair actions.

The fault management service provides the following services:

- Report a fault
- Repair a fault

Before a component can use the `clFaultReport()` or `clFaultRepairAction()` functions, it needs to:

1. Check the version compatibility - This can be performed using the `clFaultVersionVerify()` function. If the call returns `CL_OK`, the component is compatible with the fault client library.
2. Initialize the fault client library - This can be performed using the `clFaultSvcLibInitialize()` function. After the library is successfully initialized, the fault report/repair functionality can be used. When the fault service is not required, the fault library must be finalized.

Chapter 2

Service Model

TBD

Chapter 3

Service APIs

3.1 Type Definitions

3.1.1 ClAlarmStateT

typedef struct {
CL_ALARM_STATE_CLEAR = 0,
CL_ALARM_STATE_ASSERT = 1
} ClAlarmStateT;

The *ClAlarmStateT* contains the list of enumeration values for the state of the Alarm.

- *CL_ALARM_STATE_CLEAR* - Indicates Alarm condition has cleared.
- *CL_ALARM_STATE_ASSERT* - Indicates Alarm condition has occurred.

3.1.2 ClAlarmCategoryTypeT

typedef struct {
CL_ALARM_CATEGORY_INVALID = 0,
CL_ALARM_CATEGORY_COMMUNICATIONS = 1,
CL_ALARM_CATEGORY_QUALITY_OF_SERVICE = 2,
CL_ALARM_CATEGORY_PROCESSING_ERROR = 3,
CL_ALARM_CATEGORY_EQUIPMENT = 4,
CL_ALARM_CATEGORY_ENVIRONMENTAL = 5
} ClAlarmCategoryTypeT;

The structure, `ClAlarmCategoryTypeT` contains the various categories for Alarms.

- *CL_ALARM_CATEGORY_COMMUNICATIONS* - Category for Alarms that are related to communication.
- *CL_ALARM_CATEGORY_QUALITY_OF_SERVICE* - Category for Alarms that are related to quality of service.
- *CL_ALARM_CATEGORY_PROCESSING_ERROR* - Category for Alarms that are related to processing error.

- *CL_ALARM_CATEGORY_EQUIPMENT* - Category for Alarms that are related to equipment.
- *CL_ALARM_CATEGORY_ENVIRONMENTAL* - Category for Alarms that are related to environmental conditions.

3.1.3 ClAlarmSpecificProblemT

typedef ClUint32T ClAlarmSpecificProblemT;

The type of an identifier to the specific problem of the Alarm. This information is not configured but is assigned a value at run-time for segregation of alarms that have the same category and probable cause, but are different in their manifestation.

3.1.4 ClAlarmSeverityTypeT

typedef struct {
CL_ALARM_SEVERITY_INVALID=0,
CL_ALARM_SEVERITY_CRITICAL=1,
CL_ALARM_SEVERITY_MAJOR=2,
CL_ALARM_SEVERITY_MINOR=3,
CL_ALARM_SEVERITY_WARNING=4,
CL_Alarm_SEVERITY_INDETERMINATE=5,
CL_ALARM_SEVERITY_CLEAR=6
} ClAlarmSeverityTypeT;

The structure, *ClAlarmSeverityTypeT*, contains the various severity levels of an Alarm.

3.1.5 ClAlarmProbableCauseT

typedef enum {
CL_ALARM_PROB_CAUSE_LOSS_OF_SIGNAL,
CL_ALARM_PROB_CAUSE_LOSS_OF_FRAME,
CL_ALARM_PROB_CAUSE_FRAMING_ERROR,
CL_ALARM_PROB_CAUSE_LOCAL_NODE_TRANSMISSION_ERROR,
CL_ALARM_PROB_CAUSE_REMOTE_NODE_TRANSMISSION_ERROR,
CL_ALARM_PROB_CAUSE_CALL_ESTABLISHMENT_ERROR,
CL_ALARM_PROB_CAUSE_DEGRADED_SIGNAL,
CL_ALARM_PROB_CAUSE_COMMUNICATIONS_SUBSYSTEM_FAILURE,
CL_ALARM_PROB_CAUSE_COMMUNICATIONS_PROTOCOL_ERROR,
CL_ALARM_PROB_CAUSE_LAN_ERROR,
CL_ALARM_PROB_CAUSE_DTE,
CL_ALARM_PROB_CAUSE_RESPONSE_TIME_EXCESSIVE,
CL_ALARM_PROB_CAUSE_QUEUE_SIZE_EXCEEDED,
CL_ALARM_PROB_CAUSE_BANDWIDTH_REDUCED,
CL_ALARM_PROB_CAUSE_RETRANSMISSION_RATE_EXCESSIVE,
CL_ALARM_PROB_CAUSE_THRESHOLD_CROSSED,
CL_ALARM_PROB_CAUSE_PERFORMANCE_DEGRADED,
CL_ALARM_PROB_CAUSE_CONGESTION,
CL_ALARM_PROB_CAUSE_RESOURCE_AT_OR_NEARING_CAPACITY,

```
        CL_ALARM_PROB_CAUSE_STORAGE_CAPACITY_PROBLEM,
        CL_ALARM_PROB_CAUSE_VERSION_MISMATCH,
        CL_ALARM_PROB_CAUSE_CORRUPT_DATA,
        CL_ALARM_PROB_CAUSE_CPU_CYCLES_LIMIT_EXCEEDED,
        CL_ALARM_PROB_CAUSE_SOFWARE_ERROR,
        CL_ALARM_PROB_CAUSE_SOFTWARE_PROGRAM_ERROR,
        CL_ALARM_PROB_CAUSE_SOFWARE_PROGRAM_ABNORMALLY_TERMINATED,
        CL_ALARM_PROB_CAUSE_FILE_ERROR,
        CL_ALARM_PROB_CAUSE_OUT_OF_MEMORY,
        CL_ALARM_PROB_CAUSE_UNDERLYING_RESOURCE_UNAVAILABLE,
        CL_ALARM_PROB_CAUSE_APPLICATION_SUBSYSTEM_FAILURE,
        CL_ALARM_PROB_CAUSE_CONFIGURATION_OR_CUSTOMIZATION_ERROR,
        CL_ALARM_PROB_CAUSE_POWER_PROBLEM,
        CL_ALARM_PROB_CAUSE_TIMING_PROBLEM,
        CL_ALARM_PROB_CAUSE_PROCESSOR_PROBLEM,
        CL_ALARM_PROB_CAUSE_DATASET_OR_MODEM_ERROR,
        CL_ALARM_PROB_CAUSE_MULTIPLEXER_PROBLEM,
        CL_ALARM_PROB_CAUSE_RECEIVER_FAILURE,
        CL_ALARM_PROB_CAUSE_TRANSMITTER_FAILURE,
        CL_ALARM_PROB_CAUSE_RECEIVE_FAILURE,
        CL_ALARM_PROB_CAUSE_TRANSMIT_FAILURE,
        CL_ALARM_PROB_CAUSE_OUTPUT_DEVICE_ERROR,
        CL_ALARM_PROB_CAUSE_INPUT_DEVICE_ERROR,
        CL_ALARM_PROB_CAUSE_INPUT_OUTPUT_DEVICE_ERROR,
        CL_ALARM_PROB_CAUSE_EQUIPMENT_MALFUNCTION,
        CL_ALARM_PROB_CAUSE_ADAPTER_ERROR,
        CL_ALARM_PROB_CAUSE_TEMPERATURE_UNACCEPTABLE,
        CL_ALARM_PROB_CAUSE_HUMIDITY_UNACCEPTABLE,
        CL_ALARM_PROB_CAUSE_HEATING_OR_VENTILATION_OR_COOLING_SYSTEM_PROBLEM,
        CL_ALARM_PROB_CAUSE_FIRE_DETECTED,
        CL_ALARM_PROB_CAUSE_FLOOD_DETECTED,
        CL_ALARM_PROB_CAUSE_TOXIC_LEAK_DETECTED,
        CL_ALARM_PROB_CAUSE_LEAK_DETECTED,
        CL_ALARM_PROB_CAUSE_PRESSURE_UNACCEPTABLE,
        CL_ALARM_PROB_CAUSE_EXCESSIVE_VIBRATION,
        CL_ALARM_PROB_CAUSE_MATERIAL_SUPPLY_EXHAUSTED,
        CL_ALARM_PROB_CAUSE_PUMP_FAILURE,
        CL_ALARM_PROB_CAUSE_ENCLOSURE_DOOR_OPEN
} ClAlarmProbableCauseT;
```

The enumeration, `ClAlarmProbableCauseT`, indicates the possible causes of alarms. The values of this enumeration are:

Following are the probable causes for communication related alarms:

- *CL_ALARM_PROB_CAUSE_LOSS_OF_SIGNAL*
- *CL_ALARM_PROB_CAUSE_LOSS_OF_FRAME*
- *CL_ALARM_PROB_CAUSE_FRAMING_ERROR*
- *CL_ALARM_PROB_CAUSE_LOCAL_NODE_TRANSMISSION_ERROR*
- *CL_ALARM_PROB_CAUSE_REMOTE_NODE_TRANSMISSION_ERROR*
- *CL_ALARM_PROB_CAUSE_CALL_ESTABLISHMENT_ERROR*

- *CL_ALARM_PROB_CAUSE_DEGRADED_SIGNAL*
- *CL_ALARM_PROB_CAUSE_COMMUNICATIONS_SUBSYSTEM_FAILURE*
- *CL_ALARM_PROB_CAUSE_COMMUNICATIONS_PROTOCOL_ERROR*
- *CL_ALARM_PROB_CAUSE_LAN_ERROR*
- *CL_ALARM_PROB_CAUSE_DTE*

Following are the probable causes for quality of service related alarms:

- *CL_ALARM_PROB_CAUSE_RESPONSE_TIME_EXCESSIVE*
- *CL_ALARM_PROB_CAUSE_QUEUE_SIZE_EXCEEDED*
- *CL_ALARM_PROB_CAUSE_BANDWIDTH_REDUCED*
- *CL_ALARM_PROB_CAUSE_RETRANSMISSION_RATE_EXCESSIVE*
- *CL_ALARM_PROB_CAUSE_THRESHOLD_CROSSED*
- *CL_ALARM_PROB_CAUSE_PERFORMANCE_DEGRADED*
- *CL_ALARM_PROB_CAUSE_CONGESTION*
- *CL_ALARM_PROB_CAUSE_RESOURCE_AT_OR_NEARING_CAPACITY*

Following are the probable causes for equipment related alarms:

- *CL_ALARM_PROB_CAUSE_POWER_PROBLEM*
- *CL_ALARM_PROB_CAUSE_TIMING_PROBLEM*
- *CL_ALARM_PROB_CAUSE_PROCESSOR_PROBLEM*
- *CL_ALARM_PROB_CAUSE_DATASET_OR_MODEM_ERROR*
- *CL_ALARM_PROB_CAUSE_MULTIPLEXER_PROBLEM*
- *CL_ALARM_PROB_CAUSE_RECEIVER_FAILURE*
- *CL_ALARM_PROB_CAUSE_TRANSMITTER_FAILURE*
- *CL_ALARM_PROB_CAUSE_RECEIVE_FAILURE*
- *CL_ALARM_PROB_CAUSE_TRANSMIT_FAILURE*
- *CL_ALARM_PROB_CAUSE_OUTPUT_DEVICE_ERROR*
- *CL_ALARM_PROB_CAUSE_INPUT_DEVICE_ERROR*
- *CL_ALARM_PROB_CAUSE_INPUT_OUTPUT_DEVICE_ERROR*
- *CL_ALARM_PROB_CAUSE_EQUIPMENT_MALFUNCTION*
- *CL_ALARM_PROB_CAUSE_ADAPTER_ERROR*

Following are the probable causes for environmental related alarms:

- *CL_ALARM_PROB_CAUSE_TEMPERATURE_UNACCEPTABLE*

- *CL_ALARM_PROB_CAUSE_HUMIDITY_UNACCEPTABLE*
- *CL_ALARM_PROB_CAUSE_HEATING_OR_VENTILATION_OR_COOLING_SYSTEM_-PROBLEM*
- *CL_ALARM_PROB_CAUSE_FIRE_DETECTED*
- *CL_ALARM_PROB_CAUSE_FLOOD_DETECTED*
- *CL_ALARM_PROB_CAUSE_TOXIC_LEAK_DETECTED*
- *CL_ALARM_PROB_CAUSE_LEAK_DETECTED*
- *CL_ALARM_PROB_CAUSE_PRESSURE_UNACCEPTABLE*
- *CL_ALARM_PROB_CAUSE_EXCESSIVE_VIBRATION*
- *CL_ALARM_PROB_CAUSE_MATERIAL_SUPPLY_EXHAUSTED*
- *CL_ALARM_PROB_CAUSE_PUMP_FAILURE*
- *CL_ALARM_PROB_CAUSE_ENCLOSURE_DOOR_OPEN*

3.1.6 ClAlarmHandleT

typedef ClUint32T ClAlarmHandleT;

Name of the alarm event channel. This is the channel on which the subscriber waits for notifications.

3.2 Library Life Cycle APIs

3.2.1 clFaultSvcLibInitialize

clFaultSvcLibInitialize

Synopsis:
Initializes the Fault Manager library.

Header File:
clFaultApi.h

Syntax:

```
ClRcT clFaultSvcLibInitialize(void);
```

Parameters:
None

Return values:
CL_OK: The function executed successfully.

CL_FAULT_ERR_CLIENT_INIT_FAILED: The fault service failed to initialize.

Description:
This function is used to initialize the Fault Manager library. It initializes the fault client and starts fault related services for a particular component. It registers various callbacks and allocates resources.

Library File(s):
ClFaultClient

Related Function(s):
clFaultSvcLibFinalize.

3.2.2 clFaultSvcLibFinalize

clFaultSvcLibFinalize

Synopsis:
Cleans up the Fault Manager library and frees resources allocated to it.

Header File:
clFaultApi.h

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

Parameters:
None

Return values:
CL_OK: The function executed successfully.

CL_FAULT_ERR_CLIENT_FINALIZE_FAILED: The fault service failed to finalize.

Description:
This function is used to clean up Fault Manager library linked to a particular component. This must be called, if the services related to the component are not required. To avoid memory leaks, every call to the `clFaultSvcLibInitialize()` function must be followed by a call to the `clFaultSvcLibFinalize()` function.

Library File:
ClFaultClient

Related Function(s):
clFaultSvcLibInitialize.

3.3 Functional APIs

3.3.1 clFaultReport

clFaultReport

Synopsis:
Reports a fault to the Fault service.

Header File:
clFaultApi.h

Syntax:

```
ClRcT clFaultReport(
                          CL_IN ClNameT *compName,
                          CL_IN ClCorMOIdPtrT hMoId,
                          CL_IN ClAlarmStateT alarmState,
                          CL_IN ClAlarmCategoryTypeT category,
                          CL_IN ClAlarmSpecificProblemT specificProblem,
                          CL_IN ClAlarmSeverityTypeT severity,
                          CL_IN ClAlarmProbableCauseT cause,
                          CL_IN void *pData,
                          CL_IN ClUint32T len);
```

Parameters:

compName: (in) Name of the component on which the fault occurred.

hMoId: (in) Handler of the `MoId` of the object on which the fault occurred.

alarmState : (in) State of the alarm. It can be in the `assert` or `clear` state.

category: (in) Category of the fault.

specificProblem: (in) Specific problem of the fault. The specific problem is used to segregate the duplicate probable causes belonging to a single probable cause list, but are different with respect to the application. This information lies with the user-application, and is interpreted by it.

severity: (in) Severity of the fault.

cause: (in) Probable cause of the fault.

pData: (in) Additional information about the fault. Additional information can contain messages about the fault which the application notifies to Fault Manager.

len: (in) Length of `pData`.

Return values:

CL_OK: The function executed successfully.

CL_FAULT_ERR_MOID_NULL: `moid`, passed to the fault service, is NULL.

CL_FAULT_ERR_COMPNAME_NULL: Component name, passed to fault service, is NULL.

CL_FAULT_ERR_INVALID_CATEGORY: Fault belongs to an invalid category. These categories are mentioned in ITUX.733.

CL_FAULT_ERR_INVALID_SEVERITY: The severity of this fault is invalid. These severity levels are mentioned in ITUX.733.

CL_ERR_NO_MEMORY: Memory allocation failure.

Description:

This function is used by Alarm Manager, Chassis Manager, and Component Manager to report faults to the fault service. The fault service can be provided by the AMF or by Fault Manager. If AMF is present, the fault is processed by AMF to provide service recovery and repair. In the absence of AMF, the fault is sent to the Fault Manager. The Fault Manager executes the fault repair handler configured by the user for a specific fault type.

Library File:

ClFaultClient

Related Function(s):

None.

3.3.2 clFaultRepairAction

clFaultRepairAction

Synopsis:
Notifies fault to Fault service to execute the repair action.

Header File:
clFaultApi.h

Syntax:

```
ClRcT clFaultRepairAction(
                          CL_IN ClIocAddressT iocAddress,
                          CL_IN ClAlarmHandleT alarmHandle,
                          CL_IN ClUint32T recoveryActionTaken);
```

Parameters:

iocAddress: (in) IOC address of the node on which the fault occurred.

alarmHandle: (in) Handle to the alarm that is used to retrieve the information about the alarm and execute the repair handler for the corresponding `moClass` type.

recoveryActionTaken: (in) Recovery action taken by AMF.

Return values:

CL_OK: The function executed successfully.

CL_ERR_NO_MEMORY: Memory allocation failure.

Description:
This function is used by AMF to report the faults to the fault service. The Fault Manager executes the fault repair handler configured for the specific fault type. The repair handlers are configured as per the `moClass` type.

Related Function(s):
None.

3.3.3 clFaultVersionVerify

clFaultVersionVerify

Synopsis:
Verifies the version of the supported Fault Manager library.

Header File:
clFaultApi.h

Syntax:

```
ClRcT clFaultVersionVerify(
                        ClVersionT *version  );
```

Parameters:
version (in/out): As an input parameter, this contains the client version. In the output parameter, the supported version is returned by this function.

Return values:
CL_OK: The function executed successfully.

CL_FAULT_ERR_VERSION_UNSUPPORTED: The client version is not supported.

Description:
This function is used to verify if a version of the Fault Manager library is supported. If the version is not supported, an error is returned with the output parameter containing the supported version.

Library File:
ClFaultClient

Related Function(s):
None.

Chapter 4

Service Management Information Model

TBD

Chapter 5

Service Notifications

TBD

Chapter 6

Configuration

TBD

Chapter 7

Debug CLIs

TBD

Index