PM4351, PM4354

COMET AND COMET-QUAD DEVICE DRIVER

DRIVER MANUAL

PROPRIETARY AND CONFIDENTIAL

RELEASE

ISSUE 2: JUNE, 2001

OVERVIEW

Scope

This document is the user's driver manual for the COMET (PM4351) and COMET-QUAD (PM4354) device driver software. It describes the features and functionality provided by the driver, the software architecture, and the external interfaces of the driver software. Differences in API functionality for COMET and COMET-QUAD devices are listed throughout the API descriptions on a per-function basis.

Objectives

The main objectives of this document are as follows:

- Provide a detailed list of the device driver's features
- Describe the software architecture of the driver (data structures, state diagrams, function descriptions).
- Describe the external interfaces of the driver. The external interfaces illustrate how the driver interacts with the underlying hardware and RTOS as well as external application software

References

The main references for this document are as follows:

- PMC-1990315 – Four Channel Combined E1/T1/J1 Transceiver/Framer Data Sheet, Issue 6
- PMC-1970624 – Combined E1/T1 Transceiver Standard Product Data Sheet, Issue 10

Legal Issues

PMC-2010108 (P1), ref PMC-1991407 (P2)

Contacting PMC-Sierra

PMC-Sierra, Inc.
105-8555 Baxter Place Burnaby, BC
Canada V5A 4V7

Tel: (604) 415-6000
Fax: (604) 415-6200

Document Information: document@pmc-sierra.com
Corporate Information: info@pmc-sierra.com
Technical Support: apps@pmc-sierra.com
Web Site: http://www.pmc-sierra.com

TABLE OF CONTENTS

LIST OF FIGURES

LIST OF TABLES

1 INTRODUCTION

The following sections of the COMET and COMET-QUAD driver manual describe the COMET and COMET-QUAD device driver. Throughout the document, the device driver will be referred to as the COMET-QUAD device driver. However, the driver supports the COMET through the same set of API calls. Sections 4 and 5 of this document outline how each API function within the driver applies to both COMET and COMET-QUAD devices on a per API function basis.

The code provided throughout this document is written in the C language. This has been done to promote greater driver portability to other embedded Real Time Operating Systems (section 6) and hardware environments (section 5).

Section 2 of this document, Software Architecture, defines the software architecture of the COMET-QUAD device driver by including a discussion of the driver's external interfaces and its main components. The Data Structure information in section 3 describes the elements of the driver that configure or control its behavior. Included here are the constants, variables and structures that the COMET-QUAD Device Driver uses to store initialization, configuration and statistics information. Section 4 provides a detailed description of each function that is a member of the COMET-QUAD driver Application Programming Interface (API). The section outlines function calls that hide device-specific details and application callbacks that notify the user of significant device events.

For your convenience, section 7 of this manual provides a brief guide for porting the COMET-QUAD device driver to your hardware and RTOS platform. In addition, an extensive Appendix (page 152) and Index (page 171) provide you with useful reference information.

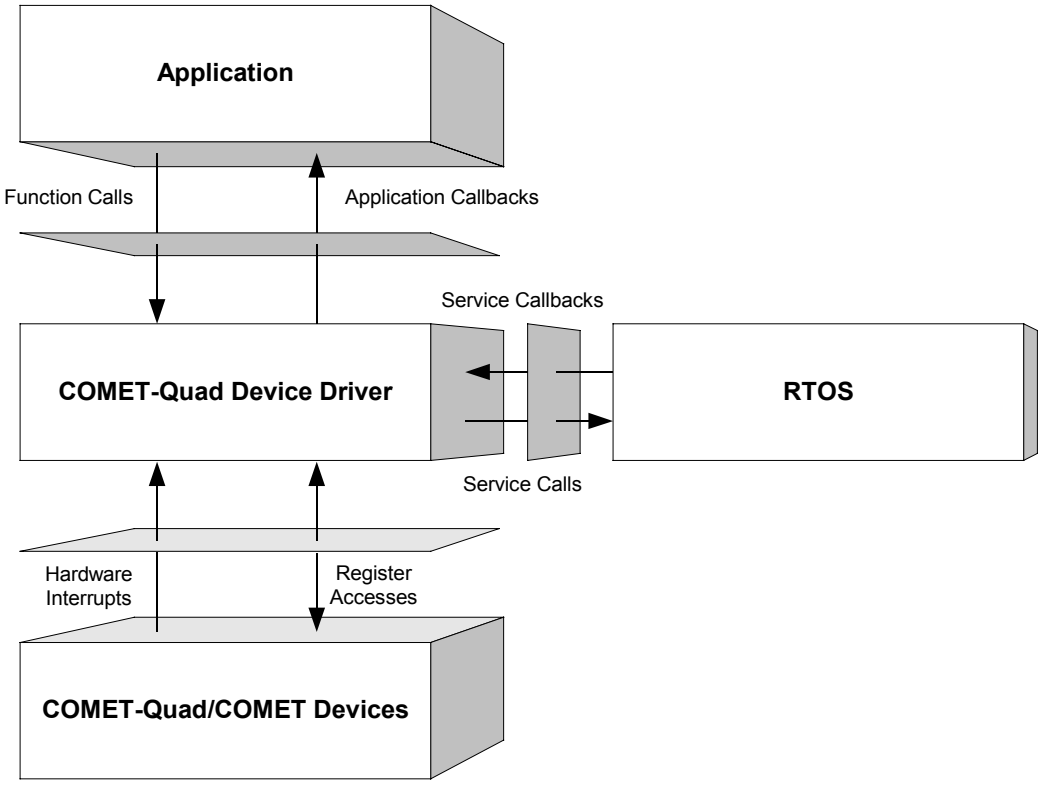
2 SOFTWARE ARCHITECTURE

This section of the manual describes the software architecture of the COMET-QUAD device driver. Details of the software architecture include a discussion of the driver's external interfaces and its main components.

2.1 Driver External Interfaces

Figure 1 illustrates the external interfaces defined for the COMET-QUAD device driver.

Figure 1: Driver External Interfaces



Application Programming Interface

The Driver Application Programming Interface (API) is a list of high-level functions that can be invoked by application programmers to configure, control and monitor COMET-QUAD and COMET devices.

The API includes the following functions:

- Initialize the device(s)
- Perform diagnostic tests
- Validate configuration information
- Retrieve status and statistics information

The driver API functions use the services of the other driver components to provide this system-level functionality to the application programmer.

The driver API also consists of callback routines that are used to notify the application of significant events that take place within the device(s) and module.

Real-Time Operating System (RTOS) Interface

The driver's RTOS interface provides functions that let the driver use the RTOS's memory, interrupt, and pre-emption services. These RTOS interface functions perform the following tasks for the driver:

- Allocate and de-allocate memory
- Manage buffers for the ISR and the DPR
- Enable and disable pre-emption

The RTOS interface also includes service callbacks. These are functions installed by the driver using RTOS service calls such as installing interrupts. These service callbacks are invoked when an interrupt occurs.

Hardware Interface

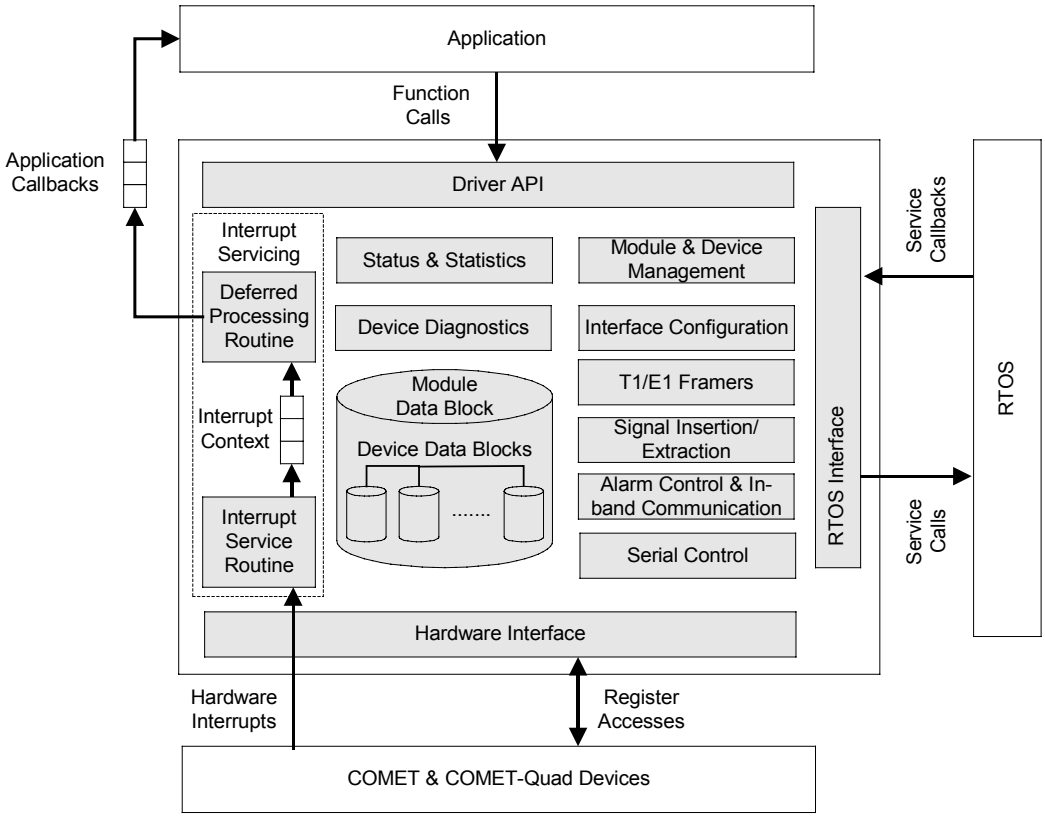
The hardware interface provides functions that read from and write to the device registers. The hardware interface also provides a template for an ISR that the driver calls when the device raises a hardware interrupt. You must modify this function based on the interrupt configuration of your system.

2.2 Main Components

Figure 2 illustrates the top-level architectural components of the COMET-QUAD device driver:

- Module data-block and device data-blocks
- Module and device management
- Interrupt servicing
- Status and statistics
- Interface configuration
- T1/E1 framers
- Signal Insertion/Extraction
- Alarm Control and Inband Communication
- Serial Control
- Device Diagnostics and Loopbacks

Figure 2: Driver Architecture



Module Data-Block and Device Data-Blocks

The Module Data-Block (MDB) is the top layer data structure created by the COMET-QUAD driver to store context information about the driver module, such as:

- Module state
- Maximum number of devices
- The DDB(s)
- The initialization profile(s)

The Device Data-Block (DDB) is contained in the MDB, and initialized by the driver module for each COMET-QUAD or COMET device that is registered. There is one DDB per device and there is a limit on the number of DDBs available. That limit is set by the USER when the module is initialized. The DDB is used to store context information about one device, such as:

- Device state
- Control information
- Initialization parameters

- Callback function pointers

Module and Device Management

The module and device management block provides the following:

- Module management that takes care of initializing the driver, allocating memory and all RTOS resources needed by the driver
- Device management that is responsible for providing basic read/write routines and initializing a device in a specific configuration, as well as enabling the device general activity

For more information on the module and device states see the state diagram on page 21. For typical module and device management flow diagrams see pages 23 and 24 respectively.

Interrupt Servicing/Polling

Interrupt Servicing is an optional feature. The user can disable device interrupts and instead poll the device periodically to monitor status and check for alarm/error conditions. The COMET-QUAD driver provides:

- An Interrupt-Service Routine (ISR) called `cometqISR` that checks for valid interrupt conditions
- A Deferred Processing Routine (DPR) called `cometqDPR` that processes any interrupt condition gathered by the ISR

See section 2.5 for a detailed explanation of the interrupt-servicing model.

Status and Statistics Collection

This section contains functions to gather statistics and monitor the status of the device. Specifically, this interface provides functionality to examine error counts such as framing errors, CRC errors, and line code violations as well as clock status information and pseudo-random generator error counts.

This API allows the user to monitor the status of each framer on the device. The device is polled for the following error conditions:

- Loss of signal
- Loss of frame alignment
- Alarm Indication Signal (AIS)
- Yellow or E1 RAI alarm
- E1 loss of signaling multiframe alignment
- E1 loss of CRC-4 multiframe alignment

- E1 timeslot 16 RAI

Also, transmission of T1 ESF FDL automatic performance report messages (APRM) is handled through this API. The user can perform the following APRM functions through the status and statistics collection API:

- Enable/disable automatic performance report insertion into the FDL. For COMET devices, manual insertion of the performance report can be forced by these APIs
- View the current APRM message to be sent on the FDL

Interface Configuration

This section of the driver allows the user to configure both the T1/E1 line and the receive and transmit backplane interfaces. Specifically, these functions provide the following:

- Selection of the receive and transmit line coding scheme: B8ZS, HDB3, or AMI
- Programming of transmit pulse waveform from predefined Long/Short haul waveform template or allow USER defined
- Transmitter fuse programming
- Selection of Clock & Data Recovery algorithm for low or high frequency jitter tolerance
- Receive and transmit jitter attenuation enable/disable and configuration
- Analog and digital loss of signal (LOS) configuration
- Backplane clock master or slave mode selection for the transmit and receive interfaces
- Backplane data format (channelized, full frame) and the clock rate configuration
- Frame pulse master or slave and frame pulse offset configuration
- Receive and transmit backplane H-MVIP or H-MVIP CCS mode selection

T1/E1 Framers

This section of the driver contains functions that are used to configure the T1/E1 receive and transmit framers. These functions provide the following:

- Selection between global T1 or E1 mode
- Configuration of the T1 framers to transmit or receive in SF, ESF, T1DM, SLC96 or TTC JT-G704 mode
- Configure the E1 framers to transmit or receive in CRC-4 multiframe format as well as enable channel associative signaling insertion and receiver signaling multiframe alignment
- Retrieve and transmit E1 international, national, and extra bits.

In addition to configuring the relevant framer, these functions configure the complete receive and transmit data path based on the receive and transmit framing modes specified.

Signal Insertion / Extraction

The signal insertion/extraction API provide functionality that monitors channel associative signaling (CAS) in the receive stream. These API provide the following functionality:

- Extraction of CAS from an E1 signaling multiframe or from a T1 ESF or SF frame
- Monitoring signaling state transitions
- Control on a per-timeslot basis, bit fixing, data inversion, and signal state debouncing

Alarm Control and Inband Communications

This section of the driver contains functions that are used for detection and insertion of Inband Communication codes, BOC codes, and alarms. Specifically, these functions perform the following:

- Transmission of AIS, yellow alarms, E1 y-bit alarms, and E1 timeslot 16 AIS
- Enabling or disabling of automatic device alarm response upon detection of yellow alarm, red alarm, out-of frame, or loss of signal
- Detection and transmission of BOC codes transmitted in the T1 FDL channel in ESF framing format
- Configuration of transmit and receive HDLC data links
- Configuration of T1 FDL to transmit one of 63 possible BOC codes
- Transmission and detection of T1 Inband Loopback codes

Serial Controller

This section contains functions that are used to configure the receive and transmit serial controllers. Specifically, these functions perform the following:

- DS0 control of PCM data manipulation and digital milliwatt pattern insertion
- Pseudo-random pattern generation and detection
- Idle code insertion into T1 or E1 stream

Device Diagnostics

The device diagnostics API can be used to isolate/identify problems within the device and its interfaces. Specifically, these functions perform:

- Device register read/write test
- Backplane loopback
- Line loopback
- Payload loopback

- Per-channel loopback
- Analog transmitter and receiver bypass

Specific Callback Functions

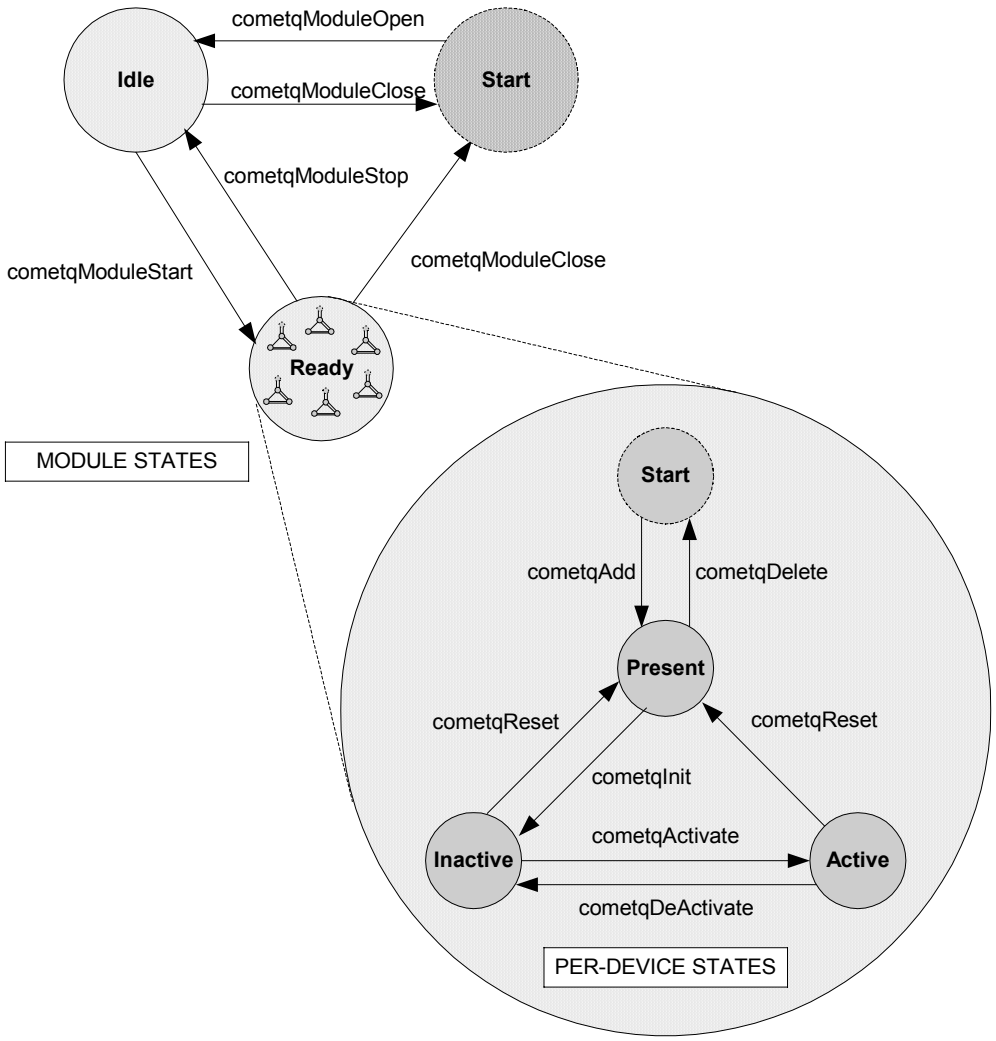
Callback functions are available to the application for event notification from the device driver. Applications will be notified via the callback functions for selected events of interest such as:

- Alarm conditions
- Statistics
- Diagnostics
- Line coding and conditioning
- T1/E1 Framer
- Data Link
- Performance Monitoring

2.3 Software States

Figure 3 shows the software state diagram for the COMET-QUAD driver. State transitions occur on the successful execution of the corresponding transition functions shown below. State information helps maintain the integrity of the MDB and DDB(s) by controlling the set of operations allowed in each state.

Figure 3: Driver Software States



Module States

The following is a description of the COMET-QUAD module states. See section 4.1 for a detailed description of the API functions that are used to change the module state. The module states are:

Start

The driver module has not been initialized. In this state the driver does not hold any RTOS resources (memory, timers, etc), has no running tasks, and performs no actions.

Idle

The driver module has been initialized successfully. The Module Initialization Vector (MIV) has been validated; the Module Data Block (MDB) has been allocated and loaded with current data; the per-device data structures have been allocated; and the RTOS has responded without error to all the requests sent to it by the driver.

Ready

This is the normal operating state for the driver module. This means that all RTOS resources have been allocated and the driver is ready for devices to be added. The driver module remains in this state while devices are in operation.

Device States

The following is a description of the COMET or COMET-QUAD per-device states. The state that is mentioned here is the software state as maintained by the driver, and not as maintained inside the device itself. See section 4.1 for a detailed description of the API functions that are used to change the per-device state.

Start

The device has not been initialized. In this state the device is unknown to the driver and performs no actions. There is a separate flow for each device that can be added, and they all start here.

Present

The device has been successfully added. A Device Data Block (DDB) has been associated with the device and updated with the user context; and a device handle has been given to the USER. In this state, the device performs no actions.

Inactive

In this state the device is configured but all data functions are de-activated, including interrupts and alarms, and status and statistics functions.

Active

This is the normal operating state for the device. In this state, interrupt servicing or polling is enabled.

2.4 Processing Flows

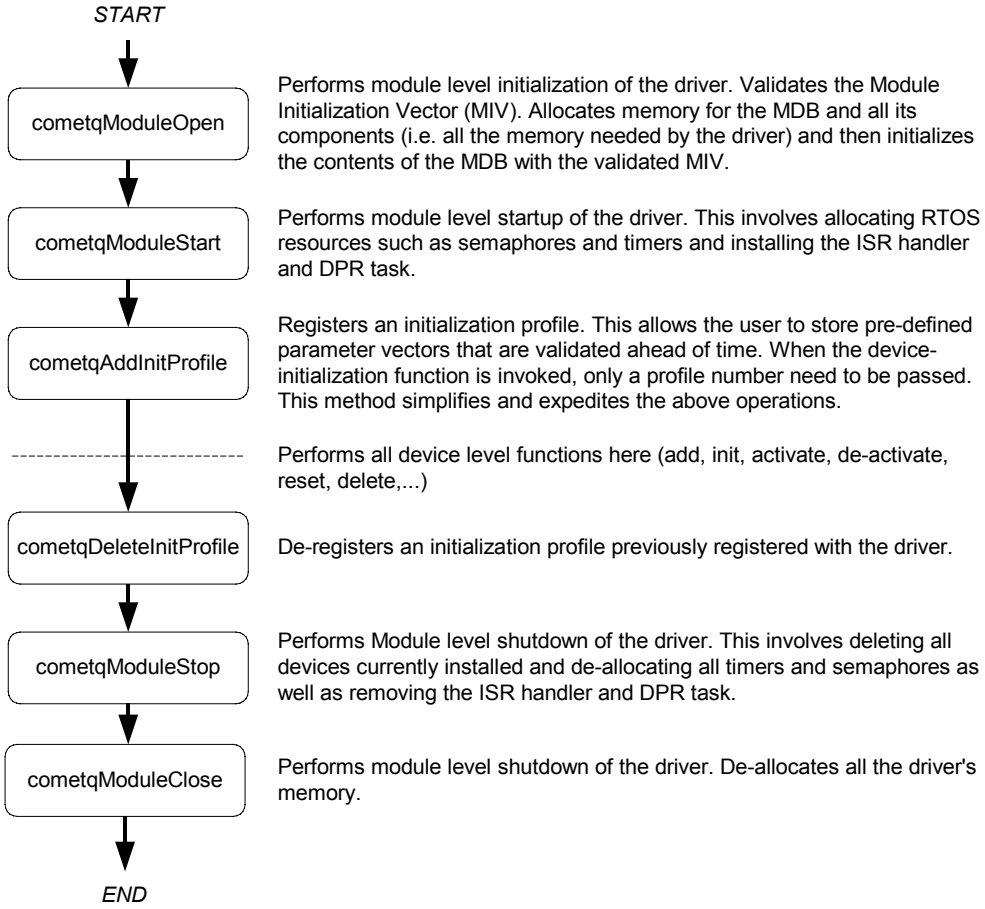
This section of the manual describes the main processing flows of the COMET-QUAD driver components.

The flow diagrams presented here illustrate the sequence of operations that take place for different driver functions. The diagrams also serve as a guide to the application programmer by illustrating the sequence in which the application must invoke the driver API.

Module Management

The following diagram illustrates the typical function call sequences that occur when either initializing or shutting down the COMET-QUAD driver module.

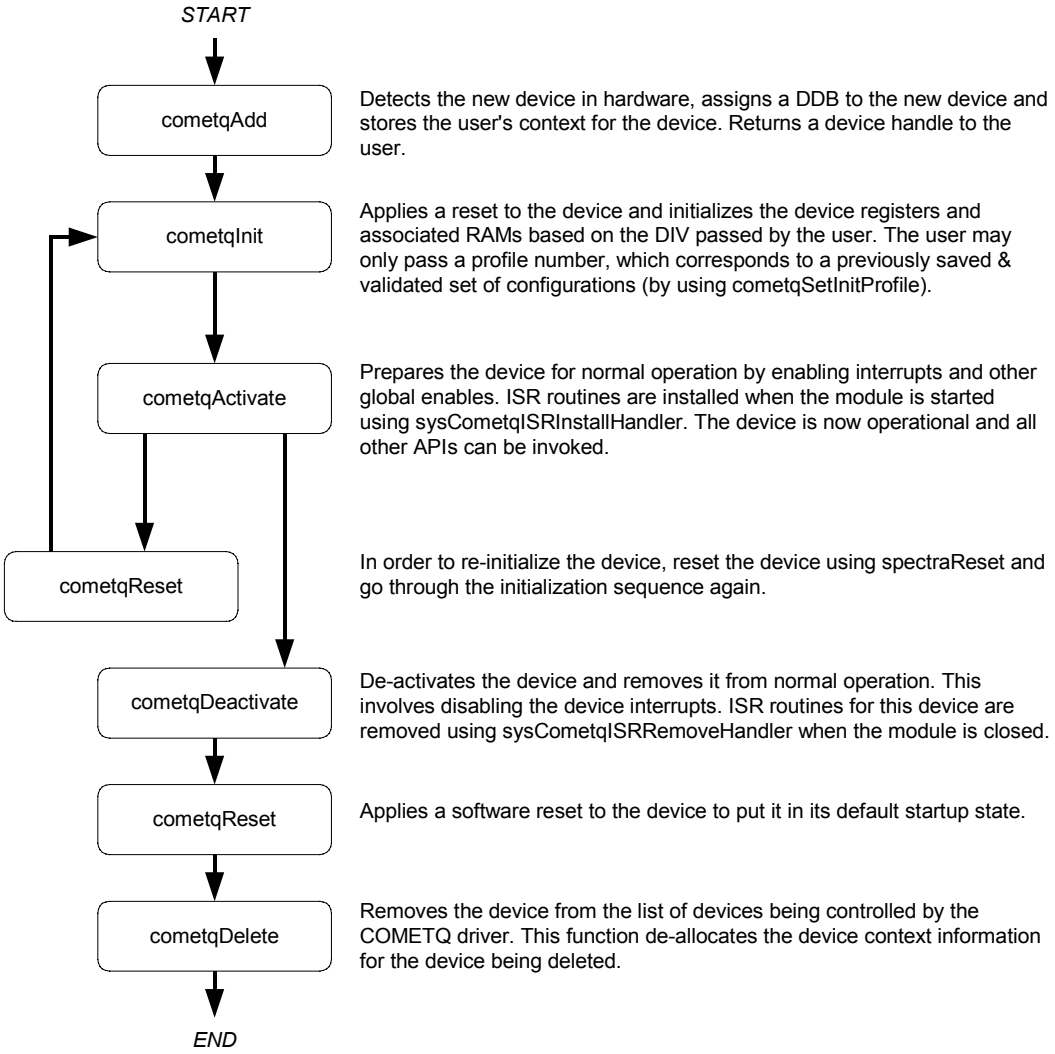
Figure 4: Module Management Flow Diagram



Device Management

The following figure shows the typical function call sequences that the driver uses to add, initialize, re-initialize, and delete a COMET or COMET-QUAD device.

Figure 5: Device Management Flow Diagram



2.5 Interrupt Servicing

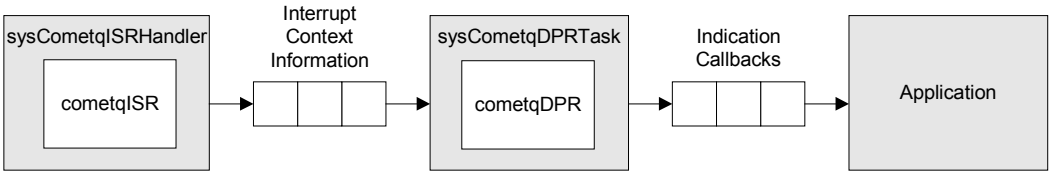
The COMET-QUAD driver services device interrupts using an interrupt service routine (ISR) that traps interrupts. It also contains a deferred processing routine (DPR) that actually processes the interrupt conditions and clears them. This architecture enables the ISR to execute quickly and then exit. Most of the time-consuming processing of the interrupt conditions is deferred to the DPR by queuing the necessary interrupt-context information to the DPR task. The DPR function runs in the context of a separate task within the RTOS.

Note: Since the DPR task processes potentially serious interrupt conditions, you should set the DPR task's priority higher than the application task interacting with the COMET-QUAD driver.

The driver provides system-independent functions, `cometqISR` and `cometqDPR`. You must fill in the corresponding system-specific functions, `sysCometqISRHandler` and `sysCometqDPRTask`. The system-specific functions isolate the system-specific communication mechanism (between the ISR and DPR) from the system-independent functions, `cometqISR` and `cometqDPR`.

Figure 6 illustrates the interrupt service model used in the COMET-QUAD driver design.

Figure 6: Interrupt Service Model



Note: Instead of using an interrupt service model, you can use a polling service model in the COMET-QUAD driver to process the device's event-indication registers (see page 26).

Calling cometqISR

An interrupt handler function (which is system dependent), must call `cometqISR`. But first, the low-level interrupt-handler function must trap the device interrupts. You must implement this function (`sysCometqISRHandler`) to fit your own system. For an example implementation of the interrupt handler and its prototype, see page 137.

The interrupt handler that you implement (`sysCometqISRHandler`) is installed in the interrupt vector table of the system processor. It is called when one or more COMET or COMET-QUAD devices interrupt the processor. The interrupt handler then calls `cometqISR` for each device in the active state that has interrupt processing enabled.

The cometqISR function reads from the master interrupt-status registers and the miscellaneous interrupt-status registers of the COMET or COMET-QUAD. If at least one valid interrupt condition is found then cometqISR fills an Interrupt Service Vector (ISV) with this status information as well as the current device handle. The cometqISR function also clears and disables all the interrupts detected by the device. The sysCometqISRHandler function is then responsible for sending this ISV buffer to the DPR task.

Note: Normally you should save the status information for deferred processing by implementing a message queue. The interrupt handler sends the status information to the queue by sysCometqISRHandler.

Calling cometqDPR

The sysCometqDPRTask function is a system specific function that runs as a separate task within the RTOS. You should set the DPR task's priority higher than the application task(s) interacting with the COMET-QUAD driver. In the message-queue implementation model, this task has an associated message queue. The task waits for messages from the ISR on this message queue. When a message arrives, sysCometqDPRTask calls the DPR (cometqDPR) with the received ISV.

Then cometqDPR processes the status information and takes appropriate action based on the specific interrupt condition detected. The nature of this processing can differ from system to system. Therefore, cometqDPR calls different indication callbacks for different interrupt conditions.

Typically, you should implement these callback functions as simple message posting functions that post messages to an application task. However, you can implement the indication callback to perform processing within the DPR task context and return without sending any messages. In this case, ensure that this callback function does not call any API functions that would change the driver's state, such as cometqDelete. In addition, ensure that the callback function is non-blocking because the DPR task executes while COMET or COMET-QUAD interrupts are disabled. You can customize these callbacks to suit your system. See page 131 for example implementations of the callback functions.

Note: Since the cometqISR and cometqDPR routines themselves do not specify a communication mechanism, you have full flexibility in choosing a communication mechanism between the two. A convenient way to implement this is to use a message queue, which is a service that most RTOSs provide.

You must implement the two system specific functions, sysCometqISRHandler and sysCometqDPRTask. When the driver calls sysCometqISRHandlerInstall, the application installs sysCometqISRHandler in the interrupt vector table of the processor, and the sysCometqDPRTask function is spawned as a task by the application. The sysCometqISRHandlerInstall function also creates the communication channel between sysCometqISRHandler and sysCometqDPRTask. This communication channel is most commonly a message queue associated with sysCometqDPRTask.

Similarly, during removal of interrupts, the driver removes `sysCometqISRHandler` from the microprocessor's interrupt vector table and deletes the task associated with `sysCometqDPRTask`.

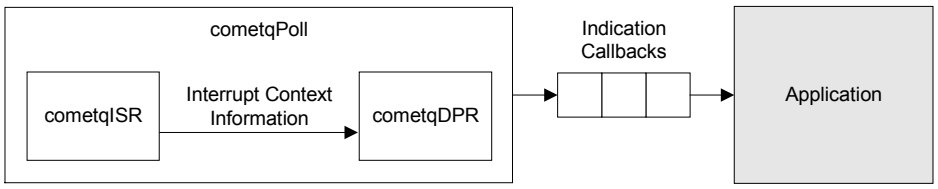
As a reference, the driver provides example implementations of the interrupt installation and removal functions on pages 137 and 152. You can modify these functions to suit your specific needs.

Calling cometqPoll

Instead of using an interrupt service model, you can use a polling service model in the COMET-QUAD driver to process the device's event-indication registers.

Figure 7 illustrates the polling service model used in the COMET-QUAD driver design.

Figure 7: Polling Service Model



In polling mode, the application is responsible for calling `cometqPoll` often enough to service any pending error or alarm conditions. When `cometqPoll` is called, the `cometqISR` function is called internally.

The `cometqISR` function reads from the master interrupt-status registers and the miscellaneous interrupt-status registers of the COMET or COMET-QUAD device. If at least one valid interrupt condition is found then `cometqISR` fills an Interrupt Service Vector (ISV) with this status information as well as the current device handle. The `cometqISR` function also clears and disables all the device's interrupts detected. In polling mode, `cometqPoll` then invokes `cometqDPR` directly and passes the ISV buffer (returned by `cometqISR`) as an input parameter.

3 DATA STRUCTURES

This section of the manual describes the elements of the driver that configure or control its behavior. Included here are the constants, variables and structures that the COMET-QUAD Device Driver uses to store initialization, configuration and statistics information. For more information on naming conventions, please see Appendix A (page 152).

3.1 Constants

The following Constants are used throughout the driver code:

- `<COMET-QUAD ERROR CODES>`: this contains error codes returned by the API functions and used in the global error number field of the MDB and DDB. For a complete list of error codes see Appendix B.
- `CMQ_MAX_DEVS`: this defines the maximum number of devices that can be supported by this driver. This constant must not be changed without a thorough analysis of the consequences to the driver code.
- `CMQ_MAX_INIT_PROFS`: this defines the maximum number of profiles that can be supported by this driver.
- `CMQ_MOD_START`, `CMQ_MOD_IDLE`, `CMQ_MOD_READY`: these contain the three possible Module states (stored in the MDB as `stateModule`).
- `CMQ_START`, `CMQ_PRESENT`, `CMQ_ACTIVE`, `CMQ_INACTIVE`: these contain the four possible Device states (stored in the DDB as `stateDevice`).

3.2 Structures Passed by the Application

These structures are defined for use by the application and are passed as an argument to functions within the driver. These structures are the Module Initialization Vector (MIV), the Device Initialization Vector (DIV) and the ISR mask. The following explains their workings in detail.

Module Initialization Vector: MIV

Passed via the `cometqModuleOpen` call, this structure contains all the information needed by the driver to initialize and connect to the RTOS.

- The variable `maxDevs` is used to inform the driver how many devices will be operating concurrently during this session. The number is used to calculate the amount of memory that will be allocated to the driver. The maximum value that can be passed is `CMQ_MAX_DEVS` (see section 3.1).

- The variable `maxInitProfs` is used to inform the driver how many profiles will be used during this session. The number is used to calculate the amount of memory that will be allocated to the driver. The maximum value that can be passed is `CMQ_MAX_INIT_PROFS` (see section 3.1).

Table 1: COMET-QUAD Module Initialization Vector: `sCMQ_MIV`

Field Name	Field Type	Field Description
`perrModule`	`INT4 *`	(pointer to) `errModule` (see description in the MDB)
`maxDevs`	`UINT2`	Maximum number of devices supported during this session
`maxInitProfs`	`UINT2`	Maximum number of initialization profiles

Device Initialization Vector: DIV

Passed via the `cometqInit` call, this structure contains all the information needed by the driver to initialize a COMET or COMET-QUAD device. This structure is also passed via the `cometqSetInitProfile` call when used as an initialization profile.

Note that when initializing COMET-QUAD devices with a DIV, the hardware initialization specified in the DIV is applied to all four quadrants.

- `valid` indicates that this initialization profile has been properly initialized and may be used by the USER. This field should be ignored when the DIV is passed directly.
- `pollISR` is a flag that indicates the type of interrupt servicing the driver is to use. The choices are 'polling' (`CMQ_POLL_MODE`), and 'interrupt driven' (`CMQ_ISR_MODE`). When configured in polling the Interrupt capability of the device is NOT used, and the USER is responsible for calling `cometqPoll` periodically. The actual processing of the event information is the same for both modes.
- `cbackFramer, cbackIntf, cbackAlarmInBand, cbackPMon, cbackSerialCtl,` and `cbackSigInsExt` are used to pass the address of application functions that will be used by the DPR to inform the application code of pending events. If these fields are set as NULL, then any events that might cause the DPR to 'call back' the application will be processed during ISR processing but ignored by the DPR.
- `initDevice` is a flag that indicates whether or not the hardware is to be initialized based on the information in the `analogInit`, `framerInit`, and `backplaneInit` members of the DIV. If `initDevice` is not set, the hardware remains in its reset state upon initialization. When set, the device is initialized as appropriate for the values of the hardware initialization members. This field is only used when the DIV is not being used as an initialization profile. When adding an initialization profile, the hardware configuration must always be valid.

Table 2: COMET-QUAD Device Initialization Vector: `sCMQ_DIV`

Field Name	Field Type	Field Description
valid	UINT2	Indicates that this structure is valid
pollISR	eCMQ_ISR_MODE	Indicates type of interrupt processing (ISR mode or polling)
cbackIntf	CMQ_CBACK	Address of the callback function for Interface Events
cbackFramer	CMQ_CBACK	Address of the callback function for Framer Events
cbackAlarmInBand	CMQ_CBACK	Address of the callback function for Alarm Inband Events
cbackSigInsExt	CMQ_CBACK	Address of the callback function for Signal Insertion and Extraction Events
cbackPMon	CMQ_CBACK	Address of the callback function for Performance Monitoring Events
cbackSerialCtl	CMQ_CBACK	Address of the callback function for Serial Control Events
initDevice	UINT1	Flag to indicate whether or not to apply hardware configuration (analogInit, framerInit, backplaneInit) to the device. If this flag is not set, the device is left in its reset state. By setting this flag, analogInit, framerInit, and backplaneInit are all applied to the device. This field is not used when the DIV is used as an initialization profile.
analogInit	sCMQ_ANALOG_INIT	Initialization configuration for the analog interfaces.
framerInit	sCMQ_FRAMER_INIT	Initialization configuration for the transmit and receive framers.
backplaneInit	sCMQ_BACKPLANE_INIT	Initialization configuration for the transmit and receive backplane interfaces.

DIV Sub-structures

The following structures are members of the device initialization vector (DIV) structure. They have the function of initializing the analog transmitter and receiver, the transmit and receive framers, and the transmit and receive backplane interfaces respectively. The contents of these structures are not applied to the device hardware when the `initDevice` member of the DIV is set to false. When this member is true, the hardware configuration is applied to the device.

Initialization of the framers includes configuring all blocks within the device to conform to the framing modes that are given in the framer initialization structure. Backplane hardware is programmed as recommended in the Operations section of the COMET and COMET-QUAD data sheet.

The transmit and receive analog structure allows the user to select the transmit pulse waveform and the receive equalizer RAM from a table stored within the driver. These tables correspond to the values specified in the Operations section of the COMET and COMET-QUAD data sheet.

Table 3: COMET-QUAD Analog Transmitter and Receiver Initialization: `sCMQ_ANALOG_INIT`

Field Name	Field Type	Field Description

`txLineBuildOut`	`eCMQ_TX_LBO`	Selects XLPG line build out and waveform scale factor. Choose from predefined tables and corresponding waveform scale factors stored within the driver (as defined in the COMET and COMET-QUAD data sheet):
		
		`CMQ_TX_LBO_T1_LONG_HAUL_0DB,`
		`CMQ_TX_LBO_T1_LONG_HAUL_7_5DB,`
		`CMQ_TX_LBO_T1_LONG_HAUL_15DB,`
		`CMQ_TX_LBO_T1_LONG_HAUL_22_5DB,`
		`CMQ_TX_LBO_T1_LONG_HAUL_TR62411_0DB,`
		`CMQ_TX_LBO_T1_SHORT_HAUL_110FT,`
		`CMQ_TX_LBO_T1_SHORT_HAUL_220FT,`
		`CMQ_TX_LBO_T1_SHORT_HAUL_330FT,`
		`CMQ_TX_LBO_T1_SHORT_HAUL_440FT,`
		`CMQ_TX_LBO_T1_SHORT_HAUL_550FT,`
		`CMQ_TX_LBO_T1_SHORT_HAUL_660FT,`
		`CMQ_TX_LBO_T1_SHORT_HAUL_TR62411_110FT,`
		`CMQ_TX_LBO_T1_SHORT_HAUL_TR62411_220FT,`
		`CMQ_TX_LBO_T1_SHORT_HAUL_TR62411_330FT,`
		`CMQ_TX_LBO_T1_SHORT_HAUL_TR62411_440FT,`
		`CMQ_TX_LBO_T1_SHORT_HAUL_TR62411_550FT,`
		`CMQ_TX_LBO_T1_SHORT_HAUL_TR62411_660FT,`
		`CMQ_TX_LBO_E1_75OHM,`
		`CMQ_TX_LBO_E1_120OHM`

`txEnable`	`UINT1`	Set to 1 to enable transmitter, otherwise analog outputs are left in high impedance state

Field Name	Field Type	Field Description
rxEqualizerTable	eCMQ_RX_LINE_EQ	Selects one of predefined RLPS equalizer RAMs (as defined in the COMET and COMET-QUAD data sheet): CMQ_RX_LINE_EQ_RAM_T1, CMQ_RX_LINE_EQ_RAM_E1
csuClkMode	eCMQ_CSU_SVC_CLK	Selects the clock synthesis unit (CSU) operational mode based on the XCLK frequency: CMQ_XCLK_2048_TXCLK_2048, CMQ_XCLK_1544_TXCLK_1544, CMQ_XCLK_2048_TXCLK_1544

Table 4: COMET-QUAD Transmit and Receive Framer Initialization: sCMQ_FRAMER_INIT

Field Name	Field Type	Field Description
txFramerMode	eCMQ_FRAME_MODE	Selects a T1 or E1 framing format for the transmit framer: CMQ_FRM_MODE_E1, CMQ_FRM_MODE_E1_CRC_MFRM, CMQ_FRM_MODE_E1_UNFRAMED, CMQ_FRM_MODE_T1_SF, CMQ_FRM_MODE_T1_DM, CMQ_FRM_MODE_T1_SLC96, CMQ_FRM_MODE_T1_DM_FDL, CMQ_FRM_MODE_T1_ESF, CMQ_FRM_MODE_T1_SF_JPN_ALARM, CMQ_FRM_MODE_T1_DM_JPN_ALARM, CMQ_FRM_MODE_T1_SLC96_JPN_ALARM, CMQ_FRM_MODE_T1_DM_FDL_JPN_ALARM, CMQ_FRM_MODE_T1_JT_G704, CMQ_FRM_MODE_T1_UNFRAMED

Field Name	Field Type	Field Description
rxFramerMode	eCMQ_FRAME_MODE	Selects a T1 or E1 framing format for the receive framer: CMQ_FRM_MODE_E1, CMQ_FRM_MODE_E1_CRC_MFRM, CMQ_FRM_MODE_E1_UNFRAMED, CMQ_FRM_MODE_T1_SF, CMQ_FRM_MODE_T1_DM, CMQ_FRM_MODE_T1_SLC96, CMQ_FRM_MODE_T1_DM_FDL, CMQ_FRM_MODE_T1_ESF, CMQ_FRM_MODE_T1_SF_JPN_ALARM, CMQ_FRM_MODE_T1_DM_JPN_ALARM, CMQ_FRM_MODE_T1_SLC96_JPN_ALARM, CMQ_FRM_MODE_T1_DM_FDL_JPN_ALARM, CMQ_FRM_MODE_T1_JT_G704, CMQ_FRM_MODE_T1_UNFRAMED Note that both the transmit and receive framers must be operating in either T1 or E1.

Table 5: COMET-QUAD Transmit and Receive Backplane Interface Initialization: sCMQ_BACKPLANE_INIT

Field Name	Field Type	Field Description
backplaneTxMode	eCMQ_BACKPLANE_TX_MODE	Selects the configuration of the transmit backplane interface: CMQ_BACKPLANE_TX_CLOCK_MASTER_FULL_T1E1, CMQ_BACKPLANE_TX_CLOCK_MASTER_Nx64, CMQ_BACKPLANE_TX_CLOCK_MASTER_CLEAR_CHAN CMQ_BACKPLANE_TX_CLOCK_SLAVE_FULL_T1E1, CMQ_BACKPLANE_TX_CLOCK_SLAVE_CLEAR_CHAN, CMQ_BACKPLANE_TX_CLOCK_SLAVE_HMVIP (COMET-QUAD only), CMQ_BACKPLANE_TX_CLOCK_SLAVE_FULL_T1E1_HMVIP_CCS (COMET-QUAD only) For a detailed description of each of these configurations, consult the device data sheet.

Field Name	Field Type	Field Description
backplaneRxMode	eCMQ_BACKPLANE_RX_MODE	Selects the configuration of the receive backplane interface: CMQ_BACKPLANE_RX_CLOCK_MASTER_FULL_T1E1, CMQ_BACKPLANE_RX_CLOCK_MASTER_Nx64, CMQ_BACKPLANE_RX_CLOCK_MASTER_CLEAR_CHAN CMQ_BACKPLANE_RX_CLOCK_SLAVE_FULL_T1E1, CMQ_BACKPLANE_RX_CLOCK_SLAVE_HMVIP (COMET-QUAD Only) CMQ_BACKPLANE_RX_CLOCK_SLAVE_FULL_T1E1_HMVIP_CCS (COMET-QUAD only) For a detailed description of each of these configurations, please consult the device data sheet.
CCSTimeslot15	UINT1	If backplaneTxMode is CMQ_BACKPLANE_TX_CLOCK_SLAVE_FULL_T1E1_HMVIP_CCS, this member enables CCS insertion into timeslot 15
CCSTimeslot16	UINT1	If backplaneTxMode is CMQ_BACKPLANE_TX_CLOCK_SLAVE_FULL_T1E1_HMVIP_CCS, this member enables CCS insertion into timeslot 16
CCSTimeslot31	UINT1	If backplaneTxMode is CMQ_BACKPLANE_TX_CLOCK_SLAVE_FULL_T1E1_HMVIP_CCS, this member enables CCS insertion into timeslot 31

ISR Enable/Disable Mask

Passed via the cometqSetMask, cometqGetMask and cometqClrMask calls, this structure contains all the information needed by the driver to enable and disable any of the interrupts on the COMET or COMET-QUAD.

ISR Mask Top-Level Structure

Table 6: COMET-QUAD ISR Mask: ***sCMQ_ISR_MASK***

Field Name	Field Type	Field Description
cdrc	sCMQ_ISR_MASK_CDRC[4]	Clock & data recovery mask

Field Name	Field Type	Field Description
rjat	sCMQ_ISR_MASK_JAT[4]	Receive jitter attenuation mask
tjat	sCMQ_ISR_MASK_JAT[4]	Transmit jitter attenuation mask
rxSlip	sCMQ_ISR_MASK_SLIP[4]	Receive elastic store slip mask
txSlip	sCMQ_ISR_MASK_SLIP[4]	Transmit elastic store slip mask
btif	sCMQ_ISR_MASK_BTIF[4]	Transmit backplane interface signal and data parity detection mask
t1Frmr	sCMQ_ISR_MASK_T1FRMR[4]	T1 receive framer mask
ibcd	sCMQ_ISR_MASK_IBCD[4]	Inband communications loopback mask
pmon	sCMQ_ISR_MASK_PMON[4]	Performance monitoring mask
almi	sCMQ_ISR_MASK_ALMI[4]	Alarm integrator mask
pdvd	sCMQ_ISR_MASK_PDVD[4]	Receive pulse density violation mask
xpde	sCMQ_ISR_MASK_XPDE[4]	Transmit pulse density enforcer mask
rboc	sCMQ_ISR_MASK_RBOC[4]	Receive BOC mask
tboc	UINT1[4]	Transmit BOC interrupt enable (COMET-QUAD Only)
e1Tran	sCMQ_ISR_MASK_E1TRAN[4]	E1 transmit section mask
e1Frmr	sCMQ_ISR_MASK_E1FRMR[4]	E1 receive framer mask
rxCCSSlip	sCMQ_ISR_MASK_SLIP[4]	Receive signal elastic store slip mask (COMET-QUAD Only)
txCCSSlip	sCMQ_ISR_MASK_SLIP[4]	Transmit signal elastic store slip mask (COMET-QUAD Only)
tdpr	sCMQ_ISR_MASK_TDPR[4]	HDLC transmitter mask
rdlc	sCMQ_ISR_MASK_RDLC[4]	HDLC receiver mask
prbs	sCMQ_ISR_MASK_PRBS[4]	Pseudo random binary sequence mask
rlps	sCMQ_ISR_MASK_RLPS[4]	RLPS analog loss of signal mask

Field Name	Field Type	Field Description
aprmEn	UINT1[4]	Auto performance report mask
sigx	sCMQ_ISR_MASK_SIGX[4]	Signaling extractor mask

ISR Mask Sub-structures

Table 7: COMET-QUAD ISR SubMask: sCMQ_ISR_MASK_CDRC

Field Name	Field Type	Field Description
lcvEn	UINT1	CDRC line code violation detection enable
losEn	UINT1	CDRC loss of signal detection enable
lcsdEn	UINT1	CDRC line code signature detection enable
zndEn	UINT1	CDRC excess zero detection enable
altLosEn	UINT1	CDRC alternate loss of signal detection enable
atlLosInd	UINT1	CDRC alternate loss of signal status indicator
losInd	UINT1	CDRC loss of signal status indicator

Table 8: COMET-QUAD ISR SubMask: sCMQ_ISR_MASK_RLPS

Field Name	Field Type	Field Description
losEn	UINT1	RLPS loss of signal detection enable
losInd	UINT1	RLPS loss of signal status indicator

Table 9: COMET-QUAD ISR SubMask: sCMQ_ISR_MASK_JAT

Field Name	Field Type	Field Description
undEn	UINT1	Jitter attenuator FIFO underrun detection enable
ovrEn	UINT1	Jitter attenuator FIFO overrun detection enable

Table 10: COMET-QUAD ISR SubMask: `sCMQ_ISR_MASK_SLIP`

Field Name	Field Type	Field Description
slipEn	UINT1	Elastic store FIFO underrun/overrun detection enable
slipInd	UINT1	Elastic store FIFO underrun/overrun indicator (1 = overrun, 0 = underrun)

Table 11: COMET-QUAD ISR SubMask: `sCMQ_ISR_MASK_T1FRMR`

Field Name	Field Type	Field Description
COFAEn	UINT1	Change of frame alignment detection enable
ferEn	UINT1	Framing bit error detection enable
beeEn	UINT1	Bit error event detection enable
sefEn	UINT1	Severely errored frame detection enable
mfpEn	UINT1	Mimic frame pattern detection enable
mfpInd	UINT1	Mimic frame pattern status indicator
infrEn	UINT1	In frame detection enable
infrInd	UINT1	In frame status indicator

Table 12: COMET-QUAD ISR SubMask: `sCMQ_ISR_MASK_IBCD`

Field Name	Field Type	Field Description
lbaEn	UINT1	Activate loopback code detection enable
lbaInd	UINT1	Activate loopback code status indicator
lbdEn	UINT1	Deactivate loopback code detection enable
lbdInd	UINT1	Deactivate loopback code status indicator

Table 13: COMET-QUAD ISR SubMask: `sCMQ_ISR_MASK_ALMI`

Field Name	Field Type	Field Description
yelEn	UINT1	Yellow alarm detection enable

Field Name	Field Type	Field Description
yelInd	UINT1	Yellow alarm status indicator
redEn	UINT1	Red alarm detection enable
redInd	UINT1	Red alarm status indicator
AISEn	UINT1	AIS detection enable
AISInd	UINT1	AIS status indicator

Table 14: COMET-QUAD ISR SubMask: sCMQ_ISR_MASK_PDVD

Field Name	Field Type	Field Description
z16dEn	UINT1	16 consecutive zero detection enable
pdvdEn	UINT1	Pulse density violation detection enable
pdvdInd	UINT1	Pulse density violation status indicator

Table 15: COMET-QUAD ISR SubMask: sCMQ_ISR_MASK_XPDE

Field Name	Field Type	Field Description
stufEn	UINT1	Bit stuff detection enable
z16dEn	UINT1	16 consecutive zero detection enable
pdvEn	UINT1	Pulse density enforcer violation detection enable
pdvInd	UINT1	Pulse density enforcer violation status indicator

Table 16: COMET-QUAD ISR SubMask: sCMQ_ISR_MASK_RBOC

Field Name	Field Type	Field Description
idleEn	UINT1	Idle code detection enable
BOCEn	UINT1	BOC code detection enable

Table 17: COMET-QUAD ISR SubMask: sCMQ_ISR_MASK_E1TRAN

Field Name	Field Type	Field Description
sigmfEn	UINT1	Signaling multiframe boundary detection enable
nfasEn	UINT1	NFAS frame boundary detection enable
mfEn	UINT1	CRC-4 multiframe boundary detection enable
smfEn	UINT1	Signaling multiframe boundary detection enable
frmEn	UINT1	Frame boundary detection enable

Table 18: COMET-QUAD ISR SubMask: sCMQ_ISR_MASK_E1FRMR

Field Name	Field Type	Field Description
c2nciwEn	UINT1	CRC to non-CRC internetworking detection enable
c2nciwInd	UINT1	CRC to non-CRC internetworking status indicator
OOFEn	UINT1	Out of frame detection enable
OOFInd	UINT1	Out of frame status indicator
oosmfEn	UINT1	Out of signaling multiframe detection enable
oosmfInd	UINT1	Out of signaling multiframe status indicator
oocmfEn	UINT1	Out of CRC-4 multiframe detection enable
oocmfInd	UINT1	Out of CRC-4 multiframe status indicator
COFAEn	UINT1	Change of frame alignment detection enable
ferEn	UINT1	Frame error detection enable
smferEn	UINT1	Signaling multiframe error detection enable
cmferEn	UINT1	CRC-4 multiframe error detection enable
RAIEn	UINT1	Remote alarm indication detection enable
RAIInd	UINT1	Remote alarm indication status indicator

Field Name	Field Type	Field Description
RMAIEn	UINT1	Remote muliframe alarm indication detect enable
RMAIInd	UINT1	Remote multiframe alarm indication status indicator
AISdEn	UINT1	AIS (low zero bit density) detection enable
AISdInd	UINT1	AIS (low zero bit density) status indicator
redEn	UINT1	Red alarm detection enable
redInd	UINT1	Red alarm status indicator
AISEn	UINT1	AIS (unframed all ones) detect enable
AISInd	UINT1	AIS (unframed all ones) status indicator
FEBEEn	UINT1	far end block error detection enable
CRCEn	UINT1	CRC error detection enable
sa4En	UINT1	National bit codeword 4 change detection enable
sa5En	UINT1	National bit codeword 5 change detection enable
sa6En	UINT1	National bit codeword 6 change detection enable
sa7En	UINT1	National bit codeword 7 change detection enable
sa8En	UINT1	National bit codeword 8 change detection enable
oOOfEn	UINT1	Out of offline frame detection enable
oOOFInd	UINT1	Out of offline frame status indicator
RAIcCRCEn	UINT1	Remote alarm indication and continuous CRC detection enable
RAIcCRCInd	UINT1	Remote alarm indication and continuous CRC status indicator

Field Name	Field Type	Field Description
cFEBEEn	UINT1	Continuous far end block error detection enable
cFEBEInd	UINT1	Continuous far end block error status indicator
v52linkEn	UINT1	V5.2 link identification detection enable
v52linkInd	UINT1	V5.2 link identification status indicator
brfpEn	UINT1	Basic frame boundary interrupt enable
icsmfpEn	UINT1	CRC-4 sub-multiframe boundary interrupt enable
icmfpEn	UINT1	CRC-4 multiframe boundary interrupt enable
ismfpEn	UINT1	Signaling multiframe boundary enable

Table 19: COMET-QUAD ISR SubMask: sCMQ_ISR_MASK_TDPR

Field Name	Field Type	Field Description
printEn	UINT1	Performance report ready detection enable
fullEn	UINT1	TDPR FIFO full detection enable
ovrEn	UINT1	TDPR FIFO overrun detection enable
udrEn	UINT1	TDPR FIFO underrun detection enable
lfillEn	UINT1	TDPR FIFO low level fill threshold detection enable
fullInd	UINT1	TDPR FIFO full indicator
blFillInd	UINT1	TDPR FIFO below lower threshold indicator

Table 20: COMET-QUAD ISR SubMask: sCMQ_ISR_MASK_RDLC

Field Name	Field Type	Field Description
rdlcEn	UINT1	Receive data link control interrupt enable
pcktInLastBInd	UINT1	Last byte of non-aborted packet in FIFO indicator

Field Name	Field Type	Field Description
colsInd	UINT1	Change of Link Status indicator
fifoOvrInd	UINT1	RDLC FIFO overrun indicator
fifoEmptyInd	UINT1	RDLC FIFO empty indicator

Table 21: COMET-QUAD ISR SubMask: sCMQ_ISR_MASK_PRBS

Field Name	Field Type	Field Description
syncEn	UINT1	Change in PRBS/PRGD checker state detection enable
syncInd	UINT1	PRBS/PRGD synchronization state indicator
beEn	UINT1	PRBS/PRGD receive bit error detection enable
xferEn	UINT1	PRBS/PRGD metrics updated detection enable
ovrnInd	UINT1	PRBS/PRGD transfer overwrite indicator

Table 22: COMET-QUAD ISR SubMask: sCMQ_ISR_MASK_SIGX

Field Name	Field Type	Field Description
COSSEn	UINT1	Change of signaling state detection enable
COSSInd	UINT1[32]	Change of signaling state per time slot indication

Table 23: COMET-QUAD ISR SubMask: sCMQ_ISR_MASK_PMON

Field Name	Field Type	Field Description
pmonEn	UINT1	Performance monitoring transfer interrupt enable
ovrnInd	UINT1	Performance monitoring count overrun indication

Table 24: COMET-QUAD ISR SubMask: `sCMQ_ISR_MASK_BTIF`

Field Name	Field Type	Field Description
parEn	UINT1	Parity error detection enable
speInd	UINT1	Signal parity error indicator
dpeInd	UINT1	Data parity error indicator

Other API Structures

The following structures are used by the application when executing API functions. The user is encouraged to refer to this section for detailed explanations of the configuration options available in the functions defined in section 4, Application Programming Interface.

Interface Configuration API Structures

Table 25: COMET-QUAD Transmit Jitter Attenuator Configuration: `sCMQ_CFG_TX_JAT`

Field Name	Field Type	Field Description
enable	UINT2	Enables the TJAT or selects TJAT bypass.
refDiv	UINT1	One less than the ratio between the frequency of the recovered clock and the frequency of the phase discriminator input
outputDiv	UINT1	One less than the ratio between the frequency of the output clock and the frequency of the phase discriminator input
FIFOselfCenter	UINT1	Enables the FIFO to self-center the read pointer upon FIFO overrun or underrun
preventOvfUndf	UINT1	Set to prevent FIFO underflows/overflows at the expense of limited jitter attenuation.
outputClock	eCMQ_TJAT_OUTPUT_CLK_SRC	Selects output clock source: CMQ_TJAT_OUTPUT_CLK_INTERN_JAT, CMQ_TJAT_OUTPUT_CLK_CTCLK, CMQ_TJAT_OUTPUT_CLK_FIFO_INPUT

Field Name	Field Type	Field Description
pllRefClock	eCMQ_TJAT_PLL_REF_CLK_SRC	Selects phase lock loop reference clock source: CMQ_TJAT_PLL_REF_CLK_FIFO_INPUT, CMQ_TJAT_PLL_REF_CLK_BACKPLANE, CMQ_TJAT_PLL_REF_CLK_RECOVERED, CMQ_TJAT_PLL_REF_CLK_CTCLK

Table 26: COMET-QUAD Receive Jitter Attenuator Configuration: sCMQ_CFG_RX_JAT

Field Name	Field Type	Field Description
enable	UINT2	Enables the RJAT or selects RJAT bypass
refDiv	UINT1	One less than the ratio between the frequency of the recovered clock and the frequency of the phase discriminator input
outputDiv	UINT1	One less than the ratio between the frequency of the output clock and the frequency of the phase discriminator input
FIFOselfCenter	UINT1	Enables the FIFO to self-center the read pointer upon FIFO overrun or underrun
preventOvfUndf	UINT1	Set to prevent FIFO underflows/overflows at the expense of limited jitter attenuation

Table 27: COMET-QUAD Receive Clock Configuration: sCMQ_CFG_RX_CLK

Field Name	Field Type	Field Description
recoverClkSel	eCMQ_RX_RECOVER_CLK	Clock recovery algorithm; select between high or low frequency jitter tolerance algorithms: CMQ_RECOVER_CLK_LOW_FREQ_JAT, CMQ_RECOVER_CLK_HIGH_FREQ_JAT

Field Name	Field Type	Field Description
LOSThresh	eCMQ_CDRC_LOS_THRESH	Digital loss of signal threshold in PCM samples: CMQ_LOS_THRESH_PCM_10_HDB3 (E1), CMQ_LOS_THRESH_PCM_15_B8ZS (T1), CMQ_LOS_THRESH_PCM_15_AMI, CMQ_LOS_THRESH_PCM_31, CMQ_LOS_THRESH_PCM_63, CMQ_LOS_THRESH_PCM_175

Table 28: COMET-QUAD Line Side Interface Analog Transmitter Configuration: ***sCMQ_CFG_TX_ANALOG***

Field Name	Field Type	Field Description
txEn	UINT2	If zero, transmit lines TXTIP and TXRING are held in high impedance state

Field Name	Field Type	Field Description
wvFormType	eCMQ_TX_LBO	Selection of the transmit pulse waveform type and waveform scale factor. Predefined tables are from the COMET and COMET-QUAD data sheet. To specify a user-defined waveform definition and scale factor, this should be set to CMQ_TX_LBO_USER_DEFINED. The user must then specify their own waveform data in wvFormData and their own waveform amplitude scaling factor in wvFormScFac. If this value is CMQ_TX_LBO_RETAIN_CURRENT, no changes are made. Select one of: CMQ_TX_LBO_T1_LONG_HAUL_0DB, CMQ_TX_LBO_T1_LONG_HAUL_7_5DB, CMQ_TX_LBO_T1_LONG_HAUL_15DB, CMQ_TX_LBO_T1_LONG_HAUL_22_5DB, CMQ_TX_LBO_T1_LONG_HAUL_TR62411_0DB CMQ_TX_LBO_T1_SHORT_HAUL_110FT, CMQ_TX_LBO_T1_SHORT_HAUL_220FT, CMQ_TX_LBO_T1_SHORT_HAUL_330FT, CMQ_TX_LBO_T1_SHORT_HAUL_440FT, CMQ_TX_LBO_T1_SHORT_HAUL_550FT, CMQ_TX_LBO_T1_SHORT_HAUL_660FT, CMQ_TX_LBO_T1_SHORT_HAUL_TR62411_110FT, CMQ_TX_LBO_T1_SHORT_HAUL_TR62411_220FT, CMQ_TX_LBO_T1_SHORT_HAUL_TR62411_330FT, CMQ_TX_LBO_T1_SHORT_HAUL_TR62411_440FT, CMQ_TX_LBO_T1_SHORT_HAUL_TR62411_550FT, CMQ_TX_LBO_T1_SHORT_HAUL_TR62411_660FT, CMQ_TX_LBO_E1_75OHM, CMQ_TX_LBO_E1_120OHM, CMQ_TX_LBO_USER_DEFINED, CMQ_TX_LBO_RETAIN_CURRENT

Field Name	Field Type	Field Description
wvFormScFac	UINT1	Amplitude control of DAC output. Increments are in 11.14 mA. This parameter is only used if wvFormType = CMQ_TX_LBO_USER_DEFINED. A value of 0 tri-states the output line and the max value is 21 (234mA). If the transmit waveform type is not CMQ_TX_LBO_USER_DEFINED, the value corresponding to the specified line build out configuration is applied.
wvFormData	UINT1[24][5]	User defined waveform. 24 7-bit samples of five unit intervals.
fuseDataSel	eCMQ_TX_FUSE_DATA	Select either fuse programming burned into the transmit LIU with CMQ_TX_FUSE_DATA_LIU_FUSE or user defined with CMQ_TX_FUSE_DATA_USER_DEFINED
alogTstPosCtrl	INT1	Used when fuseDataSel is CMQ_TX_FUSE_DATA_USER_DEFINED. Controls the digital to analog converted positive current control in steps of 0.78125% in either the negative or positive direction. Valid range is -63 to +63.
alogTstNegCtrl	INT1	Used when fuseDataSel is CMQ_TX_FUSE_DATA_USER_DEFINED. Controls the digital to analog converted negative current control in steps of 0.78125% in either the negative or positive direction. Valid range is -63 to +63.

Table 29: COMET-QUAD Analog Receiver Configuration: ***sCMQ_CFG_RX_ANALOG***

Field Name	Field Type	Field Description
aLosThreshold	eCMQ_RX_ALOS_THRESH	Analog loss of signal threshold. Note that the device requires that both the detection and clearance thresholds be the same so this value is applied to both the clearance and detection thresholds. One of: CMQ_RX_ALOS_9DB_THRESH, CMQ_RX_ALOS_14_5DB_THRESH, CMQ_RX_ALOS_20DB_THRESH, CMQ_RX_ALOS_22DB_THRESH, CMQ_RX_ALOS_25DB_THRESH, CMQ_RX_ALOS_30DB_THRESH, CMQ_RX_ALOS_31DB_THRESH, CMQ_RX_ALOS_35DB_THRESH
aLosDetectPeriod	UINT1	Duration for declaring analog loss of signal. The actual duration used is 16 x aLosDetectPeriod pulse intervals
aLosClearPeriod	UINT1	Duration for clearing analog loss of signal. The actual duration used is 16 x aLosClearPeriod pulse intervals

Field Name	Field Type	Field Description
eqFreq	eCMQ_RX_ALOG_EQ_FREQ	Equalizer feedback loop frequency. One of: CMQ_RX_EQ_FREQ_T1_24_125KHZ (24.125 kHz), CMQ_RX_EQ_FREQ_T1_12_063KHZ (12.063 kHz), CMQ_RX_EQ_FREQ_T1_8_0417KHZ (8.0417 kHz), CMQ_RX_EQ_FREQ_T1_6_0313KHZ (6.0313 kHz), CMQ_RX_EQ_FREQ_T1_4_8250KHZ (4.8250 kHz), CMQ_RX_EQ_FREQ_T1_4_0208KHZ (4.0208 kHz), CMQ_RX_EQ_FREQ_T1_3_4464KHZ (3.4464 kHz), CMQ_RX_EQ_FREQ_T1_3_0156KHZ (3.0156 kHz), CMQ_RX_EQ_FREQ_E1_32_000KHZ (32.000 kHz), CMQ_RX_EQ_FREQ_E1_16_000KHZ (16.000 kHz), CMQ_RX_EQ_FREQ_E1_10_667KHZ (10.667 kHz), CMQ_RX_EQ_FREQ_E1_8_000KHZ (8.000 kHz), CMQ_RX_EQ_FREQ_E1_6_40KHZ (6.40 kHz), CMQ_RX_EQ_FREQ_E1_5_333KHZ (5.333 kHz), CMQ_RX_EQ_FREQ_E1_4_5714KHZ (4.5714 kHz), CMQ_RX_EQ_FREQ_E1_4_0KHZ (4.0 kHz)

Field Name	Field Type	Field Description
eqFdBckPer	eCMQ_RX_ALOG_EQ_PER	Specifies the time interval that the equalizer dB counter must be stable before declaring stable equalization state. The period is based on equalizer feedback loop frequency. One of: CMQ_RX_EQ_VALID_PERIOD_32, CMQ_RX_EQ_VALID_PERIOD_64, CMQ_RX_EQ_VALID_PERIOD_128 CMQ_RX_EQ_VALID_PERIOD_256
ramType	eCMQ_RX_LINE_EQ	Selects either user defined receiver equalizer RAM in eqCoef, a pre-defined equalizer RAM from the COMET and COMET-QUAD data sheet, or no change to the equalizer settings: CMQ_RX_LINE_EQ_RAM_T1, CMQ_RX_LINE_EQ_RAM_E1, CMQ_RX_LINE_EQ_USER_DEFINED, CMQ_RX_LINE_EQ_RETAIN_CURRENT
eqCoef	UINT4[256]	Programmable equalizer coefficients. If ramType is CMQ_RX_LINE_EQ_USER_DEFINED, this array should contain the new coefficients.
squelchEn	UINT1	Enable/disable data squelching (force data to all 0's) upon analog loss of signal detection

Table 30: COMET-QUAD Backplane Access Configuration: `sCMQ_BACKPLANE_ACCESS_CFG`

Field Name	Field Type	Field Description
masterMode	UINT1	Selects slave or master backplane mode
dataMode	eCMQ_BACKPLANE_DATA_MODE	Data format mode. Only used if masterMode is set (clock master), otherwise ignored. Select one of: CMQ_BACKPLANE_FULL_FRAME_MODE, CMQ_BACKPLANE_NX56K_MODE, CMQ_BACKPLANE_NX64K_MODE, CMQ_BACKPLANE_NX64K_E1_MODE,
clkTimes2	UINT1	Select clock mode multiplication by two. If set, clock operates at twice the backplane rate.
dataRate	eCMQ_BACKPLANE_DATA_RATE	The backplane data rate. Should be one of: CMQ_BACKPLANE_CLK_RATE_1544, CMQ_BACKPLANE_CLK_RATE_2048, CMQ_BACKPLANE_CLK_RATE_8192,

Table 31: COMET-QUAD Backplane Receive Frame Configuration: `sCMQ_CFG_BRIF_FRM`

Field Name	Field Type	Field Description
fpMaster	UINT1	Selects frame pulse signal master/slave
fpmMode	eCMQ_BACKPLANE_RX_FRAMEPULSE_MODE	Configure the type of frame pulse transmitted on the backplane. Only used if frame pulse is master. Should be one of: CMQ_BACKPLANE_RX_FP_T1_HIGH_ON_SF_ESF, CMQ_BACKPLANE_RX_FP_T1E1_HIGH_EVERY_FRAME, CMQ_BACKPLANE_RX_FP_E1_HIGH_ON_CRC_MFRM, CMQ_BACKPLANE_RX_FP_E1_HIGH_ON_SIG_MFRM, CMQ_BACKPLANE_RX_FP_E1_COMP_MFRM , CMQ_BACKPLANE_RX_FP_E1_HIGH_ON_OVERHEAD
fpInvEn	UINT1	Enable inversion of the frame pulse signal
parInsEn	UINT1	Enable parity insertion
oddPar	UINT1	Odd/even parity selection. Only used if parity insertion enabled

Field Name	Field Type	Field Description
extParEn	UINT1	Enable extension of parity over current and previous frame
fBitFix	UINT1	Enable fixing of F bit, only valid when not in parity insertion mode
fBitPol	UINT1	Polarity of the F bit, valid only when fBitFix is 1 and not in parity mode
fpFrmOffset	UINT1	Offset in bytes between the framing pulse and start of next frame: valid range: 0 - 127 bytes
fpBitOffsetEn	UINT1	Enables offset between frame pulse and first timeslot in bits
fpBitOffset	UINT1	If bit offset is enabled, this value will be applied as the bit offset between the frame pulse and the first timeslot. This field should be a 3 bit value.
altFDLEn	UINT1	Enabling this option causes the framing bit to contain FDL data. Only supported for T1 ESF frame formats
tslotMapFormat	eCMQ_BACKPLANE_TIMESLOT_MAP	Selects timeslot mapping format when mapping a 1.544 MHz line onto a 2.048 MHz backplane. If this timeslot translation is not required (i.e. when the line rate matches the backplane rate) this value is ignored Should be one of: CMQ_BACKPLANE_TIMESLOT_MAP_3_OF_4, CMQ_BACKPLANE_TIMESLOT_MAP_24_OF_32

Table 32: COMET-QUAD Backplane Transmit Frame Configuration: sCMQ_CFG_BTIF_FRM

Field Name	Field Type	Field Description
fpMaster	UINT1	Selects frame pulse signal master/slave
fpInvEn	UINT1	Enable inversion of the frame pulse signal
oddPar	UINT1	Odd/even parity selection
extParEn	UINT1	Enable extension of parity over current and previous frame

Field Name	Field Type	Field Description
fpFrmOffset	UINT1	Offset in bytes between the framing pulse and start of next frame: valid range: 0 - 127 bytes
T1ESFAlign	UINT1	For T1 mode, enables ESF alignment as opposed to SF alignment. This option is not supported for E1 mode
fpBitOffsetEn	UINT1	Enables offset between frame pulse and first timeslot in bits
fpBitOffset	UINT1	If bit offset is enabled, this value will be applied as the bit offset between the frame pulse and the first timeslot (3 bit value)
tslotMapFormat	eCMQ_BACKPLANE_TIMESLOT_MAP	Selects timeslot mapping format when mapping 2.048 MHz backplane onto 1.544 MHz line. If this timeslot translation is not required (i.e., when the line rate matches the backplane rate), this value is ignored. Should be one of: CMQ_BACKPLANE_TIMESLOT_MAP_3_OF_4 CMQ_BACKPLANE_TIMESLOT_MAP_24_OF_32

*Table 33: COMET-QUAD H-MVIP Configuration: **sCMQ_CFG_HMVIP***

Field Name	Field Type	Field Description
rxHMVIPMode	eCMQ_BACKPLANE_HMVIP_MODE	Enable HMVIP, enable CCS, or disable both on the receive backplane interface. Should be one of: CMQ_BACKPLANE_HMVIP_MODE, CMQ_BACKPLANE_HMVIP_CCS, CMQ_BACKPLANE_HMVIP_DISABLE
txHMVIPMode	eCMQ_BACKPLANE_HMVIP_MODE	Enable HMVIP, enable CCS, or disable both on the transmit backplane interface. Should be one of: CMQ_BACKPLANE_HMVIP_MODE, CMQ_BACKPLANE_HMVIP_CCS, CMQ_BACKPLANE_HMVIP_DISABLE

Field Name	Field Type	Field Description
txEnableCCS InTs15	UINT1[4]	Selects whether or not common channel signaling from the transmit backplane interface is inserted into timeslot 15. Only used when E1 mode and txHMVIPMode is set for CCS
txEnableCCS InTs16	UINT1[4]	Selects whether or not common channel signaling from the transmit backplane interface is inserted into timeslot 16. Only used when E1 mode and txHMVIPMode is set for CCS
txEnableCCS InTs31	UINT1[4]	Selects whether or not common channel signaling from the transmit backplane interface is inserted into timeslot 31. Only used when E1 mode and txHMVIPMode is set for CCS

Table 34: COMET-QUAD Receive Elastic Store Configuration: sCMQ_CFG_RX_ELST

Field Name	Field Type	Field Description
elstEnable	UINT1	Enable/disable the elastic store
idleCode	UINT1	Elastic store idle code
CCSidleCode	UINT1	Elastic store CCS idle code (Comet-Quad only)

T1/E1 Framers API Structures

Table 35: COMET-QUAD T1 Transmit Framer Configuration: sCMQ_CFG_T1TX_FRM

Field Name	Field Type	Field Description
frmMode	eCMQ_FRAME_MODE	T1 transmit framing format. Should be one of: CMQ_FRM_MODE_T1_SF, CMQ_FRM_MODE_T1_DM, CMQ_FRM_MODE_T1_SLC96, CMQ_FRM_MODE_T1_DM_FDL, CMQ_FRM_MODE_T1_ESF, CMQ_FRM_MODE_T1_SF_JPN_ALARM, CMQ_FRM_MODE_T1_DM_JPN_ALARM, CMQ_FRM_MODE_T1_SLC96_JPN_ALARM, CMQ_FRM_MODE_T1_DM_FDL_JPN_ALARM, CMQ_FRM_MODE_T1_JT_G704, CMQ_FRM_MODE_T1_UNFRAMED

Field Name	Field Type	Field Description
zSupFormat	eCMQ_T1_ZSUP_FORMAT	Zero code suppression format to be used. Should be one of: CMQ_T1_ZSUP_NONE, CMQ_T1_ZSUP_GTE, CMQ_T1_ZSUP_DDS, CMQ_T1_ZSUP_BELL
SFSigAlignerEn	UINT1	Enables the signaling aligner to ensure that signaling alignment between superframes on the backplane and the transmit framer is maintained. Note that when using this option, the ESF alignment option in the transmit backplane frame pulse configuration must be off.

Table 36: COMET-QUAD T1 Receive Framer Configuration: sCMQ_CFG_T1RX_FRM

Field Name	Field Type	Field Description
frmMode	eCMQ_FRAME_MODE	T1 receive framing format. Should be one of: CMQ_FRM_MODE_T1_SF, CMQ_FRM_MODE_T1_DM, CMQ_FRM_MODE_T1_SLC96, CMQ_FRM_MODE_T1_DM_FDL, CMQ_FRM_MODE_T1_ESF, CMQ_FRM_MODE_T1_SF_JPN_ALARM, CMQ_FRM_MODE_T1_DM_JPN_ALARM, CMQ_FRM_MODE_T1_SLC96_JPN_ALARM, CMQ_FRM_MODE_T1_DM_FDL_JPN_ALARM, CMQ_FRM_MODE_T1_JT_G704, CMQ_FRM_MODE_T1_UNFRAMED
outOfFrameCriteria	eCMQ_T1_FRAME_LOSS_THRESH	Selects criteria for declaring out of frame. Note that if frmMode is T1DM, this parameter has no effect as the criteria is fixed at 4 out of 12. Select one of: CMQ_T1_OOF_2OF4, CMQ_T1_OOF_2OF5, CMQ_T1_OOF_2OF6

Field Name	Field Type	Field Description
frmESFAlgo	eCMQ_T1_ESF_FRAME_ALGO	For ESF framing formats, this parameter selects whether to use the ESF frame alignment algorithm based on a continuous CRC-6 calculation or the ESF frame alignment algorithm where frame alignment is not declared while there are two framing candidates. Select one of CMQ_T1_ESF_FRAME_ALGO_ONE_CANDIDATE or CMQ_T1_ESF_FRAME_ALGO_CRC_6
COFACntEn	UINT1	Enable counting of change of frame alignment (COFA) events rather than out of frame alignment (OOF) events in the PMON

Table 37: COMET-QUAD E1 Transmit Framer Configuration: sCMQ_CFG_E1TX_FRM

Field Name	Field Type	Field Description
frmMode	eCMQ_FRAME_MODE	E1 transmit framing format. Should be one of: CMQ_FRM_MODE_E1, CMQ_FRM_MODE_E1_CRC_MFRM, CMQ_FRM_MODE_E1_UNFRAMED
ts16Signaling	eCMQ_E1_SIG_INSERTION	Type of signaling to insert into timeslot 16. Select from: CMQ_E1_SIG_INS_NONE, CMQ_E1_SIG_INS_HDLC_CCS, CMQ_E1_SIG_INS_CAS
insNatIntBitEn	UINT1	Enables insertion of international and national bits into timeslot 0 of NFAS frames
insXtraBitsEn	UINT1	Enables insertion of extra bits into timeslot 16 of frame 0 in a signaling multiframe
insFEBEEn	UINT1	If CRC multiframe mode selected, this value enables insertion of far end block error (FEBE) bits.

Table 38: COMET-QUAD E1 Receive Framer Configuration: `sCMQ_CFG_E1RX_FRM`

Field Name	Field Type	Field Description

`frmMode`	`eCMQ_FRAME_MODE`	E1 transmit framing format. Should be one of:
		
		`CMQ_FRM_MODE_E1,`
		`CMQ_FRM_MODE_E1_CRC_MFRM,`
		`CMQ_FRM_MODE_E1_UNFRAMED`

`CASAlignmentEn`	`UINT1`	Enable alignment to channel associative signaling multiframes
`CRC2NCRCEn`	`UINT1`	Enable checking of CRC multiframe in CRC to non-CRC internetworking mode
`noReframeOnErrEn`	`UINT1`	Disable reframing upon any error event
`reframeOnXSCrcErrEn`	`UINT1`	Enable a forced reframe to occur when excessive CRC errors are reported
`lofBit2CritEn`	`UINT1`	Enable loss of frame criteria: Bit 2, timeslot 0 of NFAS frames is 0 in 3 consecutive frames
`NFASErrEn`	`UINT1`	Enable errors in bit 2 of timeslot 0 of NFAS frames to contribute towards the framing error count
`multFASEOneFEEn`	`UINT1`	Enable multiple FAS errors to generate a single framing error. If NFAS errors are counted towards framing errors, enabling this option includes Bit 2 of timeslot 0 in NFAS frames as part of the FAS word

Field Name	Field Type	Field Description
mfrmLossAlignCrit	eCMQ_E1_LOSS_MFRM_ALIGN_TS16_CRIT	Select additional timeslot 16 criteria for declaring loss of signaling mutliframe alignment of either all 0's in timeslot 16 for 1 or 2 multiframes. Note that the selection made here is in addition to the criteria of 2 consecutive mutliframe alignment pattern errors which always generates a loss of signaling multiframe alignment declaration Select from: CMQ_E1_LOSS_MFRM_ALIGN_TS16_CRIT_NONE, CMQ_E1_LOSS_MFRM_ALIGN_TS16_CRIT_ZERO_1_MFRM, CMQ_E1_LOSS_MFRM_ALIGN_TS16_CRIT_ZERO_2_MFRM
AISCriteria	eCMQ_E1_AIS_CRIT	Criteria for declaring AIS. Select between less than 3 zeroes in 512 consecutive bits or 2 consecutive periods of less than 3 zeroes in 512 consecutive bits: CMQ_E1_AIS_CRIT_3Z_IN_512BITs CMQ_E1_AIS_CRIT_2_PERIODS_3Z_IN_512BITS
RAICriteria	eCMQ_E1_RAI_CRIT	Criteria for declaring RAI. Select between declaration upon receiving any A-bit (bit 3, timeslot 0) that is 1 or declaration upon four consecutive A-bits that are one CMQ_E1_RAI_CRIT_ALL_A_1, CMQ_E1_RAI_CRIT_4_CONSEC_A_1

Alarm Control and Inband Communications API Structures

Table 39: COMET-QUAD HDLC Link Extraction/Insertion Location Configuration: sCMQ_CFG_HDLC_LINK

Field Name	Field Type	Field Description
useT1DataLink	UINT1	Specifies the use of the T1 FDL (ESF or T1DM with FDL modes). For COMET, only valid for first HDLC controller. In E1 mode, this value should be 0

Field Name	Field Type	Field Description
evenFrames	UINT1	Enable link extraction/insertion from even frames. Ignored when using T1 data link
oddFrames	UINT1	Enable link extraction/insertion from odd frames. Ignored when using T1 data link
timeslot	UINT1	Timeslot to extract data link from (0 based). Ignored if both oddFrames and evenFrames are 0 or if using T1 data link
dataLinkBitMask	UINT1	Bit mask selecting which of the bits in a timeslot/channel are to be used. Ignored if both oddFrames and evenFrames are 0 or if using T1 Data Link

Table 40: COMET-QUAD HDLC Receiver Configuration: sCMQ_CFG_HDLC_RX

Field Name	Field Type	Field Description
linkLocation	sCMQ_CFG_HDLC_LINK	Specifies where to extract the HDLC link from
addrMatchEn	UINT1	Forces detection of packets with either an address of all 1's or an address matching either the primary or secondary addresses
addrMaskEn	UINT1	Specifies the receive HDLC controller to ignore the two least significant bits of the primary and secondary addresses when looking for matches

Table 41: COMET-QUAD HDLC Receiver Configuration: sCMQ_CFG_HDLC_TX

Field Name	Field Type	Field Description
linkLocation	sCMQ_CFG_HDLC_LINK	Specifies where to insert the HDLC link
flagShareEn	UINT1	Enables sharing of start/end flags between packets
crcFCSEn	UINT1	Enables CRC frame check sequence generation
pmRepEn	UINT1	Enables performance report transmission. For COMET devices, this option is valid only for the first HDLC controller.

Status and Statistics API Structures

Table 42: COMET-QUAD Clock Status Structure: `sCMQ_CLK_STATUS`

Field Name	Field Type	Field Description
XCLCKactive	UINT1	Indicates XCLK is alive. Note that this value on COMET-QUAD devices is not quadrant based.
BTCLKactive	UINT1	Indicates BTCLK is alive
CTCLKactive	UINT1	Indicates CTCLK is alive
BRCLKactive	UINT1	Indicates BRCLK is alive
RCLKIactive	UINT1	Indicates RCLKI is alive
CSULock	UINT1	Indicates CSU has locked onto XCLK. Note that this value on COMET-QUAD devices is not quadrant based.

Table 43: COMET-QUAD Framer Statistics: `sCMQ_FRM_CNTS`

Field Name	Field Type	Field Description
frmErrCnt	UINT1[4]	Framing error count (T1 and E1)
T1_OOF_COFA_Cnt	UINT4[4]	Out of frame count or change of frame alignment count depending on option selected for T1 receive framer
E1_FEBECnt	UINT2[4]	E1 Far end block error count
T1_BitErrCnt	UINT4[4]	T1 bit error event count
E1_CRCErrCnt	UINT2[4]	E1 CRC error count
lcvCnt	UINT4[4]	Line code violation count (T1 and E1)

Table 44: COMET-QUAD Framer Status: `sCMQ_FRM_STATUS`

Field Name	Field Type	Field Description
lossOfSignal	UINT1[4]	Loss of signal indicator
lossOfFrame	UINT1[4]	Loss of frame alignment indicator

Field Name	Field Type	Field Description
AIS	UINT1[4]	AIS alarm indicator
yelAlm	UINT1[4]	Yellow alarm (E1 RAI) indicator
E1_lossOfSMF	UINT1[4]	E1 loss of signaling multiframe alignment indicator (unused for T1)
E1_lossOfCMF	UINT1[4]	E1 loss of CRC-4 multiframe alignment indicator (unused for T1)
E1_ts16RAI	UINT1[4]	E1 timeslot 16 RAI alarm indicator (unused for T1)

Table 45: COMET-QUAD T1 Automatic Performance Monitoring Message: sCMQ_STAT_APRM

Field Name	Field Type	Field Description
octet2	UINT1	Service access point identifier, command response flag, extended address flag
octet3	UINT1	Terminal endpoint identifier, extended address2 flag
octet4	UINT1	Performance report control field
octet5	UINT1	Event threshold flags: see data sheet
octet6	UINT1	Event threshold flags: see data sheet

3.3 Structures in the Driver's Allocated Memory

These structures are defined and used by the driver and are part of the context memory allocated when the driver is opened. These structures are the Module Data Block (MDB) and the Device Data Block (DDB).

Module Data Block: MDB

The MDB is the top-level structure for the module. It contains configuration data about the module level code and pointers to configuration data about the device level codes.

- `errModule` indicates specific error codes returned by API functions that are not passed directly to the application. Most of the module API functions return a specific error code directly. When the returned code is not `CMQ_SUCCESS`, this indicates that the top-level function was not able to carry the specified error code back to the application. Under those circumstances, the proper error code is recorded in this element. The element is the first in the structure so that the USER can cast the MDB pointer into an INT4 pointer and retrieve the local error. This eliminates the need to include the MDB template into the application code
- `valid` indicates that this structure has been properly initialized and may be read by the USER
- `stateModule` contains the current state of the module and can be any one of: `CMQ_MOD_START`, `CMQ_MOD_IDLE` or `CMQ_MOD_READY`

Table 46: COMET-QUAD Module Data Block: `sCMQ_MDB`

Field Name	Field Type	Field Description
`errModule`	`INT4`	Global error indicator for module calls
`valid`	`UINT2`	Indicates that this structure has been initialized
`stateModule`	`eCMQ_MOD_STATE`	Module state; can be any one of the following: `CMQ_MOD_START`, `CMQ_MOD_IDLE` or `CMQ_MOD_READY`
`maxDevs`	`UINT2`	Maximum number of devices supported
`numDevs`	`UINT2`	Number of devices currently registered
`maxInitProfs`	`UINT2`	Maximum number of initialization profiles
`pddb`	`sCMQ_DDB *`	(array of) Device Data Blocks (DDB) in context memory
`pinitProfs`	`sCMQ_DIV *`	(array of) Initialization profiles in context memory

Device Data Block: DDB

The DDB is the top-level structure for each device. It contains configuration data about the device level code and pointers to configuration data about device level sub-blocks.

- errDevice indicates specific error codes returned by API functions that are not passed directly to the application. Most of the device API functions return a specific error code directly. When the returned code is CMQ_FAILURE, this indicates that the top-level function is not able to carry the specified error code back to the application. In addition, some device functions do not return an error code. Under those circumstances, the proper error code is recorded in this element. This element is the first in the structure so that the USER can cast the DDB pointer to an INT4 pointer and retrieve the local error. This eliminates the need to include the DDB template in the application code
- valid indicates that this structure has been properly initialized and may be read by the USER
- stateDevice contains the current state of the device and can be any one of: CMQ_START, CMQ_PRESENT, CMQ_ACTIVE or CMQ_INACTIVE
- usrCtxt is a value that can be used by the USER to identify the device during the execution of the callback functions. It is passed to the driver when cometqAdd is called and returned to the USER when a callback function is invoked

Table 47: COMET-QUAD Device Data Block: sCMQ_DDB

Field Name	Field Type	Field Description
errDevice	INT4	Global error indicator for device calls
valid	UINT2	Indicates that this structure has been initialized
stateDevice	CMQ_DEV_STATE	Device State; can be one of the following: CMQ_PRESENT, CMQ_ACTIVE or CMQ_INACTIVE
baseAddr	UINT1 *	Base address of the Device
usrCtxt	void *	Stores the user's context for the device. It is passed as an input parameter when the driver invokes an application callback
profileNum	UINT2	Profile number used at initialization
pollISR	sCMQ_POLL	Indicates the current mode of interrupt processing
cbackFramer	CMQ_CBACK	Address for the callback function for Framer Events
cbackIntf	CMQ_CBACK	Address for the callback function for Line Side Interface Events

Field Name	Field Type	Field Description
cbackAlarmInBand	CMQ_CBACK	Address for the callback function for Alarm InBand Events
cbackSigInsExt	CMQ_CBACK	Address for the callback function for Signal Insertion and Extraction Events
cbackPMon	CMQ_CBACK	Address for the callback function for Performance Monitoring Events
cbackSerialCtl	CMQ_CBACK	Address for the callback function for Serial Control Events
cometqFlag	UINT1	Indicates the device type: COMET (= 0) or COMET-QUAD (= 1)
modeE1	UINT1	Operational mode of the device. A one indicates E1 mode and a value of zero indicates T1 mode
mask	sCMQ_ISR_MASK	Interrupt enable mask

3.4 Structures Passed through RTOS Buffers

Interrupt Service Vector: ISV

This buffer structure is used to capture the status of the device (during a poll or ISR processing) for use by the Deferred Processing Routine (DPR). It is the application's responsibility to create a pool of ISV buffers (using this template to determine the buffer's size) when the driver calls the user-supplied `sysCometqBufferStart` function. An individual ISV buffer is then obtained by the driver via `sysCometqISVBufferGet` and returned to the 'pool' via `sysCometqISVBufferRtn`.

Table 48: COMET-QUAD Interrupt Service Vector: `sCMQ_ISV`

Field Name	Field Type	Field Description
deviceHandle	sCMQ_HNDL	Handle to the device generating interrupts
mask	sCMQ_ISR_MASK	ISR mask with interrupt status information

Deferred Processing Vector: DPV

This structure is used in two ways. First, it is used to determine the size of the buffer required by the RTOS for use in the driver. Second, it defines the format of the data that is assembled by the DPR and sent to the application code. Note: the application code is responsible for returning this buffer to the RTOS buffer pool.

The DPR reports events to the application using user-defined callbacks. The DPR uses each callback to report a functionally-related group of events. Refer to section 4.13 for a description of the COMET and COMET-QUAD callback functions, and to Appendix C for a list of events and bit masks used in the interpretation of the DPV.

Within the callback there are three event fields. The callback routine can determine what events are being passed by the DPR by examining the bits in these fields. Appendix C provides a listing of the bit masks required to interpret the bits in the event fields.

In addition to the three event fields, the DPR also contains fields that hold additional information that is relevant only when the corresponding event has occurred.

For all interrupts that are processed by the DPR simultaneously, only a single invocation of the relevant callback functions takes place. For the multiple framers on a COMET-QUAD device, there is a single callback for all framers. The framer number can be identified by the index into the event fields for which an event bit is set. For COMET devices, only the first element of the arrays in the DPV are used.

Table 49: COMET-QUAD Deferred Processing Vector: `sCMQ_DPV`

Field Name	Field Type	Field Description
`event1`	`UINT4[4]`	Event indicator bit field
`event2`	`UINT4[4]`	Event indicator bit field
`event3`	`UINT4[4]`	Event indicator bit field
`CDRCLosInd`	`UINT1[4]`	CDRC loss of signal status indicator. Used with event `CMQ_EVENT_CDRC_LOS`.
`CDRCAltLosInd`	`UINT1[4]`	CDRC alternate loss of signal status indicator. Used with event `CMQ_EVENT_CDRC_ALT_LOS`.
`RLPSLosInd`	`UINT1[4]`	RLPS analog loss of signal status indicator. Used with event `CMQ_EVENT_RLPS_ALOS`.
`T1FRMRMfpInd`	`UINT1[4]`	T1-FRMR mimic frame pattern status indicator. Used with event `CMQ_EVENT_T1_FRMR_MIMIC_FRM`.

Field Name	Field Type	Field Description
T1FRMRInfrInd	UINT1[4]	T1-FRMR in frame status indicator. Used with event CMQ_EVENT_T1_FRMR_INFRM.
IBCDLbaInd	UINT1[4]	Activate loopback code status indicator. Used with event CMQ_EVENT_IBCD_LPBCK_ACT_CODE.
IBCDLbdInd	UINT1[4]	Deactivate loopback code status indicator. Used with event CMQ_EVENT_IBCD_LPBCK_DEACT_CODE.
ALMIYelInd	UINT1[4]	ALMI yellow alarm status indicator. Used with event CMQ_EVENT_ALMI_YELLOW_ALARM.
ALMIRedInd	UINT1[4]	ALMI red alarm status indicator. Used with event CMQ_EVENT_ALMI_RED_ALARM.
ALMIAISInd	UINT1[4]	ALMI AIS status indicator. Used with event CMQ_EVENT_ALMI_AIS_ALARM.
PDVDPdvInd	UINT1[4]	PDVD pulse density violation status indicator. Used with event CMQ_EVENT_PDVD_PULSE_DENSITY_VIOLT.
XPDEPdvInd	UINT1[4]	XPDE pulse density enforcer violation status indicator. Used with event CMQ_EVENT_XPDE_PULSE_DENSITY_VIOLT.
PRBSSyncInd	UINT1[4]	PRBS/PRGD synchronization state indicator. Used with event CMQ_EVENT_PRBS_PAT_SYNC.
PRBSOvrnInd	UINT1[4]	PRBS/PRGD transfer overwrite indicator. Used with event CMQ_EVENT_PRBS_XFER_UPD.
E1FRMRC2nciwInd	UINT1[4]	E1-FRMR CRC to non-CRC internetworking status indicator. Used with event CMQ_EVENT_E1_FRMR_CRC2NCRC.
E1FRMROOFInd	UINT1[4]	E1-FRMR out of frame status indicator. Used with event CMQ_EVENT_E1_FRMR_OOF.
E1FRMOOSMFInd	UINT1[4]	E1-FRMR out of signaling multiframe status indicator. Used with event CMQ_EVENT_E1_FRMR_OOF_SMFRM.

Field Name	Field Type	Field Description
E1FRMOOCMFInd	UINT1[4]	E1-FRMR out of CRC-4 multiframe status indicator. Used with event CMQ_EVENT_E1_FRMR_OOF_CRC_MFRM.
E1FRMOoofInd	UINT1[4]	E1-FRMR out of offline frame status indicator. Used with event CMQ_EVENT_E1_FRMR_OOF_ALARM.
E1FRMRaiCrcInd	UINT1[4]	E1-FRMR remote alarm indication and continuous CRC status indicator. Used with event CMQ_EVENT_E1_FRMR_RAI_CONT_CRC_ALARM.
E1FRMCfebeInd	UINT1[4]	E1-FRMR far end block error status indicator. Used with event CMQ_EVENT_E1_FRMR_CONT_FEBE_ALARM.
E1FRMV52LinkInd	UINT1[4]	E1-FRMR V5.2 link identification status indicator. Used with event CMQ_EVENT_E1_FRMR_V52LINKID_ALARM.
E1FRMRaiInd	UINT1[4]	E1-FRMR remote alarm indication status indicator. Used with event CMQ_EVENT_E1_FRMR_RAI_ALARM.
E1FRMRmaiInd	UINT1[4]	E1-FRMR remote multiframe alarm indication status indicator. Used with event CMQ_EVENT_E1_FRMR_RMAI_ALARM.
E1FRMAisdInd	UINT1[4]	E1-FRMR AIS (low zero bit density) status indicator. Used with event CMQ_EVENT_E1_FRMR_AISD_ALARM.
E1FRMRedInd	UINT1[4]	E1-FRMR red alarm status indicator. Used with event CMQ_EVENT_E1_FRMR_RED_ALARM.
E1FRMAisInd	UINT1[4]	E1-FRMR AIS (unframed all ones) status indicator. Used with event CMQ_EVENT_E1_FRMR_AIS_ALARM.
COSStimeslot	UINT4[4]	Change of signaling state information bitmap. For E1, bit 0 represents timeslot 0 and bit 31 represents timeslot 31. For T1, bit 1 represents channel 1 and bit 24 represents channel 24. Used with event CMQ_EVENT_SIGX_COS_STATE

Field Name	Field Type	Field Description
RDLCCOLS	UINT1[4]	RDLC change of link state. Used with event CMQ_EVENT_RDLC_EVENT
RDLCPacketIn	UINT1[4]	RDLC packet indicator. Used with event CMQ_EVENT_RDLC_EVENT
RDLCFIFOOvr	UINT1[4]	RDLC FIFO overrun indicator. Used with event CMQ_EVENT_RDLC_EVENT
RDLCFIFOEmpty	UINT1[4]	RDLC FIFO underrun indicator. Used with event CMQ_EVENT_RDLC_EVENT
TDPRFIFOBelowThresh	UINT1[4]	TDPR FIFO below threshold indicator. Used with event CMQ_EVENT_TDPR_FIFO_FILL_LOWLVL_THRESH
TDPRFIFOFull	UINT1[4]	TDPR FIFO full indicator. Used with event CMQ_EVENT_TDPR_FIFO_FULL

3.5 Global Variable

Although most of the variables within the driver are not intended for used by the application code, there is one global variable that can be of great use to the application code.

This variable is called cometqMdb and it acts as a global pointer to the Module Data Block (MDB). The content of this global variable should be considered read-only by the application.

- errModule: This structure element is used to store an error code that specifies the reason for an API function's failure. The field is only valid for functions that do not return an error code or when a value of CMQ_FAILURE is returned.
- stateModule: This structure element is used to store the module state (as shown in Figure 3).
- pddb[ ]: An array of pointers to the individual Device Data Blocks. The USER is cautioned that a DDB is only valid if the valid flag is set. Note that the array of DDBs is in no particular order.
    - errDevice: This structure element is used to store an error code that specifies the reason for an API function's failure. The field is only valid for functions that do not return an error code or when a value of CMQ_FAILURE is returned.
    - stateDevice: This structure element is used to store the device state (as shown in Figure 3).

4 APPLICATION PROGRAMMING INTERFACE

This section of the manual provides a detailed description of each function that is a member of the COMET-QUAD driver Application Programming Interface (API). API functions typically execute in the context of an application task.

It is important to note that these functions are not re-entrant. This means that two application tasks can not invoke the same API at the same time. However the driver protects its data structures from concurrent accesses by the Application and the DPR task.

4.1 Module Management

The module management is a set of API functions that are used by the Application to open, start, stop and close the driver module. These functions initialize the driver; they also allocate memory and all the RTOS resources needed by the driver. They are also used to change the module state. For more information on the module states see the state diagram on page 21. For a typical module management flow diagram see page 23.

All module management functions are device independent, and thus have the same behavior irrespective of whether the driver is being used with COMET-QUAD or COMET devices.

Opening the Driver Module: cometqModuleOpen

This performs module level initialization of the device driver, which involves allocating all of the memory needed by the driver and initializing the internal structures.

Prototype `INT4 cometqModuleOpen(sCMQ_MIV *pmiv)`

Inputs `pmiv` : (pointer to) Module Initialization Vector

Outputs Places the address of the MDB into the MIV passed by the Application

Returns Success = `CMQ_SUCCESS`
Failure = <COMET-QUAD ERROR CODE>

Valid States `CMQ_MOD_START`

Side Effects Changes the MODULE state to `CMQ_MOD_IDLE`

Closing the Driver Module: cometqModuleClose

Performs module level shutdown of the driver. This involves deleting all devices being controlled by the driver (by calling `cometqDelete` for each device) and de-allocating all the memory allocated by the Driver.

Prototype `INT4 cometqModuleClose(void)`

Inputs None

Outputs None

Returns Success = `CMQ_SUCCESS`
Failure = <COMET-QUAD ERROR CODE>

Valid States ALL STATES

Side Effects Changes the MODULE state to `CMQ_MOD_START`

Starting the Driver Module: cometqModuleStart

Connects the RTOS resources to the Driver. This involves allocating semaphores and timers, initializing buffers, and installing the ISR handler and DPR task. Upon successful return from this function, the driver is ready to add devices.

Prototype `INT4 cometqModuleStart(void)`

Inputs None

Outputs None

Returns Success = `CMQ_SUCCESS`
Failure = <COMET-QUAD ERROR CODE>

Valid States `CMQ_MOD_IDLE`

Side Effects Changes the MODULE state to `CMQ_MOD_READY`

Stopping the Driver Module: cometqModuleStop

Disconnects the RTOS resources from the Driver. This involves de-allocating semaphores and timers, freeing-up buffers, and uninstalling the ISR handler and the DPR task. If there are any registered devices, `cometqDelete` is called for each.

Prototype `INT4 cometqModuleStop(void)`

Inputs None

Outputs None

Returns Success = `CMQ_SUCCESS`
Failure = <COMET-QUAD ERROR CODE>

Valid States `CMQ_MOD_READY`

Side Effects Changes the MODULE state to `CMQ_MOD_IDLE`

4.2 Profile Management

This section of the manual describes the functions that add, get and clear an initialization profile. Initialization profiles allow the user to store pre-defined Device Initialization Vectors (DIV) that are validated ahead of time. When the device initialization function is invoked only a profile number need be passed. This method simplifies and expedites the initialization process.

Adding an Initialization Profile: cometqAddInitProfile

Creates an initialization profile that is stored by the driver. A device can now be initialized by simply passing the initialization profile number.

Prototype `INT4 cometqAddInitProfile(sCMQ_DIV *pProfile, UINT2 *pProfileNum)`

Inputs
`pProfile` : (pointer to) initialization profile being added
`pProfileNum` : (pointer to) profile number to be assigned by the driver

Outputs `pProfileNum` : profile number assigned by the driver

Returns
Success = `CMQ_SUCCESS`
Failure = <COMET-QUAD ERROR CODE>

Valid States `CMQ_MOD_IDLE, CMQ_MOD_READY`

Side Effects None

Getting an Initialization Profile: cometqGetInitProfile

Gets the content of an initialization profile given its profile number.

Prototype `INT4 cometqGetInitProfile(UINT2 profileNum, sCMQ_DIV *pProfile)`

Inputs
`profileNum` : initialization profile number
`pProfile` : (pointer to) initialization profile

Outputs `pProfile` : contents of the corresponding profile

Returns
Success = `CMQ_SUCCESS`
Failure = <COMET-QUAD ERROR CODE>

Valid States `CMQ_MOD_IDLE, CMQ_MOD_READY`

Side Effects None

Deleting an Initialization Profile: cometqDeleteInitProfile

Deletes an initialization profile given its profile number.

Prototype `INT4 cometqDeleteInitProfile(UINT2 profileNum)`

Inputs `profileNum` : initialization profile number

Outputs None

Returns Success = `CMQ_SUCCESS`
Failure = <COMET-QUAD ERROR CODE>

Valid States `CMQ_MOD_IDLE, CMQ_MOD_READY`

Side Effects None

4.3 Device Management

The device management is a set of API functions that are used by the Application to control the device. These functions take care of initializing a device in a specific configuration, enabling the device's general activity as well as enabling interrupt processing for that device. They are also used to change the software state for that device. For more information on the device states see the state diagram on page 21. For a typical device management flow diagram see page 24.

Adding a Device: cometqAdd

This function verifies the presence of a new device in the hardware, then returns a handle back to the user. The device handle is passed as a parameter to most of the Device API Functions. It is used by the driver to identify the device on which the operation is to be performed.

When a COMET or COMET-QUAD device is added to the module, the TYPE field in the Revision/Chip ID register is used to determine whether the user is adding a COMET or COMET-QUAD device. The device data block stores the device type (COMET or COMET-QUAD) in the `cometqFlag` field.

Prototype `sCMQ_HNDL cometqAdd(void *usrCtxt, UINT1 *baseAddr, INT4 **pperrDevice)`

Inputs
`usrCtxt` : user context for this device
`baseAddr` : base address of the device
`pperrDevice` : (pointer to) an area of memory

Outputs ERROR code written to the MDB on failure

`pperrDevice` : (pointer to) errDevice (inside the DDB)

Returns `Device Handle` (to be used as an argument to most of the COMET-QUAD APIs) or NULL (pointer) in case of an error

Valid States `CMQ_MOD_READY`

Side Effects Changes the Device state to `CMQ_PRESENT`

Deleting a Device: cometqDelete

This function is used to remove the specified device from the list of devices being controlled by the COMET-QUAD driver. Deleting a device involves invalidating the DDB for that device and releasing its associated device handle.

This API call is identical for both COMET and COMET-QUAD devices.

Prototype `INT4 cometqDelete(sCMQ_HNDL deviceHandle)`

Inputs `deviceHandle` : device handle (from `cometqAdd`)

Outputs None

Returns Success = `CMQ_SUCCESS`
Failure = <COMET-QUAD ERROR CODE>

Valid States `CMQ_PRESENT`, `CMQ_ACTIVE`, `CMQ_INACTIVE`

Side Effects `None`

Initializing a Device: cometqInit

Initializes the Device Data Block (DDB) associated with that device during `cometqAdd`, applies a soft reset to the device and configures it according to the DIV passed by the Application. If the DIV is passed as a NULL, the profile number is used. A profile number of zero indicates that all the register bits are to be left in their default state (after a soft reset). Note that the profile number is ignored UNLESS the passed DIV is NULL.

To set the callbacks on a device while retaining the device in its reset state, the `initDevice` member of the DIV should be set to 0. The analogInit, backplaneInit, and framerInit members are then not applied to the hardware and are not validated before application of the DIV.

This function is supported by both COMET and COMET-QUAD devices. For COMET devices, the backplane configuration modes in the DIV should not be H-MVIP or H-MVIP CCS.

Prototype
```
INT4 cometqInit(sCMQ_HNDL deviceHandle, sCMQ_DIV
*pdiv, UINT2 profileNum)
```

Inputs	`deviceHandle`	: device Handle (from `cometqAdd`)
	`pdiv`	: (pointer to) Device Initialization Vector
	`profileNum`	: profile number (ignored if pdiv is NULL)
Outputs	None	
Returns	Success = `CMQ_SUCCESS` Failure = <COMET-QUAD ERROR CODE>	
Valid States	`CMQ_PRESENT`	
Side Effects	Changes the DEVICE state to `CMQ_INACTIVE`	

Resetting a Device: cometqReset

Applies a software reset to the COMET or COMET-QUAD device. Also resets all the DDB contents (except for the user context). This function is typically called before re-initializing the device (via `cometqInit`).

This function is supported by both COMET and COMET-QUAD devices.

Prototype	`INT4 cometqReset(sCMQ_HNDL deviceHandle)`	
Inputs	`deviceHandle`	: device Handle (from `cometqAdd`)
Outputs	None	
Returns	Success = `CMQ_SUCCESS` Failure = <COMET-QUAD ERROR CODE>	
Valid States	`CMQ_PRESENT`, `CMQ_ACTIVE`, `CMQ_INACTIVE`	
Side Effects	Changes the DEVICE state to `CMQ_PRESENT`	

Updating the Configuration of a Device: cometqUpdate

Updates the configuration of the device as well as the Device Data Block (DDB) associated with that device according to the DIV passed by the Application. The only difference between `cometqUpdate` and `cometqInit` is that a soft reset is not applied to the device. In addition, a profile number of zero is not allowed if a DIV is not passed to the function.

Prototype	`INT4 cometqUpdate(sCMQ_HNDL deviceHandle, sCMQ_DIV *pdiv, UINT2 profileNum)`	
Inputs	`deviceHandle`	: device handle (from cometqAdd)
	`pdiv`	: (pointer to) Device Initialization Vector
	`profileNum`	: profile number (only used if pdiv is NULL)

Outputs None

Returns Success = `CMQ_SUCCESS`
Failure = <COMET-QUAD ERROR CODE>

Valid States `CMQ_ACTIVE`, `CMQ_INACTIVE`

Side Effects None

Activating a Device: cometqActivate

This function activates a device by enabling interrupts. If the device was deactivated, the interrupt mask remains as it was prior to deactivation. Interrupts will be re-enabled if the device was in ISR mode.

This function is supported by both COMET and COMET-QUAD.

Prototype `INT4 cometqActivate(sCMQ_HNDL deviceHandle)`

Inputs `deviceHandle` : device Handle (from `cometqAdd`)

Outputs None

Returns Success = `CMQ_SUCCESS`
Failure = <COMET-QUAD ERROR CODE>

Valid States `CMQ_INACTIVE`

Side Effects Changes the DEVICE state to `CMQ_ACTIVE`

Deactivating a Device: cometqDeActivate

Deactivates a device by disabling interrupts. The interrupt mask is retained within the DDB and is restored upon device activation.

This function is supported by both COMET and COMET-QUAD devices.

Prototype `INT4 cometqDeActivate(sCMQ_HNDL deviceHandle)`

Inputs `deviceHandle` : device Handle (from `cometqAdd`)

Outputs None

Returns Success = `CMQ_SUCCESS`
Failure = <COMET-QUAD ERROR CODE>

Valid States `CMQ_ACTIVE`

Side Effects Changes the DEVICE state to `CMQ_INACTIVE`

4.4 Device Read and Write

Reading from Device Registers: cometqRead

This function can be used to read any one of the registers on a specific COMET or COMET-QUAD device by providing the register number. This function derives the actual address location based on the device handle and register number inputs. It then reads the contents of this address location using the system specific macro, `sysCometqRead`. Note that a failure to read returns a zero and any error indication is written to the associated DDB. In the event that the device handle passed to the function is invalid, the corresponding error code is written to the MDB.

This function is supported by both COMET and COMET-QUAD devices and behaves identically for both devices. Register address bounds checking is performed as is appropriate for the specific device.

Prototype `UINT1 cometqRead(sCMQ_HNDL deviceHandle, UINT2 regNum)`

Inputs

`deviceHandle`	: device Handle (from `cometqAdd`)
`regNum`	: register number

Outputs ERROR code written to the DDB. If DDB is invalid, the error code is written to the MDB.

Returns Success = value read
Failure = 0

Valid States `CMQ_PRESENT`, `CMQ_ACTIVE`, `CMQ_INACTIVE`

Side Effects May affect registers that change after a read operation

Writing to Device Registers: cometqWrite

This function can be used to write any one of the registers on a specific COMET or COMET-QUAD device by providing the register number. The function derives the actual address location based on the device handle and register number inputs. It then writes the contents of this address location using the system specific macro, `sysCometqWrite`. Note that a failure to write returns a zero and any error indication is written to the DDB. In the event that the device handle passed to the function is invalid, the corresponding error code is written to the MDB.

This function is supported by both COMET and COMET-QUAD devices and behaves identically for both devices. Register address bounds checking is performed as is appropriate for the specific device.

Prototype `UINT1 cometqWrite(sCMQ_HNDL deviceHandle, UINT2 regNum, UINT1 value)`

Inputs
`deviceHandle` : device Handle (from `cometqAdd`)
`regNum` : register number
`value` : value to be written

Outputs ERROR code written to the DDB. If the DDB is invalid, the error code is written to the MDB.

Returns Success = value written
Failure = 0

Valid States `CMQ_PRESENT`, `CMQ_ACTIVE`, `CMQ_INACTIVE`

Side Effects May change the configuration of the Device

Reading from a block of Device Registers: cometqReadBlock

This function can be used to read a block of registers on a COMET or COMET-QUAD device by providing the starting register number and the size to read. This function derives the actual start address location based on the device handle and starting register number inputs. It then reads the contents of this data block using multiple calls to the system specific macro, `sysCometqRead`. Note that a failure to read returns a zero and any error indication is written to the DDB. In the event that the device handle passed to the function is invalid, the corresponding error code is written to the MDB. It is the USER's responsibility to allocate enough memory for the block read.

This function is supported by both COMET and COMET-QUAD devices and behaves identically for both devices. Register address bounds checking is performed as is appropriate for the specific device.

Prototype `UINT1 cometqReadBlock(sCMQ_HNDL deviceHandle, UINT2 startRegNum, UINT2 size, UINT1 *pblock)`

Inputs
`deviceHandle` : device Handle (from `cometqAdd`)
`startRegNum` : starting register number
`size` : size of the block to read

Outputs ERROR code written to the DDB. If the DDB is invalid, the error code is written to the MDB.

`pblock` : (pointer to) block read memory

Returns Last register value read

Valid States `CMQ_PRESENT`, `CMQ_ACTIVE`, `CMQ_INACTIVE`

Side Effects May affect registers that change after a read operation

Writing to a Block of Device Registers: cometqWriteBlock

This function can be used to write to a block of registers on a COMET or COMET-QUAD device by providing the starting register number, the block size, and the data. This function derives the actual starting address location based on the device handle and starting register number inputs. It then writes the contents of this data block using multiple calls to the system specific macro, `sysCometqWrite`. A bit from the passed block is only modified in the device's registers if the corresponding bit is set in the passed mask. Note that any error indication is written to the DDB. In the event that the device handle passed to the function is invalid, the corresponding error code is written to the MDB.

This function is supported by both COMET and COMET-QUAD devices and behaves identically for both devices. Register address bounds checking is performed as is appropriate for the specific device.

Prototype
```
UINT1 cometqWriteBlock(sCMQ_HNDL deviceHandle, UINT2
startRegNum, UINT2 size, UINT1 *pblock, UINT1 *pmask)
```

Inputs

`deviceHandle`	: device Handle (from `cometqAdd`)
`startRegNum`	: starting register number
`size`	: size of block to read
`pblock`	: (pointer to) block to write
`pmask`	: (pointer to) mask

Outputs ERROR code written to the DDB. If the DDB is invalid, the error code is written to the MDB.

Returns Last register value written

Valid States `CMQ_PRESENT`, `CMQ_ACTIVE`, `CMQ_INACTIVE`

Side Effects May change the configuration of the Device

Reading from Framer Device Registers: cometqReadFr

This function can be used to read any one of the indirect registers on a specific COMET-QUAD device by providing the register number. This function derives the actual address location based on the device handle and register number inputs. It then reads the contents of this address location using the system specific macro, `sysCometqRead`. Note that a failure to read returns a zero and any error indication is written to the associated DDB. In the event that the device handle passed to the function is invalid, the corresponding error code is written to the MDB.

This function is not supported by COMET devices as it contains only a single framer. Invocations of this API for a COMET device will fail with error code `CMQ_ERR_INVALID_DEV`.

Prototype
```
UINT1 cometqReadFr(sCMQ_HNDL deviceHandle, UINT1
frmNum, UINT2 regNum)
```

Inputs

`deviceHandle`	: device Handle (from `cometqAdd`)

	`frmNum`	: framer number: 0, 1, 2, or 3
	`regNum`	: register number

Outputs ERROR code written to the DDB

Returns Success = value read
Failure = 0

Valid States `CMQ_PRESENT`, `CMQ_ACTIVE`, `CMQ_INACTIVE`

Side Effects May affect registers that change after a read operation

Writing to Framer Device Registers: cometqWriteFr

This function can be used to write to any one of the indirect registers on a specific COMET-QUAD device by providing the register number. This function derives the actual address location based on the device handle and register number inputs. It then writes the contents of this address location using the system specific macro, `sysCometqWrite`. Note that a failure to write returns a zero and any error indication is written to the DDB. In the event that the device handle passed to the function is invalid, the corresponding error code is written to the MDB.

This function is not supported by COMET devices as it provides only a single framer. Invocations of this API for a COMET device will fail with error code `CMQ_ERR_INVALID_DEV`.

Prototype
```
UINT1 cometqWriteFr(sCMQ_HNDL deviceHandle, UINT1
frmNum, UINT2 regNum, UINT1 value)
```

Inputs	`deviceHandle`	: device Handle (from `cometqAdd`)
	`frmNum`	: framer number: 0, 1, 2, or 3
	`regNum`	: register number
	`value`	: value to be written

Outputs ERROR code written to the DDB

Returns Success = value written
Failure = 0

Valid States `CMQ_PRESENT`, `CMQ_ACTIVE`, `CMQ_INACTIVE`

Side Effects May change the configuration of the Device

Reading from Device Indirect Registers: cometqReadFrInd

This function can be used to read any one of the indirect registers on a specific COMET or COMET-QUAD device by providing the indirect address and the indirect memory section. This function derives the actual address location based on the device handle and indirect address inputs. It then reads the contents of this address location using the system specific macro, `sysCometqRead`. Note that a failure to read returns a zero and any error indication is written to the associated DDB. In the event that the device handle passed to the function is invalid, the corresponding error code is written to the MDB.

Note that the RLPS Equalizer RAM indirect registers are accessed through cometqReadRLPS and cometqWriteRLPS.

Prototype `UINT1 cometqReadFrInd(sCMQ_HNDL deviceHandle, UINT1 frmNum, eCMQ_SECTION section, UINT2 regNum)`

Inputs

`deviceHandle`	: device Handle (from `cometqAdd`)
`frmNum`	: framer number: COMET-QUAD: 0, 1, 2, or 3 COMET: not used
`section`	: section number: `CMQ_SIGX_SECT`, `CMQ_TPSC_SECT`, `CMQ_RPSC_SECT`, `CMQ_XLPG_SECT`
`regNum`	: indirect address

Outputs ERROR code written to the DDB

Returns Success = value read
Failure = 0

Valid States `CMQ_PRESENT`, `CMQ_ACTIVE`, `CMQ_INACTIVE`

Side Effects May affect registers that change after a read operation

Writing to Device Indirect Registers: cometqWriteFrInd

This function can be used to write to any one of the indirect registers on a specific COMET or COMET-QUAD device by providing the indirect address and indirect memory section. This function derives the actual address location based on the device handle and indirect address inputs. It then writes the contents of this address location using the system specific macro, `sysCometqWrite`. Note that a failure to write returns a zero and any error indication is written to the DDB. In the event that the device handle passed to the function is invalid, the corresponding error code is written to the MDB.

Note that the RLPS Equalizer RAM indirect registers are accessed through cometqReadRLPS and cometqWriteRLPS.

Prototype `UINT1 cometqWriteFrInd(sCMQ_HNDL deviceHandle, UINT1 frmNum, eCMQ_SECTION section, UINT2 regNum, UINT1 value)`

Inputs

`deviceHandle`	: device Handle (from `cometqAdd`)
`frmNum`	: framer number: COMET-QUAD: 0, 1, 2, or 3 COMET: not used
`section`	: section number: `CMQ_SIGX_SECT`, `CMQ_TPSC_SECT`, `CMQ_RPSC_SECT`, `CMQ_XLPG_SECT`
`regNum`	: indirect address
`value`	: value to be written

Outputs ERROR code written to the DDB

Returns Success = value written
Failure = 0

Valid States `CMQ_PRESENT`, `CMQ_ACTIVE`, `CMQ_INACTIVE`

Side Effects May change the configuration of the Device

Reading from Device RLPS Indirect Registers: cometqReadRLPS

This function can be used to read any one of the RLPS indirect registers on a specific COMET or COMET-QUAD device. This function derives the actual address location based on the device handle and indirect address inputs. It then reads the contents of this address location using the system specific macro, `sysCometqRead`. Note that a failure to read returns a zero and any error indication is written to the associated DDB. In the event that the device handle passed to the function is invalid, the corresponding error code is written to the MDB.

Prototype `UINT4 cometqReadRLPS(sCMQ_HNDL deviceHandle, UINT1 frmNum, UINT1 regNum)`

Inputs

`deviceHandle`	: device Handle (from `cometqAdd`)
`frmNum`	: framer number: COMET-QUAD: 0, 1, 2, or 3 COMET: not used
`regNum`	: indirect address

Outputs ERROR code written to the DDB

Returns Success = value read
Failure = 0

Valid States `CMQ_PRESENT`, `CMQ_ACTIVE`, `CMQ_INACTIVE`

Side Effects May affect registers that change after a read operation

Writing to Device RLPS Indirect Registers: cometqWriteRLPS

This function can be used to write to any one of the RLPS indirect registers on a specific COMET or COMET-QUAD device. This function derives the actual address location based on the device handle and indirect address inputs. It then writes the contents of this address location using the system specific macro, `sysCometqWrite`. Note that a failure to write returns a zero and any error indication is written to the DDB. In the event that the device handle passed to the function is invalid, the corresponding error code is written to the MDB.

Prototype `UINT4 cometqWriteRLPS(sCMQ_HNDL deviceHandle, UINT1 frmNum, UINT1 regNum, UINT4 value)`

Inputs

`deviceHandle`	: device Handle (from `cometqAdd`)
`frmNum`	: framer number: COMET-QUAD: 0, 1, 2, or 3 COMET: not used
`regNum`	: indirect address
`value`	: value to be written

Outputs ERROR code written to the DDB

Returns Success = value written
Failure = 0

Valid States `CMQ_PRESENT`, `CMQ_ACTIVE`, `CMQ_INACTIVE`

Side Effects May change the configuration of the Device

4.5 Interface Configuration

The Interface Configuration section of the driver is used for configuring the COMET and COMET-QUAD T1/E1 line interfaces and the receive and transmit backplane interfaces, including the transmit and receive H-MVIP interfaces on the COMET-QUAD. These APIs allow the user to configure the receive and transmit line coding scheme to B8ZS, HDB3, or AMI, the receive and transmit analog characteristics, jitter attenuators and their associated timing options, and digital and analog loss of signal definitions. For both COMET and COMET-QUAD, the receive and transmit elastic stores can be configured through this interface in addition to backplane interface configuration. The Interface Configuration API does not provide a backplane profile mechanism. In order to configure the backplane based upon any of the standard configurations given in the device data sheets, the profile initialization API or a device initialization vector (DIV) must be used.

Transmit line coding configuration: cometqLineTxEncodeCfg

Allows configuration of the transmit line coding scheme. In T1 operational mode, selection between alternate mark inversion (AMI) and bipolar with eight zero substitution (B8ZS) is allowed. In E1 mode, the user can select either AMI or high density bipolar (HDB3) schemes.

Prototype `INT4 cometqLineTxEncodeCfg(sCMQ_HNDL deviceHandle, UINT2 chan, eCMQ_LINE_CODE encScheme)`

Inputs

`deviceHandle`	: device Handle (from `cometqAdd`)
`chan`	: E1/T1 channel: COMET-QUAD: 0, 1, 2, or 3 COMET: not used
`encScheme`	: transmit encoding scheme for E1/T1. One of: `CMQ_LINE_CODE_AMI`, `CMQ_LINE_CODE_HDB3_E1`, or `CMQ_LINE_CODE_B8ZS_T1`

Outputs None

Returns Success = `CMQ_SUCCESS`
Failure = <COMET-QUAD ERROR CODE>

Valid States `CMQ_ACTIVE`, `CMQ_INACTIVE`

Side Effects None

Receive line coding configuration: cometqLineRxEncodeCfg

Allows configuration of the receive line coding scheme. In T1 operational mode, selection between alternate mark inversion (AMI) and bipolar with eight zero substitution (B8ZS) is allowed. In E1 mode, the user can select either AMI or high density bipolar (HDB3) schemes. This function also allows the user to configure the bipolar violation definition.

Prototype `INT4 cometqLineRxEncodeCfg(sCMQ_HNDL deviceHandle, UINT2 chan, eCMQ_LINE_CODE encScheme, UINT1 incXSZeros, UINT1 E1_O162En)`

Inputs

`deviceHandle`	: device Handle (from `cometqAdd`)
`chan`	: E1/T1 channel: COMET-QUAD: 0, 1, 2, or 3 COMET: not used
`encScheme`	: transmit encoding scheme for E1/T1. One of: `CMQ_LINE_CODE_AMI`, `CMQ_LINE_CODE_HDB3_E1`, or `CMQ_LINE_CODE_B8ZS_T1`
`incXSZeros`	: indication to include excess zero violations as bipolar violations. Excess zero run lengths are 5 bits for T1 AMI, 8 bits in T1 B8ZS, and 4 bits for both E1 schemes
`E1_O162En`	: enable the O.162 bipolar violation definition (E1 only)

Outputs None

Returns Success = `CMQ_SUCCESS`
Failure = <COMET-QUAD ERROR CODE>

Valid States `CMQ_ACTIVE, CMQ_INACTIVE`

Side Effects None

Analog transmitter configuration: cometqLineTxAnalogCfg

Allows configuration of the transmit analog interface including the waveform scale factor, the waveform pulse shape, transmit tri-state, and fuse programming.

Prototype
```
INT4 cometqLineTxAnalogCfg(sCMQ_HNDL deviceHandle,
UINT2 chan, sCMQ_CFG_TX_ANALOG* ptxAnalogCfg)
```

Inputs
`deviceHandle` : device Handle (from `cometqAdd`)
`chan` : E1/T1 channel: COMET-QUAD: 0, 1, 2, or 3
COMET: not used
`ptxAnalogCfg` : transmit analog configuration structure

Outputs None

Returns Success = `CMQ_SUCCESS`
Failure = <COMET-QUAD ERROR CODE>

Valid States `CMQ_ACTIVE, CMQ_INACTIVE`

Side Effects None

Analog receiver configuration: cometqLineRxAnalogCfg

Configures the analog line receive interface including analog loss of signal thresholds and periods and equalizer feedback frequency and stabilization period. Also, the equalizer RAM can be configured through this function.

Prototype
```
INT4 cometqLineRxAnalogCfg(sCMQ_HNDL deviceHandle,
UINT2 chan, sCMQ_CFG_RX_ANALOG* prxAnalogCfg)
```

Inputs
`deviceHandle` : device Handle (from `cometqAdd`)
`chan` : E1/T1 channel: COMET-QUAD: 0, 1, 2, or 3
COMET: not used
`prxAnalogCfg` : receive analog configuration structure

Outputs None

Returns Success = `CMQ_SUCCESS`
Failure = <COMET-QUAD ERROR CODE>

Valid States `CMQ_ACTIVE, CMQ_INACTIVE`

Side Effects None

Transmit jitter attenuator configuration: cometqLineTxJatCfg

Configures the line transmit interface jitter attenuator by allowing the user to select bypass, clock divisors, and reference clocks.

Prototype
```
INT4 cometqLineTxJatCfg(sCMQ_HNDL deviceHandle, UINT2
chan, sCMQ_CFG_TX_JAT *ptxJatCfg)
```

Inputs
`deviceHandle` : device Handle (from `cometqAdd`)
`chan` : E1/T1 channel: COMET-QUAD: 0, 1, 2, or 3
COMET: not used
`ptxJatCfg` : transmit jitter configuration data

Outputs None

Returns
Success = `CMQ_SUCCESS`
Failure = <COMET-QUAD ERROR CODE>

Valid States `CMQ_ACTIVE, CMQ_INACTIVE`

Side Effects Implicitly sets voltage reference of the analog receiver's equalizer based on the operational mode.

Receive jitter attenuator configuration: cometqLineRxJatCfg

Configures the line receive interface jitter attenuator by allowing the user to select jitter attenuator bypass and clock divisors.

Prototype
```
INT4 cometqLineRxJatCfg(sCMQ_HNDL deviceHandle, UINT2
chan, sCMQ_CFG_RX_JAT *prxJatCfg)
```

Inputs
`deviceHandle` : device Handle (from `cometqAdd`)
`chan` : E1/T1 channel: COMET-QUAD: 0, 1, 2, or 3
COMET: not used
`prxJatCfg` : receive jitter configuration data

Outputs None

Returns
Success = `CMQ_SUCCESS`
Failure = <COMET-QUAD ERROR CODE>

Valid States `CMQ_ACTIVE, CMQ_INACTIVE`

Side Effects None

Clock service unit configuration: cometqLineClkSvcCfg

This API function allows the user to configure the clock service unit operating mode. The user can select between a straight mapping of 2.048 MHz onto 2.048 MHz or 1.544 MHz onto 1.544 MHz as well as the option of transmitting at a 1.544 MHz line rate when the XCLK input is 2.048 MHz.

This function is supported by both COMET and COMET-QUAD devices.

Prototype `INT4 cometqLineClkSvcCfg(sCMQ_HNDL deviceHandle, eCMQ_CSU_SVC_CLK synthTxFreq)`

Inputs
`deviceHandle` : device Handle (from `cometqAdd`)
`synthTxFreq` : synthesis clock frequency and transmit clock clock frequency. Select one of:
`CMQ_XCLK_2048_TXCLK_2048`,
`CMQ_XCLK_1544_TXCLK_1544`, or
`CMQ_XCLK_2048_TXCLK_1544`

Outputs None

Returns
Success = `CMQ_SUCCESS`
Failure = <COMET-QUAD ERROR CODE>

Valid States `CMQ_ACTIVE, CMQ_INACTIVE`

Side Effects None

Receive clock and data recovery options: cometqLineRxClkCfg

Configures the line receive interface clock and data recovery characteristics including the clock and data recovery algorithm and the digital loss of signal threshold. Note that selection of either `CMQ_LOS_THRESH_PCM_10_HDB3` (E1 only), `CMQ_LOS_THRESH_PCM_15_B8ZS` (T1 only), or `CMQ_LOS_THRESH_PCM_15_AMI` as the loss of signal threshold will set the receive line coding scheme to the specified value, overriding the current setting.

Prototype `INT4 cometqLineRxClkCfg(sCMQ_HNDL deviceHandle, UINT2 chan, sCMQ_CFG_RX_CLK *prxClkCfg)`

Inputs
`deviceHandle` : device Handle (from `cometqAdd`)
`chan` : E1/T1 channel: COMET-QUAD: 0, 1, 2, or 3
COMET: not used
`prxClkCfg` : receive clock and data recovery structure

Outputs None

Returns
Success = `CMQ_SUCCESS`
Failure = <COMET-QUAD ERROR CODE>

Valid States `CMQ_ACTIVE, CMQ_INACTIVE`

Side Effects The line coding scheme will change if one of `CMQ_LOS_THRESH_PCM_10_HDB3`, `CMQ_LOS_THRESH_PCM_15_B8ZS`, or `CMQ_LOS_THRESH_PCM_15_AMI` was selected as the loss of signal threshold

Backplane transmit interface configuration: cometqBTIFAccessCfg

This function allows selection between transmit backplane master/slave modes as well as basic backplane data mode configuration.

Prototype
```
INT4 cometqBTIFAccessCfg(sCMQ_HNDL deviceHandle, UINT2
chan, sCMQ_BACKPLANE_ACCESS_CFG *pBTIFCfgData)
```

Inputs
`deviceHandle` : device Handle (from `cometqAdd`)
`chan` : E1/T1 channel: COMET-QUAD: 0, 1, 2, or 3
COMET: not used
`pBTIFCfgData` : backplane transmit interface configuration structure

Outputs None

Returns Success = `CMQ_SUCCESS`
Failure = <COMET-QUAD ERROR CODE>

Valid States `CMQ_ACTIVE, CMQ_INACTIVE`

Side Effects None

Backplane transmit interface configuration: cometqBTIFFrmCfg

This function allows configuration of the transmit backplane interface frame pulse, parity odd/even selection and inversion, as well as frame pulse bit offset and T1 ESF alignment.

Prototype
```
INT4 cometqBTIFFrmCfg(sCMQ_HNDL deviceHandle, UINT2
chan, sCMQ_CFG_BTIF_FRM* pfrmCfg)
```

Inputs
`deviceHandle` : device Handle (from `cometqAdd`)
`chan` : E1/T1 channel: COMET-QUAD: 0, 1, 2, or 3
COMET: not used
`pfrmCfg` : backplane transmit interface framing structure

Outputs None

Returns Success = `CMQ_SUCCESS`
Failure = <COMET-QUAD ERROR CODE>

Valid States `CMQ_ACTIVE, CMQ_INACTIVE`

Side Effects None

Backplane receive interface configuration: cometqBRIFAccessCfg

This function allows selection between receive backplane master/slave modes as well as basic backplane data mode configuration.

Prototype `INT4 cometqBRIFAccessCfg(sCMQ_HNDL deviceHandle, UINT2 chan, sCMQ_BACKPLANE_ACCESS_CFG* pBRIFCfgData)`

Inputs
`deviceHandle` : device Handle (from `cometqAdd`)
`chan` : E1/T1 channel: COMET-QUAD: 0, 1, 2, or 3
COMET: not used
`pBRIFCfgData` : backplane receive interface configuration structure

Outputs None

Returns
Success = `CMQ_SUCCESS`
Failure = <COMET-QUAD ERROR CODE>

Valid States `CMQ_ACTIVE, CMQ_INACTIVE`

Side Effects None

Backplane receive interface configuration: cometqBRIFFrmCfg

The user is able to configure the backplane receive interface frame pulse mode, parity configuration, frame pulse bit offset, bit fixing, T1 ESF insertion, and timeslot mapping through the use of this function.

Prototype `INT4 cometqBRIFFrmCfg(sCMQ_HNDL deviceHandle, UINT2 chan, sCMQ_CFG_BRIF_FRM *pfrmCfg)`

Inputs
`deviceHandle` : device Handle (from `cometqAdd`)
`chan` : E1/T1 channel: COMET-QUAD: 0, 1, 2, or 3
COMET: not used
`pfrmCfg` : backplane receive interface framing structure

Outputs None

Returns
Success = `CMQ_SUCCESS`
Failure = <COMET-QUAD ERROR CODE>

Valid States `CMQ_ACTIVE, CMQ_INACTIVE`

Side Effects None

Receive and transmit HMVIP interfaces configuration: cometqHMVIPCfg

This function provides configuration of the H-MVIP receive and transmit interfaces for COMET-QUAD devices.

This function is not supported by COMET devices.

Prototype `INT4 cometqHMVIPCfg(sCMQ_HNDL deviceHandle, sCMQ_CFG_HMVIP *pHMVIPCfg)`

Inputs
`deviceHandle` : device Handle (from `cometqAdd`)
`pHMVIPCfg` : H-MVIP interface configuration structure

Outputs None

Returns
Success = `CMQ_SUCCESS`
Failure = <COMET-QUAD ERROR CODE>

Valid States `CMQ_ACTIVE, CMQ_INACTIVE`

Side Effects None

Receive elastic store configuration: cometqRxElstStCfg

This function provides configuration of the elastic store in the receive data path. The user can select whether or not to bypass the receive elastic store and configure the idle codes for both the PCM data stream (COMET and COMET-QUAD) and the CCS stream (COMET-QUAD only).

Prototype `INT4 cometqRxElstStCfg(sCMQ_HNDL deviceHandle, UINT2 chan, sCMQ_CFG_RX_ELST* pElstCfg)`

Inputs
`deviceHandle` : device Handle (from `cometqAdd`)
`chan` : E1/T1 channel: COMET-QUAD: 0, 1, 2, or 3
COMET: not used
`pElstCfg` : elastic store configuration data

Outputs None

Returns
Success = `CMQ_SUCCESS`
Failure = <COMET-QUAD ERROR CODE>

Valid States `CMQ_ACTIVE, CMQ_INACTIVE`

Side Effects None

Transmit elastic store configuration: cometqTxElstStCfg

This function configures the elastic store in the transmit data path. The user can select whether or not to bypass the transmit elastic store.

Prototype	`INT4 cometqTxElstStCfg(sCMQ_HNDL deviceHandle, UINT2 chan, UINT1 elstEnable)`
Inputs	`deviceHandle` : device Handle (from `cometqAdd`) `chan` : E1/T1 channel: COMET-QUAD: 0, 1, 2, or 3 COMET: not used `elstEnable` : enable the elastic store or force bypass
Outputs	None
Returns	Success = `CMQ_SUCCESS` Failure = <COMET-QUAD ERROR CODE>
Valid States	`CMQ_ACTIVE, CMQ_INACTIVE`
Side Effects	None

4.6 T1 /E1 Framers

This section of the driver configures and monitors the T1/E1 Framers. The T1 framers can be configured to detect and transmit ESF, SF, J1 and the other T1 variants supported by the COMET and COMET-QUAD. The E1 framer can be configured to detect and transmit basic frame alignment or CRC-4multiframe with additional criteria to generate/detect channel associative signaling.

Set Device Operational Mode: cometqSetOperatingMode

This function specifies whether the device will operate in T1 or E1 mode. If `cometqInit` was called and hardware initialization was not specified, this function must be called immediately after device initialization. The behavior of subsequent calls to the configuration APIs throughout the driver assumes correct operating mode configuration. Configuration via any of the APIs throughout the driver may not be correct if `cometqSetOperatingMode` is not executed first.

The following device configuration is performed based on the operating mode:

- The RLPS voltage reference is configured as specified on the data sheet for the current device operation mode. Note that these values differ for the COMET and COMET-QUAD and are programmed accordingly
- RX-ELST Configuration register is configured to reflect the operating mode on both COMET and COMET-QUAD devices
- RX-ELST CCS Configuration register is configured to the reflect operating mode on the COMET-QUAD only
- TX-ELST Configuration register is configured to reflect the operating mode on both COMET and COMET-QUAD devices

- TX-ELST CCS Configuration register is configured to the reflect operating mode on the COMET-QUAD only

For COMET-QUAD devices, all four framers operate in the same operational mode as it is a global device value.

Prototype `INT4 cometqSetOperatingMode(sCMQ_HNDL deviceHandle, eCMQ_OPER_MODE operMode)`

Inputs
`deviceHandle` : device Handle (from `cometqAdd`)
`operMode` : Selects either T1 or E1 mode: `CMQ_MODE_E1`, `CMQ_MODE_T1`

Outputs None

Returns Success = `CMQ_SUCCESS`
Failure = <COMET-QUAD ERROR CODE>

Valid States `CMQ_INACTIVE`

Side Effects RLPS Equalizer Loop Voltage Reference, RX-ELST Configuration, TX-ELST Configuration, RX-ELST CCS Configuration (COMET-QUAD), and TX-ELST CCS Configuration (COMET-QUAD) registers configured to reflect operating mode

T1 transmit framer configuration: cometqT1TxFramerCfg

This function configures the transmit framing format and the zero code suppression format. Also, the user can enable the signal aligner block (SIGA) between the backplane and the transmit framer.

Prototype `INT4 cometqT1TxFramerCfg (sCMQ_HNDL deviceHandle, UINT2 chan, sCMQ_CFG_T1TX_FRM *pfrmCfg)`

Inputs
`deviceHandle` : device Handle (from `cometqAdd`)
`chan` : T1 channel: COMET-QUAD: 0, 1, 2, or 3
COMET: not used
`pfrmCfg` : T1 transmit framer configuration structure

Outputs None

Returns Success = `CMQ_SUCCESS`
Failure = <COMET-QUAD ERROR CODE>

Valid States `CMQ_ACTIVE, CMQ_INACTIVE`

Side Effects None

T1 receive framer configuration: cometqT1RxFramerCfg

This function provides the interface to configure the receive framing format and out of frame criteria. Also, if the frame format is ESF, the user can select the framing algorithm. Selection between use of frame alignment (COFA) event counts or out of frame alignment (OOFA) event counts in the PMON block can be configured here as well.

Based on the framing format specified in `pfrmCfg`, the alarm integrator (ALMI) is configured for the T1 framing format and the signaling extractor (SIGX) block is configured for ESF if ESF is the selected framing format.

Prototype
```
INT4 cometqT1RxFramerCfg (sCMQ_HNDL deviceHandle,
UINT2 chan, sCMQ_CFG_T1RX_FRM *pfrmCfg)
```

Inputs
`deviceHandle` : device Handle (from `cometqAdd`)
`chan` : T1 channel: COMET-QUAD: 0, 1, 2, or 3
COMET: param not used
`pfrmCfg` : T1 receive framer configuration

Outputs `None`

Returns
Success = `CMQ_SUCCESS`
Failure = <COMET-QUAD ERROR CODE>

Valid States `CMQ_ACTIVE, CMQ_INACTIVE`

Side Effects Framing format is configured in SIGX and ALMI device blocks

E1 transmit framer configuration: cometqE1TxFramerCfg

This function provides an interface to configure the E1 transmit framing characteristics. The user is able to specify whether or not to generate CRC-4 multiframes and the type of signaling, if any, to insert into timeslot 16. Also, the user can enable or disable the insertion of national bits, international bits, extra bits, and the FEBE indicator bits through this API.

Prototype
```
INT4 cometqE1TxFramerCfg (sCMQ_HNDL deviceHandle,
UINT2 chan, sCMQ_CFG_E1TX_FRM *pfrmCfg)
```

Inputs
`deviceHandle` : device Handle (from `cometqAdd`)
`chan` : E1 channel: COMET-QUAD: 0, 1, 2, or 3
COMET: not used
`pfrmCfg` : E1 transmit framer configuration structure

Outputs None

Returns
Success = `CMQ_SUCCESS`
Failure = <COMET-QUAD ERROR CODE>

Valid States `CMQ_ACTIVE, CMQ_INACTIVE`

Side Effects None

E1 receive framer configuration: cometqE1RxFramerCfg

This API allows the user to configure the receive framing format and specify whether or not to align to signaling multiframes. In addition to basic framing and signaling configuration, the user can specify the criteria for AIS, RAI, framing errors, and loss of frame detection.

Prototype
```
INT4 cometqE1RxFramerCfg (sCMQ_HNDL deviceHandle,
UINT2 chan, sCMQ_CFG_E1RX_FRM *pfrmCfg)
```

Inputs
`deviceHandle` : device handle (from `cometqAdd`)
`chan` : E1 channel: COMET-QUAD: 0, 1, 2, or 3
COMET: not used
`pfrmCfg` : E1 receive framer configuration

Outputs None

Returns
Success = `CMQ_SUCCESS`
Failure = <COMET-QUAD ERROR CODE>

Valid States `CMQ_ACTIVE, CMQ_INACTIVE`

Side Effects None

E1 transmit framer extra bits insertion: cometqE1TxSetExtraBits

This API allows the user to set the extra bit values that will be inserted into bits 5, 7, and 8 of timeslot 16 in the first frame of every signaling multiframe. Note that these bits will only be inserted if the user has enabled extra bit insertion in the E1 transmit framer configuration.

Prototype
```
INT4 cometqE1TxSetExtraBits(sCMQ_HNDL deviceHandle,
UINT2 chan, UINT1 extraBits)
```

Inputs
`deviceHandle` : device handle (from `cometqAdd`)
`chan` : E1 channel: COMET-QUAD: 0, 1, 2, or 3
COMET: not used
`extraBits` : bitmap containing extra bits information
bit 0: X1 (timeslot 16, bit 5)
bit 1: X3 (timeslot 16, bit 7)
bit 2: X4 (timeslot 16, bit 8)
bits 3-7: unused

Outputs None

Returns
Success = `CMQ_SUCCESS`
Failure = <COMET-QUAD ERROR CODE>

Valid States CMQ_ACTIVE, CMQ_INACTIVE

Side Effects None

E1 transmit framer international bits configuration: cometqE1TxSetIntBits

This API provides the user with an interface to set the value of the international bits to insert into the E1 stream. The user can specify two bits, one for FAS frames and the other for NFAS frames. Note that insertion of the international bits must already be enabled in the transmit framer.

Prototype
```
INT4 cometqE1TxSetIntBits(sCMQ_HNDL deviceHandle,
UINT2 chan, UINT1 intBits)
```

Inputs
deviceHandle : device handle (from cometqAdd)
chan : E1 channel: COMET-QUAD: 0, 1, 2, or 3
COMET: not used
intBits : bitmap containing international bits
bit 0 : S_i[0]: inserted into NFAS frames
bit 1: S_i[1]: inserted into FAS frames
bit 2-7 : unused

Outputs None

Returns
Success = CMQ_SUCCESS
Failure = <COMET-QUAD ERROR CODE>

Valid States CMQ_ACTIVE, CMQ_INACTIVE

Side Effects None

E1 transmit framer national bits configuration: cometqE1TxSetNatBits

This function allows the user to set the national bit codeword for each of the five national bits in timeslot 0 of an NFAS frame. The codeword consists of four bits, one for each NFAS frame in a submultiframe. The user can enable or disable insertion of each of the four bits within the codeword. National bit insertion must be enabled in the E1 transmit framer for transmission of the national bits.

Prototype
```
INT4 cometqE1TxSetNatBits(sCMQ_HNDL deviceHandle,
UINT2 chan, eCMQ_E1_NAT_BIT codeSelect, UINT1 natBits,
UINT1 natBitsEn)
```

Inputs
deviceHandle : device handle (from cometqAdd)
chan : E1 channel: COMET-QUAD: 0, 1, 2, or 3
COMET: not used
codeSelect : selects the national bit codeword to set:
CMQ_E1_NAT_BIT_SA4,
CMQ_E1_NAT_BIT_SA5,

```
                                CMQ_E1_NAT_BIT_SA6,
                                CMQ_E1_NAT_BIT_SA7,
                                CMQ_E1_NAT_BIT_SA8
```

`natBits` : bitmap containing national bit values (over one sub-multiframe)
- bit 0 : first Sa_i position in SMF
- bit 1 : second Sa_i position in SMF
- bit 2 : third Sa_i position in SMF
- bit 3 : fourth Sa_i position in SMF
- bits 4-7 : unused

`natBitsEn` : enables/disables each bit position in `natBits`:
- bit 0 : enable first Sa_i bit
- bit 1 : enable second Sa_i bit
- bit 2 : enable third Sa_i bit
- bit 3 : enable fourth Sa_i bit
- bits 4-7 : unused

Outputs None

Returns Success = `CMQ_SUCCESS`
Failure = <COMET-QUAD ERROR CODE>

Valid States `CMQ_ACTIVE, CMQ_INACTIVE`

Side Effects None

E1 receive framer extra bit extraction: cometqE1RxGetExtraBits

This API provides the value of the extra bits and the y bit from timeslot 16, frame 0 of the last received signaling multiframe.

Prototype
```
INT4 cometqE1RxGetExtraBits(sCMQ_HNDL deviceHandle,
UINT2 chan, UINT1* pExtraBits)
```

Inputs
`deviceHandle` : device handle (from `cometqAdd`)
`chan` : E1 channel: COMET-QUAD: 0, 1, 2, or 3
COMET: not used

Outputs `pExtraBits` : bitmap containing extra bits & y bit
- bit 0 : X1 (from timeslot 16, bit 5)
- bit 1 : Y bit (from timeslot 16, bit 6)
- bit 2 : X3 (from timeslot 16, bit 7)
- bit 3 : X4 (from timeslot 16, bit 8)
- bits 4-7 : unused

Returns Success = `CMQ_SUCCESS`

Failure = <COMET-QUAD ERROR CODE>

Valid States `CMQ_ACTIVE, CMQ_INACTIVE`

Side Effects None

E1 receive framer international bit extraction: cometqE1RxGetIntBits

This function returns the international bits from the incoming E1 stream for the last frame. $S_i[0]$ is updated every NFAS frame while $S_i[1]$ is updated every FAS frame.

Prototype
```
INT4 cometqE1RxGetIntBits(sCMQ_HNDL deviceHandle,
UINT2 chan, UINT1* pIntBits)
```

Inputs
`deviceHandle` : device handle (from `cometqAdd`)
`chan` : E1 channel: COMET-QUAD: 0, 1, 2, or 3
COMET: not used

Outputs
`pIntBits` : bitmap containing international bits
bit 0 : $S_i[0]$: inserted into NFAS frames
bit 1 : $S_i[1]$: inserted into FAS frames
bit 2-7: unused

Returns
Success = `CMQ_SUCCESS`
Failure = <COMET-QUAD ERROR CODE>

Valid States `CMQ_ACTIVE, CMQ_INACTIVE`

Side Effects None

E1 receive framer national bit extraction: cometqE1RxGetNatBitsNFAS

This function returns the value of the national bits from the incoming E1 stream from the last NFAS frame. This API allows the user to process the national bits on a frame by frame basis, without waiting for the complete submultiframe.

Prototype
```
INT4 cometqE1RxGetNatBitsNFAS(sCMQ_HNDL deviceHandle,
UINT2 chan, UINT1* pNatBits)
```

Inputs
`deviceHandle` : device handle (from `cometqAdd`)
`chan` : E1 channel: COMET-QUAD: 0, 1, 2, or 3
COMET: not used

Outputs
`pNatBits` : bitmap containing national bit values
bit 0 : Sa 4
bit 1 : Sa 5
bit 2 : Sa 6
bit 3 : Sa 7
bit 4 : Sa 8

bits 5-7 : unused

Returns Success = `CMQ_SUCCESS`
Failure = <COMET-QUAD ERROR CODE>

Valid States `CMQ_ACTIVE, CMQ_INACTIVE`

Side Effects None

E1 receive framer national bit extraction: cometqE1RxGetNatBitsSMFRM

This function returns a complete national bit codeword for a specified national bit in the incoming E1 stream for the last submultiframe.

Prototype
```
INT4 cometqE1RxGetNatBitsSMFRM(sCMQ_HNDL deviceHandle,
UINT2 chan, eCMQ_E1_NAT_BIT codeSelect, UINT1*
pNatBits)
```

Inputs
`deviceHandle` : device handle (from `cometqAdd`)
`chan` : E1 channel: COMET-QUAD: 0, 1, 2, or 3
COMET: not used

`codeSelect` : selects the national bit codeword to extract:
`CMQ_E1_NAT_BIT_SA4`,
`CMQ_E1_NAT_BIT_SA5`,
`CMQ_E1_NAT_BIT_SA6`,
`CMQ_E1_NAT_BIT_SA7`,
`CMQ_E1_NAT_BIT_SA8`

Outputs
`pNatBits` : bitmap containing national bit values (over one sub-multiframe)
bit 0 : first Sa_i position in SMF
bit 1 : second Sa_i position in SMF
bit 2 : third Sa_i position in SMF
bit 3 : fourth Sa_i position in SMF
bits 4-7 : unused

Returns Success = `CMQ_SUCCESS`
Failure = <COMET-QUAD ERROR CODE>

Valid States `CMQ_ACTIVE, CMQ_INACTIVE`

Side Effects None

4.7 Signal Insertion/Extraction

This section of the driver provides the framework to examine channel associative signaling in the receive T1 or E1 stream and to detect a change of signaling state event. Also an interface to manipulate the data stream on a DS0 basis through the SIGX block on the device is provided.

Change of signaling state detection: cometqExtractCOSS

This function provides a bitmap corresponding to change of signaling state (COSS) information for a T1 or E1 stream. After determining on which E1 timeslot or T1 channel the signaling state has changed through the use of this function, the user should call `cometqSigExtract` for the new signaling state.

Prototype `INT4 cometqExtractCOSS(sCMQ_HNDL deviceHandle, UINT2 chan, UINT4 *psigState)`

Inputs
`deviceHandle` : device Handle (from `cometqAdd`)
`chan` : E1/T1 channel: COMET-QUAD: 0, 1, 2, or 3
COMET: not used

Outputs `pSigState` : Signal state bit map.

E1: Bit 0 corresponds to timeslot 1 and bit 30 corresponds to timeslot 31. Timeslot 0 and 16 do not have COSS info. Bit 15 (timeslot 16) is always set to 0. Bit 31 is unused and set to zero.

T1: Bits 0 to 23 correspond to timeslots 1 to 24 respectively. Bits 24 to 31 are not used and set to 0.

Returns
Success = `CMQ_SUCCESS`
Failure = <COMET-QUAD ERROR CODE>

Valid States `CMQ_ACTIVE, CMQ_INACTIVE`

Side Effects None

Signaling state extraction: cometqSigExtract

This API extracts signaling bits for a single E1 timeslot or T1 channel from the receive stream. Note that upon a COSS event, these values are not updated until the next signaling multiframe.

Prototype `INT4 cometqSigExtract(sCMQ_HNDL deviceHandle, UINT2 chan, UINT1 timeslot, UINT1* pSigState)`

Inputs

`deviceHandle` : device Handle (from `cometqAdd`)

`chan` : E1/T1 channel: COMET-QUAD: 0, 1, 2, or 3
COMET: not used

`timeslot` : timeslot for which to retrieve signaling. For T1, this value should be from 1-24. When in E1 mode, this value should range from 1-31 as there is no signaling info for timeslot 0.

Outputs

`pSigState` : contains the signaling state information for the timeslot.
bit 0: D bit
bit 1: C bit
bit 2: B bit
bit 3: A bit
bits 4-7: unused

Returns

Success = `CMQ_SUCCESS`
Failure = <COMET-QUAD ERROR CODE>

Valid States

`CMQ_ACTIVE, CMQ_INACTIVE`

Side Effects

None

Signal trunken: cometqSigTslotTrnkDataCfg

This API allows the user to enable change of signal state debouncing by only allowing a COSS event to be generated when the new signaling data is received twice consecutively. Also, DS0 manipulation of the PCM data stream can be performed here. DS0 PCM data manipulation performed by the signal extraction block occurs before PCM data manipulation performed by the RPSC block (via `cometqRPSCPCMCtrl`).

Prototype

```
INT4 cometqSigTslotTrnkDataCfg(sCMQ_HNDL deviceHandle,
UINT2 chan, UINT2 readWrite, UINT1 timeslot, UINT1*
pSigConfig)
```

Inputs

`deviceHandle` : device Handle (from `cometqAdd`)

`chan` : E1/T1 channel: COMET-QUAD: 0, 1, 2, or 3
COMET: not used

`readWrite` : Read data = 0
: Write data = 1

`timeslot` : timeslot to operate on. For T1, this value should be from 1-24. When in E1 mode, this value should range from 0-31.

`pSigConfig` : pointer to config data when writing

Outputs `pSigConfig` : pointer to config data

bit map for T1 mode:

bit 0: enable COSS debouncing
bit 1: logic level when bit fixing enabled
bit 2: enable bit fixing
bit 3: invert all PCM data bits
bit 4-7: unused

bit map for E1 mode:

bit 0: enable COSS debouncing
bit 1: unused
bit 2,3:
0,0 – no inversion
0,1 – invert odd PCM bits
1,0 – invert even PCM bits
1,1 – invert all bits
bit 4-7: unused

Returns Success = `CMQ_SUCCESS`
Failure = <COMET-QUAD ERROR CODE>

Valid States `CMQ_ACTIVE, CMQ_INACTIVE`

Side Effects None

4.8 Alarm and Inband Communications

This section of the driver provides an interface to configure alarms, configure and use the HDLC transmitter and receiver, and transmit and receive bit oriented codes (BOCs). Alarms configuration includes forced insertion of alarms such as AIS and yellow alarms into the transmit stream as well as automatic handling of alarm conditions on the receive line.

Automatic alarm response configuration: cometqAutoAlarmCfg

This function allows the user to enable or disable the possible automatic alarm response on detection of AIS, yellow alarms, red alarms, and out of frame. For an out of frame event, data conditioning can be enabled from the receive elastic stores or the RPSC idle code registers.

Prototype
```
INT4 cometqAutoAlarmCfg(sCMQ_HNDL deviceHandle, UINT2
chan, UINT1 autoYellowEn, UINT1 autoRedEn, UINT1
OOF_RPSCEn, UINT1 OOF_RxELSTEn, UINT1 autoAISEn)
```

Inputs `deviceHandle` : device Handle (from `cometqAdd`)

`chan` : E1/T1 channel: COMET-QUAD: 0, 1, 2, or 3
COMET: not used

`autoYellowEn` : enables automatic generation of yellow (E1 RAI) alarms in the receive direction upon a red alarm

`autoRedEn` : enables automatic trunk conditioning onto the backplane data and signaling streams from the RPSC upon a red carrier fail alarm condition

`OOF_RPSCEn` : enables automatic trunk conditioning onto the backplane data stream for the duration of out of frame. The conditioning data is inserted from the RPSC registers

`OOF_RxELSTEn` : enables automatic trunk conditioning onto the backplane data stream for the duration of out of frame. The conditioning data is inserted from the Rx Elastic store idle code registers

`autoAISEn` : enables automatic insertion of AIS into the receive path upon loss of signal

Outputs None

Returns Success = `CMQ_SUCCESS`
Failure = <COMET-QUAD ERROR CODE>

Valid States `CMQ_ACTIVE, CMQ_INACTIVE`

Side Effects None

Alarm insertion: cometqInsertAlarm

This function allows the user to insert or to disable insertion of alarms into the data stream. The user can insert AIS into the receive stream (onto the backplane) or transmit yellow and AIS alarms for both T1 and E1. In addition, for E1 data, timeslot 16 y bit alarms and timeslot 16 AIS can also be inserted or disabled.

Prototype

```
INT4 cometqInsertAlarm(sCMQ_HNDL deviceHandle, UINT2
chan, eCMQ_ALARM_INS alarmType, UINT1 enable)
```

Inputs

`deviceHandle` : device Handle (from `cometqAdd`)

`chan` : E1/T1 channel: COMET-QUAD: 0, 1, 2, or 3
COMET: not used

`alarmType` : type of alarm to activate/deactivate:

`CMQ_ALARM_INS_RX_AIS` - force AIS into receive backplane interface

`CMQ_ALARM_INS_TX_YELLOW` - transmit yellow alarm (RAI for E1)

`CMQ_ALARM_INS_TX_AIS` - force AIS into transmit stream

`CMQ_ALARM_INS_TX_E1_Y_BIT` - E1 only. Sends the timeslot 16 Y-bit alarm

CMQ_ALARM_INS_TX_E1_TS16_AIS - E1 only. Transmits AIS in timeslot 16

`enable` : selects activation or deactivation of the specified alarm

Outputs None

Returns Success = `CMQ_SUCCESS`
Failure = <COMET-QUAD ERROR CODE>

Valid States `CMQ_ACTIVE, CMQ_INACTIVE`

Side Effects None

HDLC configuration: cometqHDLCEnable

Enables or disables an HDLC link by enabling its respective input clocks and the associated TDPR and RDLC blocks.

Prototype
```
INT4 cometqHDLCEnable(sCMQ_HNDL deviceHandle, UINT2
idHDLC, UINT2 enable)
```

Inputs
`deviceHandle` : device Handle (from `cometqAdd`)
`idHDLC` : HDLC controller: COMET-QUAD: 0, 1, 2, or 3
COMET: 0, 1, or 2

`enable` : enable HDLC controller if set

Outputs None

Returns Success = `CMQ_SUCCESS`
Failure = <COMET-QUAD ERROR CODE>

Valid States `CMQ_ACTIVE, CMQ_INACTIVE`

Side Effects None

HDLC configuration: cometqHDLCRxCfg

Configures the receive HDLC controller for a data link on a COMET or COMET-QUAD device. This function allows you to configure where the receive data link is extracted from. It also enables address matching and masking.

Prototype
```
INT4 cometqHDLCRxCfg(sCMQ_HNDL deviceHandle, UINT2
idHDLC, sCMQ_CFG_HDLC_RX* pData)
```

Inputs
`deviceHandle` : device Handle (from `cometqAdd`)
`idHDLC` : HDLC controller: COMET-QUAD: 0, 1, 2, or 3
COMET: 0, 1, or 2
`pData` : HDLC receiver configuration data

Outputs None

Returns Success = `CMQ_SUCCESS`
Failure = <COMET-QUAD ERROR CODE>

Valid States `CMQ_ACTIVE, CMQ_INACTIVE`

Side Effects None

HDLC configuration: cometqHDLCTxCfg

Configures the HDLC transmitter for a data link on a COMET or COMET-QUAD device. This function allows configuration of where the data link is inserted as well as options such as flag sharing, CRC frame check sequences, and performance report insertion.

Prototype
```
INT4 cometqHDLCTxCfg(sCMQ_HNDL deviceHandle, UINT2
idHDLC, sCMQ_CFG_HDLC_TX* pData)
```

Inputs
`deviceHandle` : device Handle (from `cometqAdd`)
`idHDLC` : HDLC controller: COMET-QUAD: 0, 1, 2, or 3
COMET: 0, 1, or 2
pData : HDLC transmitter configuration data

Outputs None

Returns Success = `CMQ_SUCCESS`
Failure = <COMET-QUAD ERROR CODE>

Valid States `CMQ_ACTIVE, CMQ_INACTIVE`

Side Effects None

HDLC transmitter: cometqTDPRData

This function transmits a data byte on the specified HDLC link.

Prototype
```
INT4 cometqTDPRData(sCMQ_HNDL deviceHandle, UINT2
idHDLC, UINT1 value)
```

Inputs
`deviceHandle` : device Handle (from `cometqAdd`)
`idHDLC` : HDLC controller: COMET-QUAD: 0, 1, 2, or 3
COMET: 0, 1, or 2
`value` : byte to transmit on data link

Outputs None

Returns Success = `CMQ_SUCCESS`
Failure = <COMET-QUAD ERROR CODE>

Valid States `CMQ_ACTIVE, CMQ_INACTIVE`

Side Effects None

HDLC transmitter: cometqTDPRCtl

With this function the user can transmit a control byte on an HDLC link or force a FIFO clear.

Prototype
```
INT4 cometqTDPRCtl(sCMQ_HNDL deviceHandle, UINT2
idHDLC, eCMQ_TDPR_ACTION hdlcAction)
```

Inputs
`deviceHandle` : device Handle (from `cometqAdd`)
`idHDLC` : HDLC controller: COMET-QUAD: 0, 1, 2, or 3
COMET: 0, 1, or 2
`hdlcAction` : `CMQ_TDPR_ACTION_ABORT`
- insert HDLC abort code into data link
`CMQ_TDPR_ACTION_END_ABORT`
- end abort code insertion
`CMQ_TDPR_ACTION_EOM`
- send end of message indicator
`CMQ_TDPR_ACTION_FIFOCLR`
- clear transmit fifo

Outputs None

Returns Success = `CMQ_SUCCESS`
Failure = <COMET-QUAD ERROR CODE>

Valid States `CMQ_ACTIVE, CMQ_INACTIVE`

Side Effects None

HDLC transmitter: cometqTDPRFIFOThreshCfg

Configures the upper and lower limits of the HDLC transmitter FIFO for one of the transmit HDLC Controllers.

Prototype
```
INT4 cometqTDPRFIFOThreshCfg(sCMQ_HNDL deviceHandle,
UINT2 idHDLC, UINT1 upFifoThresh, UINT1 lowFifoThresh)
```

Inputs
`deviceHandle` : device Handle (from `cometqAdd`)
`idHDLC` : HDLC controller: COMET-QUAD: 0, 1, 2, or 3
COMET: 0, 1, or
`upFifoThresh` : upper FIFO threshold for auto transmit
valid values are 0 thru 127
`lowFifoThresh`: lower FIFO threshold for LFILL interrupt
valid values are 0 thru 127
Note that the lower threshold must be less than upper threshold unless both are set to 0.

Outputs None

Returns Success = `CMQ_SUCCESS`
Failure = <COMET-QUAD ERROR CODE>

Valid States `CMQ_ACTIVE, CMQ_INACTIVE`

Side Effects None

HDLC transmitter: cometqTDPRTx

This function transmits an HDLC packet on the specified HDLC link. If this function returns `CMQ_ERR_FIFO_UNDERRUN`, a FIFO underrun has occurred and the packet should be retransmitted.

Prototype
```
INT4 cometqTDPRTx(sCMQ_HNDL deviceHandle, UINT2
idHDLC, UINT1* pPacket, UINT2 packetLength)
```

Inputs
`deviceHandle` : device Handle (from `cometqAdd`)
`idHDLC` : HDLC controller: COMET-QUAD: 0, 1, 2, or 3
COMET: 0, 1, or
`pPacket` : Buffer containing packet to transmit
`packetLength` : Number of bytes in packet to transmit.

Outputs None

Returns Success = `CMQ_SUCCESS`
Failure = <COMET-QUAD ERROR CODE>

Valid States `CMQ_ACTIVE`

Side Effects None

HDLC receiver: cometqRDLCTerm

Forces the RDLC to immediately terminate the reception of the current data frame. This function causes the current data frame to terminate and the FIFO buffer to empty. The RDLC then begins searching for a frame delimiting flag.

Prototype
```
INT4 cometqRDLCTerm(sCMQ_HNDL deviceHandle, UINT2
idHDLC)
```

Inputs
`deviceHandle` : device Handle (from `cometqAdd`)
`idHDLC` : HDLC controller: COMET-QUAD: 0, 1, 2, or 3
COMET: 0, 1, or 2

Outputs None

Returns Success = `CMQ_SUCCESS`

Failure = <COMET-QUAD ERROR CODE>

Valid States `CMQ_ACTIVE, CMQ_INACTIVE`

Side Effects None

HDLC receiver: cometqRDLCAddrMatch

This function configures the primary and secondary addresses used with address matching on an HDLC receiver. When address masking is enabled, the lower two bits of each of the primary and secondary addresses are masked during comparison.

Prototype
```
INT4 cometqRDLCAddrMatch(sCMQ_HNDL deviceHandle, UINT2
idHDLC, UINT1 addrPri, UINT1 addrSec)
```

Inputs

`deviceHandle`	: device Handle (from `cometqAdd`)	
`idHDLC`	: HDLC controller:	COMET-QUAD: 0, 1, 2, or 3 COMET: 0, 1, or 2
`addrPri`	: primary address	
`addrSec`	: secondary address	

Outputs None

Returns Success = `CMQ_SUCCESS`
Failure = <COMET-QUAD ERROR CODE>

Valid States `CMQ_ACTIVE, CMQ_INACTIVE`

Side Effects None

HDLC receiver: cometqRDLCFIFOThreshCfg

Configures the fill level threshold for an RDLC FIFO. The fill level threshold is the number of bytes that must be present in the FIFO before an interrupt is generated.

Prototype
```
INT4 cometqRDLCFIFOThreshCfg(sCMQ_HNDL deviceHandle,
UINT2 idHDLC, UINT1 fifoThresh)
```

Inputs

`deviceHandle`	: device Handle (from `cometqAdd`)	
`idHDLC`	: HDLC controller:	COMET-QUAD: 0, 1, 2, or 3 COMET: 0, 1, or 2
`fifoThresh`	: FIFO fill level threshold (7 bit value)	

Outputs None

Returns Success = `CMQ_SUCCESS`
Failure = <COMET-QUAD ERROR CODE>

Valid States CMQ_ACTIVE, CMQ_INACTIVE

Side Effects None

HDLC receiver: cometqRDLCRx

This function services an RDLC interrupt condition and should be called when an RDLC interrupt has occurred or periodically to poll the RDLC on a COMET or COMET-QUAD device to check for activity on the HDLC link.

When this function is invoked, the return code indicates the status of RDLC data processing. cometqRDLCRx should be repeatedly called until CMQ_SUCCESS is returned. The return codes should be interpreted as follows:

- CMQ_SUCCESS : The RDLC FIFO is empty and cometqRDLCRx need not be called again until the next RDLC interrupt event occurs. If pPacketStatus is set to CMQ_HDLC_PACKET_INCOMPLETE, an incomplete packet was written to pPacket. The number of bytes written to the buffer is given by pBytesWritten. The buffer pointer pPacket should be incremented by this number of bytes when the next RDLC interrupt occurs in order for cometqRDLCRx to continue writing the packet from its previous position.

- CMQ_ERR_CHANGE_OF_LINK_STATE : A change of link state event has been detected. The new link state is returned in pLinkStateFlag. cometqRDLCRx should be called again after the change of link state has been processed.

- CMQ_ERR_PACKET_COMPLETE : Indicates that a packet was terminated in pPacket. pPacketStatus specifies whether a CRC error or non-integral byte count error occurred. cometqRDLCRx should be called again.

- CMQ_ERR_FIFO_OVERRUN : The HDLC receiver FIFO has overflowed. Any incomplete packets (status CMQ_HDLC_PACKET_INCOMPLETE) must be discarded and cometqRDLCRx should be called again.

Prototype

```
INT4 cometqRDLCRx(sCMQ_HNDL deviceHandle,
                  UINT2 idHDLC,
                  UINT2* pLinkStateFlag,
                  UINT2* pBytesWritten,
                  UINT1* pPacketStatus,
                  UINT1* pPacket)
```

Inputs

deviceHandle : device Handle (from cometqAdd)

idHDLC : HDLC controller: COMET-QUAD: 0, 1, 2, or 3
COMET: 0, 1, or

pLinkStateFlag : pointer to link state flag

	`pBytesWritten`	: pointer to byte count
	`pPacketStatus`	: pointer to packet status
	`pPacket`	: packet buffer
Outputs	`pLinkStateFlag`	: new link state on a change of link state event 0 => inactive, 1 => active
	`pBytesWritten`	: number of bytes written to `pPacket`
	`pPacketStatus`	: bitmap containing status of packet written to `pPacket`: `CMQ_HDLC_PACKET_OKAY =>` the packet is complete and without errors `CMQ_HDLC_PACKET_CRC_ERR =>` the packet is complete but contains a CRC error `CMQ_HDLC_PACKET_NON_INTEGER_BYTES =>` the packet is complete but contained a non-itegral number of bytes and should be discarded `CMQ_HDLC_PACKET_INCOMPLETE =>` the packet is not yet complete `CMQ_HDLC_PACKET_NONE =>` no data was been written to the buffer
	`pPacket`	: packet buffer

Returns Success = `CMQ_SUCCESS`
Failure = <COMET-QUAD ERROR CODE>

Valid States `CMQ_ACTIVE`

Side Effects None

Inband loopack code detection: cometqIBCDActLpBkCfg

Configures the detection of inband activate loopback code length and pattern. This API is valid only in T1 mode.

Prototype
```
INT4 cometqIBCDActLpBkCfg(sCMQ_HNDL deviceHandle,
UINT2 chan, UINT1 patLen, UINT1 pat)
```

Inputs	`deviceHandle`	: device Handle (from `cometqAdd`)
	`chan`	: T1 channel: COMET-QUAD: 0, 1, 2, or 3 COMET: not used
	`patLen`	: pattern length (5-8)
	`pat`	: activate loopback pattern

Outputs None

Returns Success = `CMQ_SUCCESS`
Failure = <COMET-QUAD ERROR CODE>

Valid States `CMQ_ACTIVE, CMQ_INACTIVE`

Side Effects None

Inband loopack code detection: cometqIBCDDeActLpBkCfg

Configures the detection of inband deactivate loopback code length and pattern. This API is valid only in T1 mode.

Prototype
```
INT4 cometqIBCDDeActLpBkCfg(sCMQ_HNDL deviceHandle,
UINT2 chan, UINT1 patLen, UINT1 pat)
```

Inputs
`deviceHandle` : device Handle (from `cometqAdd`)
`chan` : T1 channel: COMET-QUAD: 0, 1, 2, or 3
COMET: not used
`patLen` : pattern length (5-8)
`pat` : deactivate loopback pattern

Outputs None

Returns Success = `CMQ_SUCCESS`
Failure = <COMET-QUAD ERROR CODE>

Valid States `CMQ_ACTIVE, CMQ_INACTIVE`

Side Effects None

Inband loopack code transmission: cometqIBCDTxCfg

Enables/disables transmission of the inband transmit loopback code and allows configuration of the code length and pattern when activating. When disabling transmission of the loopback code, the parameters `pat` and `patLen` are not used. This API is valid only in T1 mode.

Prototype
```
INT4 cometqIBCDTxCfg(sCMQ_HNDL deviceHandle, UINT2
chan, UINT1 patLen, UINT1 pat, UINT2 enable)
```

Inputs
`deviceHandle` : device Handle (from `cometqAdd`)
`chan` : T1 channel: COMET-QUAD: 0, 1, 2, or 3
COMET: not used

`patLen` : pattern length (5-8). Only used when enabling.
`pat` : loop code pattern. Only used when enabling.
`enable` : enable/disable loopback code transmission

Outputs None

Returns Success = `CMQ_SUCCESS`
Failure = <COMET-QUAD ERROR CODE>

Valid States `CMQ_ACTIVE, CMQ_INACTIVE`

Side Effects None

Bit Oriented Code transmission: cometqBOCTxCfg

Configures the bit oriented code for transmission,. On COMET-QUAD devices, the associated repeat count can also be configured. The value of parameter `boc` is transmitted with the least significant bit leading. To terminate BOC transmission, `boc` should be set to all 1's. Valid only in T1 mode.

Prototype
```
INT4 cometqBOCTxCfg(sCMQ_HNDL deviceHandle, UINT2
chan, UINT1 boc, UINT1 repCount)
```

Inputs
`deviceHandle` : device Handle (from `cometqAdd`)
`chan` : T1 channel: COMET-QUAD: 0, 1, 2, or 3
COMET: not used
`boc` : 6 bit BOC code pattern. Setting this value to all 1's disables BOC transmission
`repCount` : number of consecutive BOC codes to transmit. Valid range is 0 thru 15. Supported by COMET-QUAD only.

Outputs None

Returns Success = `CMQ_SUCCESS`
Failure = <COMET-QUAD ERROR CODE>

Valid States `CMQ_ACTIVE, CMQ_INACTIVE`

Side Effects None

Bit Oriented Code detection: cometqBOCRxCfg

Configures bit oriented code detection criteria by allowing the user to select the threshold that determines whether or not a valid BOC code has been detected. Valid BOC detection criteria are 8 out of 10 matching values (or alternately 4 out of 5 matching values). Valid only in T1 mode.

Prototype
```
INT4 cometqBOCRxCfg(sCMQ_HNDL deviceHandle, UINT2
chan, eCMQ_RBOC_CRITERIA detectCriteria)
```

Inputs
`deviceHandle` : device Handle (from `cometqAdd`)
`chan` : T1 channel: COMET-QUAD: 0, 1, 2, or 3
COMET: not used
`detectCriteria` : `CMQ_RX_BOC_VALID_4OF5`
– 4 out of 5 of the same BOC code

received
CMQ_RX_BOC_VALID_8OF10
– 8 out of 10 of the same BOC code
received

Outputs None

Returns Success = CMQ_SUCCESS
Failure = <COMET-QUAD ERROR CODE>

Valid States CMQ_ACTIVE, CMQ_INACTIVE

Side Effects None

Bit Oriented Code detection: cometqBOCRxGet

This API retrieves the current bit oriented code from the holding registers. Valid only in T1 mode.

Prototype
```
INT4 cometqBOCRxGet(sCMQ_HNDL deviceHandle, UINT2
chan, UINT1 *prxBOC)
```

Inputs
deviceHandle : device Handle (from cometqAdd)
chan : T1 channel: COMET-QUAD: 0, 1, 2, or 3
COMET: not used

Outputs prxBoc : pointer to last BOC received (6 bit value)

Returns Success = CMQ_SUCCESS
Failure = <COMET-QUAD ERROR CODE>

Valid States CMQ_ACTIVE, CMQ_INACTIVE

Side Effects None

4.9 Serial Control

This section provides the user with an interface to manipulate the T1 or E1 stream on a DS0 basis; the user can force signal or trunk conditioning on any given timeslot or manipulate the PCM data. Also, this section provides access to the on-chip pseudo random generator/detector, allowing the user to configure the pattern and insert random data into the data stream.

Transmit per-channel serial controller: cometqTPSCEnable

This function enables or disables the Transmit Per-Channel Serial Controller (TPSC) to manipulate signaling and data for all 24 T1 channels or all 32 E1 timeslots.

Prototype `INT4 cometqTPSCEnable(sCMQ_HNDL deviceHandle, UINT2 chan, UINT2 enable)`

Inputs
`deviceHandle` : device Handle (from `cometqAdd`)
`chan` : E1/T1 channel: COMET-QUAD: 0, 1, 2, or 3
COMET: not used
`enable` : enable/disable Transmit Serial Controller

Outputs None

Returns
Success = `CMQ_SUCCESS`
Failure = <COMET-QUAD ERROR CODE>

Valid States `CMQ_ACTIVE, CMQ_INACTIVE`

Side Effects None

Transmit per-channel serial controller: cometqTPSCPCMCtl

This function allows the user to invoke PCM data inversion, idle pattern insertion, DS0 loopback, and zero code suppression. The user can also force insertion of signaling bits by providing their own signaling data.

Prototype `INT4 cometqTPSCPCMCtl(sCMQ_HNDL deviceHandle, UINT2 chan, UINT1 tSlot, UINT2 rWFlag, UINT1 *pctlByte, UINT1 *ptrnkData, UINT1 *psigData)`

Inputs
`deviceHandle` : device Handle (from `cometqAdd`)
`chan` : E1/T1 channel: COMET-QUAD: 0, 1, 2, or 3
COMET: not used
`tSlot` : timeslot
E1: 0 thru 31
T1: 1 thru 24
`rWFlag` : Read/Write select. Write = 1, Read = 0

`pctlByte` : pcm data control byte
bit 7,5:
0,0 - data unchanged
0,1 - only msb inverted
1,0 - inverted all bits
1,1 - invert all except msb
bit 6: replace pcm data with trunk conditioning byte
bit 4: (T1 only)
replace pcm data with digital mW pattern
bit 3: if receive pattern generation on, data routed to PRBS/PRGD checker otherwise overwritten with PRBS/PRGD test pattern
bit 2: loopback DS0
bit 1,0: zero code suppression format

0,0 - no zero suppression
0,1 - jammed bit 8
1,0 - GTE zero suppression
1,1 - Bell zero suppression

`ptrnkData` : trunk conditioning data byte

`psigData` : signaling data byte / E1 control byte

bit map for E1 mode:

bit 7,6,5: data manipulation
0,0,0 - data unchanged
0,0,1 - invert odd timeslot bits
0,1,0 - invert even timeslot bits
0,1,1 - invert all timeslot bits
1,0,0 - replace data with idle code
1,0,1 - replace data with idle code
1,1,0 - replace data with A-law pattern
1,1,1 - replace data with U-law pattern
bit 4:
when CAS enabled, signaling bits taken from A,B,C,D bits
bit 3,2,1,0:
A,B,C,D bits

bit map for T1 mode:

bit 7: forces signaling data from A,B,C,D bits
bit 6: enables signal insertion from ABCD bits
bit 5,4:
unused
bit 3,2,1,0:
A,B,C,D bits

Outputs

`pctlByte` : control byte
`ptrnkData` : trunk conditioning data byte
`psigData` : signaling data byte

Returns

Success = `CMQ_SUCCESS`
Failure = <COMET-QUAD ERROR CODE>

Valid States `CMQ_ACTIVE, CMQ_INACTIVE`

Side Effects None

Receive per-channel serial controller: cometqRPSCEnable

This function enables or disables the Receive Per-Channel Serial Controller (RPSC) to manipulate signaling and data for all 24 T1 channels or all 32 E1 timeslots.

Prototype `INT4 cometqRPSCEnable(sCMQ_HNDL deviceHandle, UINT2 chan, UINT2 enable)`

Inputs
`deviceHandle` : device Handle (from `cometqAdd`)
`chan` : E1/T1 channel: COMET-QUAD: 0, 1, 2, or 3
COMET: not used
`enable` : enable/disable Receive Serial Controller

Outputs None

Returns Success = `CMQ_SUCCESS`
Failure = <COMET-QUAD ERROR CODE>

Valid States `CMQ_ACTIVE, CMQ_INACTIVE`

Side Effects None

Receive per-channel serial controller: cometqRPSCPCMCtl

Through the use of this function, the user can force data trunk conditioning on any individual timeslot, force insertion of A-law or u-law patterns, or retrieve signaling information. Also, the user can force PCM data inversion, idle pattern insertion, DS0 loopback, and zero code suppression. The user can also force insertion of signaling bits by providing the signaling data to the signaling bits. DS0 PCM data manipulation performed by the RPSC occurs after PCM data manipulation performed in the SIGX block (`cometqSigTslotTrnkDataCfg`).

Prototype `INT4 cometqRPSCPCMCtl(sCMQ_HNDL deviceHandle, UINT2 chan, UINT1 tSlot, UINT2 rWFlag, UINT1 *pctlByte, UINT1 *ptrnkData, UINT1 *psigData)`

Inputs
`deviceHandle` : device Handle (from `cometqAdd`)

`chan` : E1/T1 channel: COMET-QUAD: 0, 1, 2, or 3
COMET: not used

`tSlot` : timeslot
E1: 0 thru 31
T1: 1 thru 24

`rWFlag` : Read/Write select. Write = 1, Read = 0

`pctlByte` : Control byte
bit 7: if receive pattern generation off, data routed to PRBS/PRGD checker otherwise overwritten with PRBS/PRGD test pattern
bit 6: overwrite data with data trunk conditioning code byte
bit 5: overwrite signaling with signaling trunk conditioning code byte
bit 4: replace pcm data with digital miliwat pat

bit 3: selects A-law mW patter instead of U-law
bit 2: invert most significant bit of data
bit 1,0: unused

trnkData : trunk conditioning data byte

sigData : signaling data byte
bit 7,6,5,4: unused
bit 3,2,1,0: A,B,C,D bits

Outputs
`pctlByte` : array of control byte
`trnkData` : trunk conditioning data byte
`sigData` : signaling data byte

Returns
Success = `CMQ_SUCCESS`
Failure = <COMET-QUAD ERROR CODE>

Valid States `CMQ_ACTIVE, CMQ_INACTIVE`

Side Effects None

Transmit Trunk Conditioning: cometqTxTrnkCfg

This function enables or disables trunk conditioning into the transmit stream for all timeslots. PCM data is overwritten by the contents of the TPSC idle code bytes and signaling data is overwritten by the 'ABCD' values in the TPSC signaling control registers.

Prototype
```
INT4 cometqTxTrnkCfg (sCMQ_HNDL deviceHandle, UINT2
chan, UINT2 enable)
```

Inputs
`deviceHandle` : device Handle (from `cometqAdd`)
`chan` : E1/T1 channel: COMET-QUAD: 0, 1, 2, or 3
COMET: not used
`enable` : enable or disable transmit trunk conditioning

Outputs None

Returns
Success = `CMQ_SUCCESS`
Failure = <COMET-QUAD ERROR CODE>

Valid States `CMQ_ACTIVE, CMQ_INACTIVE`

Side Effects None

Receive Trunk Conditioning: cometqRxTrnkCfg

This function allows the user to enable or disable trunk conditioning onto the backplane, overwriting the received data stream. Upon enable, PCM data is overwritten by the contents of the data trunk conditioning registers in the RPSC and signaling data is overwritten by the contents of the signaling trunk conditioning registers in the RPSC.

Prototype `INT4 cometqRxTrnkCfg (sCMQ_HNDL deviceHandle, UINT2 chan, UINT2 enable)`

Inputs
`deviceHandle` : device Handle (from `cometqAdd`)
`chan` : E1/T1 channel: COMET-QUAD: 0, 1, 2, or 3
COMET: not used
`enable` : enable or disable receive trunk conditioning

Outputs None

Returns
Success = `CMQ_SUCCESS`
Failure = <COMET-QUAD ERROR CODE>

Valid States `CMQ_ACTIVE, CMQ_INACTIVE`

Side Effects None

Pattern receive and generation control: cometqPRGDCtlCfg

This function allows the user to configure the pattern generator and detector to insert or detect framed and unframed patterns in both the transmit and receive streams.

Prototype
```
INT4 cometqPRGDCtlCfg(sCMQ_HNDL deviceHandle, UINT2
chan, eCMQ_PRGD_PAT_LEN genLen, eCMQ_PRGD_PAT_LEN
detLen, UINT2 unFrmGen, UINT2 unFrmDet, UINT2
rxPatGenLoc)
```

Inputs
`deviceHandle` : device Handle (from `cometqAdd`)
`chan` : E1/T1channels: COMET-QUAD: 0, 1, 2, or 3
COMET: not used
`genLen` : select 7 or 8 bit pattern insertion:
`CMQ_PRGD_PAT_8_BIT`,
`CMQ_PRGD_PAT_7_BIT`
`detLen` : select 7 or 8 bit pattern detection:
`CMQ_PRGD_PAT_8_BIT`,
`CMQ_PRGD_PAT_7_BIT`
`unFrmGen` : generate unframed pattern if set
`unFrmDet` : detect unframed pattern if set
`rxPatGenLoc` : location of the PRBS generator/detector.
If set, the pattern detector is inserted into the transmit path and the generator is inserted into the receive path.
If clear, the generator is inserted in the transmit path and the detector is inserted in the receive path.

Outputs None

Returns Success = `CMQ_SUCCESS`
Failure = <COMET-QUAD ERROR CODE>

Valid States `CMQ_ACTIVE, CMQ_INACTIVE`

Side Effects None

Pattern receive and generation control: cometqPRGDPatCfg

This API provides configuration of the pattern type that the pseudo-random generator/detector uses. The user can specify the pattern type and select between a pseudo-random or a quasi-random sequence.

The available patterns for the COMET and COMET-QUAD and the corresponding value to assign to the `pat` parameter are as follows:

	Pattern Type	**`pat` enumerated value**
COMET-QUAD	2^{20} - 1	`CMQ_PSEUDO_RANDOM_PAT_2_TO_20th_MINUS1`
	2^{15} - 1	`CMQ_PSEUDO_RANDOM_PAT_2_TO_15th_MINUS1`
	2^{11} - 1	`CMQ_PSEUDO_RANDOM_PAT_2_TO_11th_MINUS1`
COMET	2^{3} - 1	`CMQ_PSEUDO_RANDOM_PAT_2_TO_3RD_MINUS1`
	2^{4} - 1	`CMQ_PSEUDO_RANDOM_PAT_2_TO_4th_MINUS1`
	2^{5} - 1	`CMQ_PSEUDO_RANDOM_PAT_2_TO_5th_MINUS1`
	2^{6} - 1	`CMQ_PSEUDO_RANDOM_PAT_2_TO_6th_MINUS1`
	2^{7} - 1	`CMQ_PSEUDO_RANDOM_PAT_2_TO_7th_MINUS1`
	2^{7} – 1, Fractional T1 loopback activate	`CMQ_PSEUDO_RANDOM_PAT_FRAC_T1_ACTIVATE`

COMET	**Pattern Type**	**`pat` enumerated value**
	$2^9 - 1$, O.153 compliant	`CMQ_PSEUDO_RANDOM_PAT_2_TO_9th_MINUS1_O_153`
	$2^{10} - 1$	`CMQ_PSEUDO_RANDOM_PAT_2_TO_10th_MINUS1`
	$2^{11} - 1$, O.152 compliant	`CMQ_PSEUDO_RANDOM_PAT_2_TO_11th_MINUS1_O_152`
	$2^{15} - 1$, O.151 compliant	`CMQ_PSEUDO_RANDOM_PAT_2_TO_15th_MINUS1_O_151`
	$2^{17} - 1$	`CMQ_PSEUDO_RANDOM_PAT_2_TO_17th_MINUS1`
	$2^{18} - 1$	`CMQ_PSEUDO_RANDOM_PAT_2_TO_18th_MINUS1`
	$2^{20} - 1$, O.153 compliant	`CMQ_PSEUDO_RANDOM_PAT_2_TO_20th_MINUS1_O_153`
	$2^{20} - 1$, O.151 compliant, Quasi-random (`quasiRand = 1`)	`CMQ_PSEUDO_RANDOM_PAT_2_TO_20th_MINUS1_O_151`
	$2^{21} - 1$	`CMQ_PSEUDO_RANDOM_PAT_2_TO_21th_MINUS1`
	$2^{22} - 1$	`CMQ_PSEUDO_RANDOM_PAT_2_TO_22th_MINUS1`
	$2^{23} - 1$, O.151 compliant	`CMQ_PSEUDO_RANDOM_PAT_2_TO_23th_MINUS1_O_151`
	$2^{25} - 1$	`CMQ_PSEUDO_RANDOM_PAT_2_TO_25th_MINUS1`
	$2^{28} - 1$	`CMQ_PSEUDO_RANDOM_PAT_2_TO_28th_MINUS1`
	$2^{29} - 1$	`CMQ_PSEUDO_RANDOM_PAT_2_TO_29th_MINUS1`

COMET	**Pattern Type**	**`pat` enumerated value**
	All ones	`CMQ_PSEUDO_RANDOM_PAT_ALL_ONES`
	All zeroes	`CMQ_PSEUDO_RANDOM_PAT_ALL_ZEROS`
	Alternating 1's and 0's	`CMQ_PSEUDO_RANDOM_PAT_ALT_ONES_AND_ZEROS`
	Double alternating 1's and 0's	`CMQ_PSEUDO_RANDOM_PAT_DOUBLE_ALT_ONES_AND_ZEROS`
	3 1's in 24 bits	`CMQ_PSEUDO_RANDOM_PAT_3_IN_24`
	one 1 in 16 bits	`CMQ_PSEUDO_RANDOM_PAT_1_IN_16`
	one 1 in 8 bits	`CMQ_PSEUDO_RANDOM_PAT_1_IN_8`
	one 1 in 4 bits	`CMQ_PSEUDO_RANDOM_PAT_1_IN_4`
	Inband Activate loopback pattern	`CMQ_PSEUDO_RANDOM_PAT_INBAND_LOOPBACK_ACTIVATE`
	Inband Deactivate loopback pattern	`CMQ_PSEUDO_RANDOM_PAT_INBAND_LOOPBACK_DEACTIVATE`

Prototype `INT4 cometqPRGDPatCfg(sCMQ_HNDL deviceHandle, UINT2 chan, UINT1 quasiRand, eCMQ_PSEUDO_RANDOM_PATTERN pat)`

Inputs

`deviceHandle` : device Handle (from `cometqAdd`)

`chan` : E1/T1channel: COMET-QUAD: 0, 1, 2, or 3
COMET: not used

`quasiRand` : Select quasi-random data instead of pseudo-random.

`pat` : Pattern type as appropriate for COMET-QUAD or COMET

Outputs None

Returns Success = `CMQ_SUCCESS`
Failure = <COMET-QUAD ERROR CODE>

Valid States `CMQ_ACTIVE, CMQ_INACTIVE`

Side Effects None

Pattern receive and generation control: cometqPRGDErrInsCfg

This function allows the user to configure the data generator and detector error insertion rate for a COMET device.

Prototype `INT4 cometqPRGDErrInsCfg(sCMQ_HNDL deviceHandle, eCMQ_ERROR_RATE errRate)`

Inputs `deviceHandle` : device Handle (from `cometqAdd`)

`errRate` : Selects the error insertion probability:
`CMQ_ERROR_RATE_OFF`,
`CMQ_ERROR_RATE_SINGLE`,
`CMQ_ERROR_RATE_10_TO_MINUS1`,
`CMQ_ERROR_RATE_10_TO_MINUS2`,
`CMQ_ERROR_RATE_10_TO_MINUS3`,
`CMQ_ERROR_RATE_10_TO_MINUS4`,
`CMQ_ERROR_RATE_10_TO_MINUS5`,
`CMQ_ERROR_RATE_10_TO_MINUS6`,
`CMQ_ERROR_RATE_10_TO_MINUS7`

Outputs

Returns Success = `CMQ_SUCCESS`
Failure = <COMET-QUAD ERROR CODE>

Valid States `CMQ_ACTIVE, CMQ_INACTIVE`

Side Effects None

4.10 Interrupt Service Functions

This section describes interrupt-service functions that perform the following tasks:

- Set, get and clear the interrupt enable mask
- Read and process the interrupt-status registers
- Poll and process the interrupt-status registers

See page 25 for an explanation of our interrupt servicing architecture.

Configuring ISR Processing: cometqISRConfig

Allows the USER to configure how ISR processing is to be handled: polling (CMQ_POLL_MODE) or interrupt driven (CMQ_ISR_MODE). If polling is selected, the USER is responsible for calling cometqPoll to collect interrupt data from the device.

ISR configuration is the same for both COMET and COMET-QUAD devices.

Prototype INT4 cometqISRConfig(sCMQ_HNDL deviceHandle, eCMQ_ISR_MODE mode)

Inputs deviceHandle : device Handle (from cometqAdd)
mode : mode of operation : CMQ_ISR_MODE or CMQ_POLL_MODE

Outputs None

Returns Success = CMQ_SUCCESS
Failure = <COMET-QUAD ERROR CODE>

Valid States CMQ_ACTIVE, CMQ_INACTIVE

Side Effects None

Getting the Interrupt Status Mask: cometqGetMask

Returns the interrupt mask currently stored within the DDB.

For COMET devices**,** all mask structure arrays within pmask except for the HDLC interrupt mask arrays use only one element as there is only one framer present on the device. There are three HDLC controllers on the COMET and the corresponding mask arrays, rdlcEn and tdpr, contain their interrupt masks as the first three elements of the array.

Prototype INT4 cometqGetMask(sCMQ_HNDL deviceHandle, sCMQ_ISR_MASK *pmask)

Inputs deviceHandle : device Handle (from cometqAdd)
pmask : (pointer to) mask structure

Outputs pmask : (pointer to) updated mask structure

Returns Success = CMQ_SUCCESS
Failure = <COMET-QUAD ERROR CODE>

Valid States CMQ_ACTIVE, CMQ_INACTIVE

Side Effects None

Setting the Interrupt Enable Mask: cometqSetMask

Sets individual interrupt bits and registers in the COMET or COMET-QUAD device. Any bits that are set in the passed structure are set in the associated COMET or COMET-QUAD registers. All other interrupt bits are left unmodified.

For COMET devices**,** all mask structure arrays within `pmask` except for the HDLC interrupt mask arrays use only one element as there is only one framer present on the device. There are three HDLC controllers on the COMET and the corresponding mask arrays, `rdlcEn` and `tdpr`, contain their interrupt masks as the first three elements of the array.

Prototype `INT4 cometqSetMask(sCMQ_HNDL deviceHandle, sCMQ_ISR_MASK *pmask)`

Inputs `deviceHandle` : device Handle (from `cometqAdd`)
`pmask` : (pointer to) mask structure

Outputs None

Returns Success = `CMQ_SUCCESS`
Failure = <COMET-QUAD ERROR CODE>

Valid States `CMQ_ACTIVE, CMQ_INACTIVE`

Side Effects May change the operation of the ISR / DPR

Clearing the Interrupt Enable Mask: cometqClearMask

Clears individual interrupt bits and registers in the COMET or COMET-QUAD device. Any bits that are set in the passed structure are cleared in the associated COMET or COMET-QUAD registers. All other interrupt bits are left unmodified.

For COMET devices**,** all mask structure arrays within `pmask` except for the HDLC interrupt mask arrays use only one element as there is only one framer present on the device. There are three HDLC controllers on the COMET and the corresponding mask arrays, `rdlcEn` and `tdpr`, contain their interrupt masks as the first three elements of the array.

Prototype `INT4 cometqClearMask(sCMQ_HNDL deviceHandle, sCMQ_ISR_MASK *pmask)`

Inputs `deviceHandle` : device Handle (from `cometqAdd`)
`pmask` : (pointer to) mask structure

Outputs None

Returns Success = `CMQ_SUCCESS`
Failure = <COMET-QUAD ERROR CODE>

Valid States `CMQ_ACTIVE, CMQ_INACTIVE`

Side Effects May change the operation of the ISR / DPR

Polling the Interrupt Status Registers: cometqPoll

Commands the driver to poll the interrupt registers in the device. The call will fail unless the device was initialized (via `cometqInit`) or configured (via `cometqISRConfig`) into polling mode.

Prototype `INT4 cometqPoll(sCMQ_HNDL deviceHandle)`

Inputs `deviceHandle` : device Handle (from `cometqAdd`)

Outputs None

Returns Success = `CMQ_SUCCESS`
Failure = <COMET-QUAD ERROR CODE>

Valid States `CMQ_ACTIVE`

Side Effects None

Interrupt Service Routine: cometqISR

Reads the state of the interrupt registers in the COMET or COMET-QUAD and stores them in an ISV. Performs whatever functions are needed to clear the interrupt, from simply clearing bits to complex functions. This routine is called by the application code from within `sysCometqISRHandler`. If ISR mode is configured, all interrupts that were detected are disabled and the ISV is returned to the Application. Note that the Application is then responsible for sending this buffer to the DPR task. If polling mode is selected, no ISV is returned to the Application and the DPR is called directly with the ISV.

Prototype `void* cometqISR(sCMQ_HNDL deviceHandle)`

Inputs `deviceHandle` : device Handle (from `cometqAdd`)

Outputs None

Returns (pointer to) ISV buffer (to send to the DPR) or NULL (pointer)

Valid States `CMQ_ACTIVE`

Side Effects None

Pseudocode
```
Begin
get an ISV buffer
update ISV with current interrupt status
if no valid interrupt condition
  return NULL
```

```
if in ISR mode
  disable all detected interrupts
  return ISV
else (Polling mode)
  call cometqDPR
  output NULL
End
```

Deferred Processing Routine: cometqDPR

Acts on data contained in the passed ISV, allocates one or more DPV buffers (via `sysCometqDPVBufferGet`) and invokes one or more callbacks (if defined and enabled). This routine is called by the application code, within `sysCometqDPRTask`. Note that the callbacks are responsible for releasing the passed DPV. It is recommended that this be done as soon as possible to avoid running out of DPV buffers.

This function operates the same for both COMET and COMET-QUAD devices.

Prototype `void cometqDPR(sCMQ_ISV* pisv)`

Inputs `pisv` : (pointer to) ISV buffer

Outputs None

Returns None

Valid States `CMQ_ACTIVE`

Side Effects None

Pseudocode

```
Begin
 for each ISV element (section)
   get and fill out a DPV buffer
   if callback (from cometqInit) is not NULL
     invoke (section) callback
 release ISV by calling sysCometqISVBufferRtn
End
```

4.11 Status and Statistics Functions

This section provides access to the on-chip statistics and counts interface. It also provides an interface to control the transmission of automatic performance reports in the transmit stream when in T1 ESF mode. The user can enable, disable, and force a manual update of the performance report as well as read the performance report to examine its contents.

Performance monitoring statistics: cometqForceStatsUpdate

This function forces the performance monitor counters obtained by calling `cometqGetStats` to be updated from the internal holding registers. This function must be called before `cometqGetStats` is invoked. For COMET-QUAD devices, registers for all four quadrants are updated. When the values have been loaded from the internal holding registers, an interrupt will be generated if the performance monitoring transfer indication interrupt is enabled.

Prototype `INT4 cometqForceStatsUpdate(sCMQ_HNDL deviceHandle)`

Inputs `deviceHandle` : device Handle (from `cometqAdd`)

Outputs None

Returns Success = `CMQ_SUCCESS`
Failure = <COMET-QUAD ERROR CODE>

Valid States `CMQ_ACTIVE, CMQ_INACTIVE`

Side Effects None

Performance monitoring statistics: cometqGetStats

This function retrieves framing statistics from the hardware T1/E1 performance monitoring registers. When calling this function, it is assumed that the user forced the registers to update by calling `cometqForceStatsUpdate`.

For COMET devices**,** the arrays within `pData` use only the first element of the four element arrays as there is only one framer present on the device.

Prototype
```
INT4 cometqGetStats(sCMQ_HNDL deviceHandle,
sCMQ_FRM_CNTS *pData)
```

Inputs `deviceHandle` : device Handle (from `cometqAdd`)

Outputs `pData` : framer statistics structure

Returns Success = `CMQ_SUCCESS`
Failure = <COMET-QUAD ERROR CODE>

Valid States `CMQ_ACTIVE, CMQ_INACTIVE`

Side Effects None

Framer status: cometqGetStatus

This function retrieves status and alarms information for either the T1 or E1 framers as appropriate based on the current operating mode. Loss of signal, loss of frame, AIS and yellow alarm status information is available for T1 and E1. For E1, loss of CRC-4 multiframe, loss of signaling multiframe, and timeslot 16 RAI indication is also available.

For COMET devices**,** the arrays within `pStatus` use only the first element of the four element arrays as there is only one framer present on the device.

Prototype
```
INT4 cometqGetStatus(sCMQ_HNDL deviceHandle,
sCMQ_FRM_STATUS *pStatus)
```

Inputs `deviceHandle` : device Handle (from `cometqAdd`)

Outputs `pStatus` : framer status structure

Returns Success = `CMQ_SUCCESS`
Failure = <COMET-QUAD ERROR CODE>

Valid States `CMQ_ACTIVE, CMQ_INACTIVE`

Side Effects None

Status of line clocks: cometqLineClkStatGet

This function provides the clock inputs status as well as the synchronization status of the clock service unit.

Prototype
```
INT4 cometqLineClkStatGet(sCMQ_HNDL deviceHandle,
UINT2 chan, sCMQ_CLK_STATUS *pclkStat)
```

Inputs `deviceHandle` : device Handle (from `cometqAdd`)
`chan` : E1/T1 channel: COMET-QUAD: 0, 1, 2, or 3
COMET: not used

Outputs `pclkStat` : clock status structure

Returns Success = `CMQ_SUCCESS`
Failure = <COMET-QUAD ERROR CODE>

Valid States `CMQ_ACTIVE, CMQ_INACTIVE`

Side Effects None

Pattern receive and generation control: cometqPRGDCntGet

This function retrieves the bit error count that is maintained in the pseudo-random generator/detector within the device.

Prototype `INT4 cometqPRGDCntGet(sCMQ_HNDL deviceHandle, UINT2 chan, UINT4 *pcount)`

Inputs `deviceHandle` : device Handle (from `cometqAdd`)
`chan` : E1/T1channel: COMET-QUAD: 0, 1, 2, or 3
COMET: not used

Outputs `pcount` : total bit error count

Returns Success = `CMQ_SUCCESS`
Failure = <COMET-QUAD ERROR CODE>

Valid States `CMQ_ACTIVE, CMQ_INACTIVE`

Side Effects None

Pattern receive and generation control: cometqPRGDGetBitCnt

This function provides the user with the bit count maintained in the PRBS within a COMET device. This function is supported by the COMET only.

Prototype `INT4 cometqPRGDGetBitCnt(sCMQ_HNDL deviceHandle, UINT4 *pcount)`

Inputs `deviceHandle` : device Handle (from *cometq*`Add`)

Outputs `pcount` : current bit count

Returns Success = `CMQ_SUCCESS`
Failure = <COMET-QUAD ERROR CODE>

Valid States `CMQ_ACTIVE, CMQ_INACTIVE`

Side Effects None

Automatic performance report generation: cometqPmonSet

Enable/Disable one second update of Auto Performance Report Monitoring (APRM). This function is valid only in T1 ESF framing mode. For COMET devices, the user can force a manual insertion of a performance report into the transmit stream when automatic transmission is disabled.

Prototype `INT4 cometqPmonSet(sCMQ_HNDL deviceHandle, UINT2 chan, eCMQ_APRM_ACTION action)`

Inputs
`deviceHandle` : device Handle (from `cometqAdd`)
`chan` : E1/T1 channel: COMET-QUAD: 0, 1, 2, or 3
COMET: not used
`action` : specifies the action to perform:
`CMQ_AUTO_PMON_UPDATE_DISABLE`,
`CMQ_AUTO_PMON_UPDATE_ENABLE`,
`CMQ_AUTO_PMON_UPDATE_MAN` (COMET only – forces manual update)

Outputs None

Returns
Success = `CMQ_SUCCESS`
Failure = <COMET-QUAD ERROR CODE>

Valid States `CMQ_ACTIVE CMQ_INACTIVE`

Side Effects None

Automatic performance report generation: cometqPmonReportGet

This API returns the current one second performance report. This function is valid only in T1 ESF framing mode.

Prototype
```
INT4 cometqPmonReportGet(sCMQ_HNDL deviceHandle, UINT2
chan, sCMQ_STAT_APRM *pPmonReport)
```

Inputs
`deviceHandle` : device Handle (from `cometqAdd`)
`chan` : E1/T1 channel: COMET-QUAD: 0, 1, 2, or 3
COMET: not used

Outputs `pPmonReport` : performance report structure

Returns
Success = `CMQ_SUCCESS`
Failure = <COMET-QUAD ERROR CODE>

Valid States `CMQ_ACTIVE CMQ_INACTIVE`

Side Effects None

4.12 Device Diagnostics

Register access test: cometqTestReg

This function verifies hardware register integrity by writing and reading back values.

The register test operates on both COMET or COMET-QUAD devices.

Prototype `INT4 cometqTestReg(sCMQ_HNDL deviceHandle)`

Inputs `deviceHandle` : device Handle (from `cometqAdd`)

Outputs None

Returns Success = `CMQ_SUCCESS`
Failure = <COMET-QUAD ERROR CODE>

Valid States `CMQ_PRESENT`

Side Effects None

Framer loopback: cometqLoopFramer

Clears / Sets a loopback of type line, payload, or digital within the E1/T1 framer section of the device. Note that when performing a line loopback, the transmit jitter attenuators reference and output clock divisors are set to 0x2F, the transmit timing options register is modified to jitter attenuated loop timing, and the transmit elastic store is enabled.

Prototype `INT4 cometqLoopFramer(sCMQ_HNDL deviceHandle, UINT2 framer, eCMQ_LOOPBACK_TYPE type)`

Inputs `deviceHandle` : device Handle (from `cometqAdd`)
`framer` : framer number: COMET-QUAD: 0, 1, 2, or 3
COMET: not used

`type` : loopback type:
`CMQ_LOOPBACK_NONE` (disable),
`CMQ_LOOPBACK_DIGITAL`,
`CMQ_LOOPBACK_LINE`, or
`CMQ_LOOPBACK_PAYLOAD`

Outputs None

Returns Success = `CMQ_SUCCESS`
Failure = <COMET-QUAD ERROR CODE>

Valid States `CMQ_ACTIVE, CMQ_INACTIVE`

Side Effects TJAT reference and output clock divisors, transmit timing options modified if line loopback specified

DS0 loopback: cometqLoopTslots

This function enables or disables DS0 loopback on specified DS0s. To facilitate DS0 loopback, the receive elastic store must be bypassed. As such, the current receive backplane configuration must be clock master otherwise DS0 loopback is not possible.

Prototype `INT4 cometqLoopTslots(sCMQ_HNDL deviceHandle, UINT2 framer, UINT4 timeSlot, UINT2 enable)`

Inputs `deviceHandle` : device Handle (from `cometqAdd`)

`framer` : framer number: COMET-QUAD: 0, 1, 2, or 3
COMET: not used

`timeSlot` : bit mask for timeslots to loopback
T1: bit 0-23 : channels 1-24
bit 24-31 : unused
E1: bit 0-31 : timeslots 0-31

`enable` : sets loop if non-zero, else clears loop

Outputs None

Returns Success = `CMQ_SUCCESS`
Failure = <COMET-QUAD ERROR CODE>

Valid States `CMQ_ACTIVE, CMQ_INACTIVE`

Side Effects None

Analog transmitter bypass: cometqTxAnalogByp

This function enables or disables bypass of the TXRING and TXTIP outputs to use the digital TDAT and TCLKO lines. This function is for COMET devices only.

Prototype `INT4 cometqTxAnalogByp(sCMQ_HNDL deviceHandle, UINT1 enable)`

Inputs `deviceHandle` : device Handle (from `cometqAdd`)

`enable` : enable/disable analog bypass

Outputs None

Returns Success = `CMQ_SUCCESS`
Failure = <COMET-QUAD ERROR CODE>

Valid States `CMQ_ACTIVE, CMQ_INACTIVE`

Side Effects None

Analog transmitter bypass: cometqRxAnalogByp

This function enables or disables bypass of the RXRING and RXTIP outputs to use the digital RDAT and RCLK lines. This function is for COMET devices only.

Prototype `INT4 cometqRxAnalogByp(sCMQ_HNDL deviceHandle, UINT1 enable)`

Inputs `deviceHandle` : device Handle (from `cometqAdd`)

`enable` : enable/disable analog bypass

Outputs None

Returns Success = `CMQ_SUCCESS`
Failure = <COMET-QUAD ERROR CODE>

Valid States `CMQ_ACTIVE, CMQ_INACTIVE`

Side Effects None

4.13 Callback Functions

The COMET-QUAD driver has the capability to callback to functions within the USER code when certain events occur. These events and their associated callback routine declarations are detailed below. There is no USER code action that is required by the driver for these callbacks – the USER is free to implement these callbacks in any manner or else they can be deleted from the driver.

The names given to the callback functions are given as examples only. The addresses of the callback functions invoked by the `cometqDPR` function are passed during the `cometqInit` call (inside a DIV). However the USER shall use the exact same prototype. The Application is left responsible for releasing the passed DPV as soon as possible (to avoid running out of DPV buffers) by calling `sysCometqDPVBufferRtn` either within the callback function or later inside the Application code.

Calling Back to the Application due to Interface events: cometqCbackIntf

This callback function is provided by the USER and is used by the DPR to report significant Interface section events back to the application. This function should be non-blocking. Typically, the callback routine sends a message to another task with the event identifier and other context information. The task that receives this message can then process this information according to the system requirements. NOTE: the callback function's addresses are passed to the driver doing the `cometqInit` call. If the address of the callback function was passed as a NULL at initialization no callback will be made.

Prototype `void cometqCbackIntf (sCMQ_USR_CTXT usrCtxt, sCMQ_DPV *pdpv)`

Inputs
`usrCtxt` : user context (from `cometqAdd`)
`pdpv` : (pointer to) DPV that describes this event

Outputs None

Returns None

Valid States `CMQ_ACTIVE`

Side Effects None

Calling Back to the Application due to T1 / E1 Framer events: cometqCbackFramer

This callback function is provided by the USER and is used by the DPR to report significant Framer section events back to the application. This function should be non-blocking. Typically, the callback routine sends a message to another task with the event identifier and other context information. The task that receives this message can then process this information according to the system requirements. NOTE: the callback function's addresses are passed to the driver doing the `cometqInit` call. If the address of the callback function was passed as a NULL at initialization no callback will be made.

Prototype `void cometqCbackFramer (sCMQ_USR_CTXT usrCtxt, sCMQ_DPV *pdpv)`

Inputs
`usrCtxt` : user context (from `cometqAdd`)
`pdpv` : (pointer to) DPV that describes this event

Outputs None

Returns None

Valid States `CMQ_ACTIVE`

Side Effects None

Calling Back to the Application due to Signal Insertion / Extraction events: cometqCbackSigInsExt

This callback function is provided by the USER and is used by the DPR to report significant Signal Insertion/Extraction section events back to the application. This function should be non-blocking. Typically, the callback routine sends a message to another task with the event identifier and other context information. The task that receives this message can then process this information according to the system requirements. NOTE: the callback function's addresses are passed to the driver doing the `cometqInit` call. If the address of the callback function was passed as a NULL at initialization no callback will be made.

Prototype `void cometqCbackSigInsExt (sCMQ_USR_CTXT usrCtxt, sCMQ_DPV *pdpv)`

Inputs

`usrCtxt`	: user context (from `cometqAdd`)
`pdpv`	: (pointer to) DPV that describes this event

Outputs None

Returns None

Valid States `CMQ_ACTIVE`

Side Effects None

Calling Back to the Application due to Performance Monitoring events: cometqCbackPMon

This callback function is provided by the USER and is used by the DPR to report significant Performance Monitoring section events back to the application. This function should be non-blocking. Typically, the callback routine sends a message to another task with the event identifier and other context information. The task that receives this message can then process this information according to the system requirements. NOTE: the callback function's addresses are passed to the driver doing the `cometqInit` call. If the address of the callback function was passed as a NULL at initialization no callback will be made.

Prototype `void cometqCbackPMon (sCMQ_USR_CTXT usrCtxt, sCMQ_DPV *pdpv)`

Inputs

`usrCtxt`	: user context (from `cometqAdd`)
`pdpv`	: (pointer to) DPV that describes this event

Outputs None

Returns None

Valid States `CMQ_ACTIVE`

Side Effects None

Calling Back to the Application due to Alarm Inband Communications events: cometqCbackAlarmInBand

This callback function is provided by the USER and is used by the DPR to report significant Alarm Inband Communications section events back to the application. This function should be non-blocking. Typically, the callback routine sends a message to another task with the event identifier and other context information. The task that receives this message can then process this information according to the system requirements. NOTE: the callback function's addresses are passed to the driver doing the `cometqInit` call. If the address of the callback function was passed as a NULL at initialization no callback will be made.

Prototype `void cometqCbackAlarmInBand (sCMQ_USR_CTXT usrCtxt, sCMQ_DPV *pdpv)`

Inputs

`usrCtxt`	: user context (from `cometqAdd`)
`pdpv`	: (pointer to) DPV that describes this event

Outputs None

Returns None

Valid States `CMQ_ACTIVE`

Side Effects None

Calling Back to the Application due to Serial Controller events: cometqCbackSerialCtl

This callback function is provided by the USER and is used by the DPR to report significant Serial Controller section events back to the application. This function should be non-blocking. Typically, the callback routine sends a message to another task with the event identifier and other context information. The task that receives this message can then process this information according to the system requirements. NOTE: the callback function's addresses are passed to the driver doing the `cometqInit` call. If the address of the callback function was passed as a NULL at initialization no callback will be made.

Prototype `void cometqCbackSerialCtl (sCMQ_USR_CTXT usrCtxt, sCMQ_DPV *pdpv)`

Inputs

`usrCtxt`	: user context (from `cometqAdd`)
`pdpv`	: (pointer to) DPV that describes this event

Outputs None

Returns None

Valid States `CMQ_ACTIVE`

Side Effects None

5 HARDWARE INTERFACE

The COMET-QUAD driver interfaces directly with the USER's hardware. In this section of the manual, a listing of each point of interface is shown, along with a declaration and any specific porting instructions. It is the responsibility of the USER to connect these requirements into the hardware, either by defining a macro or by writing a function for each item listed. Care should be taken when matching parameters and return values.

The hardware interface API is device independent and is thus the same for both COMET and COMET-QUAD devices.

5.1 Device I/O

Reading from a Device Register: sysCometqRead

The most basic hardware connection – reads the contents of a specific register location. This macro should be UINT1 oriented and should be defined by the user to reflect the target system's addressing logic. There is no need for error recovery in this function.

Format `#define sysCometqRead(addr)`

Prototype `UINT1 sysCometqRead(UINT1 *addr)`

Inputs `addr` : register location to be read

Outputs None

Returns value read from the addressed register location

Writing to a Device Register: sysCometqWrite

The most basic hardware connection - writes the supplied value to the specific register location. This macro should be UINT1 oriented and should be defined by the user to reflect the target system's addressing logic. There is no need for error recovery in this function.

Format `#define sysCometqWrite(addr, value)`

Prototype `UINT1 sysCometqWrite(UINT1 *addr, UINT1 value)`

Inputs `addr` : register location to be written
`value` : data to be written

Outputs None

Returns Value written to the addressed register location

5.2 System-Specific Interrupt Servicing

The porting of an ISR routine between platforms is a rather difficult task. There are many different implementations of these hardware level routines. In this driver, the USER is responsible for installing an interrupt handler (`sysCometqISRHandler`) in the interrupt vector table of the system processor. This handler shall call `cometqISR` for each device that has interrupt servicing enabled, to perform the ISR related housekeeping required by each device.

During execution of the API function `cometqModuleStart` / `cometqModuleStop` the driver informs the application that it is time to install / uninstall this shell via `sysCometqISRHandlerInstall` / `sysCometqISRHandlerRemove`.

Note: A device can be initialized with ISR disabled. In that mode, the USER should periodically invoke a provided 'polling' routine (`cometqPoll`) that in turn calls `cometqISR`.

Installing the ISR Handler: sysCometqISRHandlerInstall

Installs the USER-supplied Interrupt Service Routine (ISR), `sysCometqISRHandler`, into the processor's interrupt vector table.

Format `#define sysCometqISRHandlerInstall(func)`

Prototype `INT4 sysCometqISRHandlerInstall(void *func)`

Inputs `func` : (pointer to) the function `sysCometqISRHandler`

Outputs None

Returns Success = 0
Failure = <any other value>

Pseudocode
```
Begin
  install sysCometqISRHandler in processor's interrupt vector table
End
```

ISR Handler: sysCometqISRHandler

This routine is invoked when one or more COMET or COMET-QUAD devices raise the interrupt line to the microprocessor. This routine invokes the driver-provided routine, `cometqISR`, for each device registered with the driver.

Format `#define sysCometqISRHandler()`

Prototype `void sysCometqISRHandler(void)`

Inputs None

Outputs None

Returns None

Pseudocode
```
Begin
for each device registered with the driver
  call cometqISR
  if returned ISV buffer is not NULL
    send ISV buffer to the DPR
End
```

Removing the ISR Handler: sysCometqISRHandlerRemove

Disables interrupt processing for all COMET or COMET-QUAD device. Removes the USER-supplied Interrupt Service routine (ISR), `sysCometqISRHandler`, from the processor's interrupt vector table.

Format
```
#define sysCometqISRHandlerRemove()
```

Prototype
```
void sysCometqISRHandlerRemove(void)
```

Inputs None

Outputs None

Returns None

Pseudocode
```
Begin
 remove sysCometqISRHandler from the processor's interrupt vector
 table
End
```

6 RTOS INTERFACE

The COMET-QUAD driver requires the use of some RTOS resources. In this section of the manual, a listing of each required resource is shown, along with a declaration and any specific porting instructions. Note that it is the responsibility of the USER to connect these requirements into the RTOS, either by defining a macro or by writing a function for each item listed. Care should be taken when matching parameters and return values.

The RTOS interface API is device independent and is thus the same for both COMET and COMET-QUAD devices.

6.1 Memory Allocation / De-Allocation

Allocating Memory: sysCometqMemAlloc

Allocates specified number of bytes of memory.

Format `#define sysCometqMemAlloc(numBytes)`

Prototype `UINT1* sysCometqMemAlloc(UINT4 numBytes)`

Inputs `numBytes` : number of bytes to be allocated

Outputs None

Returns Success = Pointer to first byte of allocated memory
Failure = NULL pointer (memory allocation failed)

Freeing Memory: sysCometqMemFree

Frees memory that was allocated using `sysCometqMemAlloc`.

Format `#define sysCometqMemFree(pfirstByte)`

Prototype `void sysCometqMemFree(UINT1* pfirstByte)`

Inputs `pfirstByte` : pointer to first byte of the memory region being de-allocated

Outputs None

Returns None

Setting memory: sysCometqMemSet

Sets a specified contiguous block of memory to the given value.

Format `#define sysCometqMemSet(pmem, val, sz)`

Prototype `void sysCometqMemSet(UINT1* pmem, UINT1 val, size_t sz)`

Inputs `pmem` : pointer to first byte of memory to set

`val` : value to set the memory to

`sz` : number of bytes to set to `val` starting at `pmem`

Outputs None

Returns None

Copying memory: sysCometqMemCpy

Copies a given number of bytes from one memory location to another.

Format `#define sysCometqMemCpy(pdst, psrc, sz)`

Prototype `void sysCometqMemCpy(UINT1* pdst, UINT1* psrc, size_t sz)`

Inputs `pdst` : pointer to the destination

`psrc` : pointer to the source

`sz` : number of bytes to copy from the source to the destination

Outputs None

Returns None

6.2 Buffer Management

All operating systems provide some sort of buffer system, particularly for use in sending and receiving messages. The following calls, provided by the USER, allow the Driver to Get and Return buffers from the RTOS. It is the USER's responsibility to create any special resources or pools to handle buffers of these sizes during the `sysCometqBufferStart` call.

Starting Buffer Management: sysCometqBufferStart

Alerts the RTOS that the time has come to make sure ISV buffers and DPV buffers are available and sized correctly. This may involve the creation of new buffer pools and it may involve nothing, depending on the RTOS.

Format `#define sysCometqBufferStart()`

Prototype `INT4 sysCometqBufferStart(void)`

Inputs None

Outputs None

Returns Success = 0
Failure = <any other value>

Getting an ISV Buffer: sysCometqISVBufferGet

Gets a buffer from the RTOS that will be used by the ISR code to create an Interrupt Service Vector (ISV). The ISV consists of data transferred from the devices interrupt status registers.

Format `#define sysCometqISVBufferGet()`

Prototype `sCMQ_ISV* sysCometqISVBufferGet(void)`

Inputs None

Outputs None

Returns Success = (pointer to) a ISV buffer
Failure = NULL (pointer)

Returning an ISV Buffer: sysCometqISVBufferRtn

Returns an ISV buffer to the RTOS when the information in the block is no longer needed by the DPR.

Format `#define sysCometqISVBufferRtn(pISV)`

Prototype `void sysCometqISVBufferRtn(sCMQ_ISV* pisv)`

Inputs `pisv` : (pointer to) a ISV buffer

Outputs None

Returns None

Getting a DPV Buffer: sysCometqDPVBufferGet

Gets a buffer from the RTOS that will be used by the DPR code to create a Deferred Processing Vector (DPV). The DPV consists of information about the state of the device that is to be passed to the USER via a callback function.

Format `#define sysCometqDPVBufferGet()`

Prototype `sCMQ_DPV *sysCometqDPVBufferGet(void)`

Inputs None

Outputs None

Returns Success = (pointer to) a DPV buffer
Failure = NULL (pointer)

Returning a DPV Buffer: sysCometqDPVBufferRtn

Returns a DPV buffer to the RTOS when the information in the block is no longer needed by the DPR.

Format `#define sysCometqDPVBufferRtn(pDPV)`

Prototype `void sysCometqDPVBufferRtn(sCMQ_DPV *pdpv)`

Inputs `pdpv` : (pointer to) a DPV buffer

Outputs None

Returns None

Stopping Buffer Management: sysCometqBufferStop

Alerts the RTOS that the Driver no longer needs any of the ISV buffers or DPV buffers and that if any special resources were created to handle these buffers, they can be deleted now.

Format `#define sysCometqBufferStop()`

Prototype `void sysCometqBufferStop(void)`

Inputs None

Outputs None

Returns None

6.3 Timers

Sleeping a Task: sysCometqTimerSleep

Suspends execution of a driver task for a specified number of milliseconds.

Format `#define sysCometqTimerSleep(time)`

Prototype `void sysCometqTimerSleep(UINT4 time)`

Inputs `time` : sleep time in milliseconds

Outputs None

Returns Success = 0
Failure = <any other value>

6.4 Preemption

Disabling Preemption: sysCometqPreemptDis

This routine prevents the calling task from being preempted by both other tasks and any interrupt requests.

Format `#define sysCometqPreemptDis()`

Prototype `INT4 sysCometqPreemptDis(void)`

Inputs None

Outputs None

Returns Preemption key (passed back as an argument in `sysCometqPreemptEn`)

Re-Enabling Preemption: sysCometqPreemptEn

This routine allows the calling task to be preempted, granting access to both other tasks and interrupt processing.

Format `#define sysCometqPreemptEn(key)`

Prototype `void sysCometqPreemptEn(INT4 key)`

Inputs `key` : preemption key (returned by

sysCometqPreemptDis)

Outputs None

Returns None

6.5 System-Specific DPR Routine

The porting of a task between platforms is not always simple. There are many different implementations of the RTOS level parameters. In this driver, the USER is responsible for creating a 'shell' (sysCometqDPRTask) that in turn calls cometqDPR with an ISV to perform the ISR related processing that is required by each interrupting device.

During execution of the API functions cometqModuleStart and cometqModuleStop, the driver informs the application that it is time to install and uninstall this shell via the functions sysCometqDPRTaskInstall and sysCometqDPRTaskRemove, that needs to be supplied by the USER.

Installing the DPR Task: sysCometqDPRTaskInstall

Installs the DPR task as the function sysCometqDPRTask.

Format `#define sysCometqDPRTaskInstall(func)`

Prototype `INT4 sysCometqDPRTaskInstall(void *func)`

Inputs `func` : (pointer to) the function cometqDPR

Outputs None

Returns Success = 0
Failure = <any other value>

Pseudocode Begin
install sysCometqDPRTask in the RTOS as a task
End

DPR Task: sysCometqDPRTask

This routine is installed as a separate task within the RTOS. It waits for messages from the cometqISR that provide interrupt event notification and then invokes cometqDPR for the appropriate device.

Format `#define sysCometqDPRTask()`

Prototype `void sysCometqDPRTask(void)`

Inputs None

Outputs None

Returns None

Pseudocode
Begin
do
wait for an ISV buffer (sent by `cometqISR`)
call `cometqDPR` with that ISV
loop forever
End

Removing the DPR Task: sysCometqDPRTaskRemove

Informs the application that it is time to remove (suspend) the USER supplied task `sysCometqDPRTask`.

Format `#define sysCometqDPRTaskRemove()`

Prototype `void sysCometqDPRTaskRemove(void)`

Inputs None

Outputs None

Returns None

Pseudocode
Begin
remove/suspend `sysCometqDPRTask`
End

7 PORTING THE COMET-QUAD DRIVER

This section of the manual outlines how to port the COMET and COMET-QUAD device driver to your hardware and RTOS platform. However, this manual can offer only guidelines for porting the COMET and COMET-QUAD driver as each platform and application is unique.

7.1 Driver Source Files

The C source files listed in the following table contain the code for the COMET and COMET-QUAD driver. You may need to modify the existing code or develop additional code. The code is in the form of constants, macros, and functions. For the ease of porting, the code is grouped into source files (`src`) and header files (`inc`). The `src` files contain the functions and the `inc` files contain the constants and macros.

Directory	File	Description
`src`	`cmq_api.c`	Device and module management
	`cmq_diag.c`	Diagnostics functions
	`cmq_hw.c`	Hardware specific functions
	`cmq_isr.c`	ISR processing functions
	`cmq_rtos.c`	RTOS specific functions
	`cmq_stats.c`	Status and statistics functions
	`cmq_util.c`	Miscellaneous functions
	`cmq_frm.c`	Framer configuration functions
	`cmq_sig.c`	Signaling interface
	`cmq_ser.c`	Serial control functions
	`cmq_inbd.c`	Alarms and inband communications
	`cmq_intf.c`	Interface configuration functions

inc	cmq_api.h	API function prototypes
	cmq_defs.h	Constants, macros and enumerated types
	cmq_err.h	Driver error codes
	cmq_fns.h	Non-API function prototypes
	cmq_hw.h	Hardware specific constants, macros and function prototypes
	cmq_rtos.h	RTOS specific constants, macros and function prototypes
	cmq_strs.h	Driver structures
	cmq_typs.h	Standard types

7.2 Driver Porting Procedures

The following procedures summarize how to port the COMET-QUAD driver to your platform.

To port the COMET and COMET-QUAD driver to your platform:

Step 1: Port the driver's RTOS extensions (page 147)

Step 2: Port the driver to your hardware platform (page 148)

Step 3: Port the driver's application-specific elements (page 149)

Step 4: Build the driver (page 151)

Step 1: Porting Driver RTOS Extensions

The RTOS extensions encapsulate all RTOS specific services and data types used by the driver. These RTOS extensions include:

- Memory management
- Task management
- Message queues, semaphores and timers

The compiler-specific data type definitions are located in `cmq_typs.h`. The `cmq_rtos.h` and `cmq_rtos.c` files contain macros and functions for RTOS specific services.

To port the driver's RTOS extensions:

1. Modify the data types in `cmq_typs.h`. The number after the type identifies the data-type size. For example, UINT4 defines a 4-byte (32-bit) unsigned integer. Substitute the compiler types that yield the desired types as defined in this file.

2. Modify the RTOS specific macros in `cmq_rtos.h`:

Service Type	Macro Name	Description
Memory	`sysCometqMemAlloc`	Allocates the memory block
	`sysCometqMemFree`	Frees the memory block
	`sysCometqMemCpy`	Copies the memory block from src to dest
	`sysCometqMemSet`	Sets each character in the memory buffer

3. Modify the RTOS specific functions in `cmq_rtos.c`:

Service Type	Function Name	Description
Interrupt	`sysCometqDPRTask`	Deferred interrupt-processing routine (DPR)
Timer	`sysCometqTimerSleep`	Sleeps a task
Buffer	`sysCometqBufferStart`	Start buffer management
	`sysCometqISVBufferGet`	Gets a ISV buffer
	`sysCometqISVBufferRtn`	Returns a ISV buffer
	`sysCometqDPVBufferGet`	Gets a DPV buffer
	`sysCometqDPVBufferRtn`	Returns a DPV buffer
	`sysCometqBufferStop`	Stops buffer management

Step 2: Porting Drivers to Hardware Platforms

Step 2 describes how to modify the COMET-QUAD driver for your hardware platform.

To port the driver to your hardware platform:

1. Modify the hardware specific functions in `cmq_hw.c`:

Service Type	Function Name	Description
Interrupt	`sysCometqISRHandlerInstall`	Installs the interrupt handler into the processor's interrupt vector table and spawns the DPR task
	`sysCometqISRHandlerRemove`	Removes the interrupt handler from the RTOS and deletes the DPR task
	`sysCometqISRHandler`	Interrupt handler for the COMET or COMET-QUAD device
Device I/O	`sysCometqRead`	Reads from a device register
	`sysCometqWrite`	Writes to a device register

Step 3: Porting Driver Application Specific Elements

Application specific elements are configuration constants used by the API for developing an application. This section of the manual describes how to modify the application specific elements in the COMET-QUAD driver.

To port the driver's application specific elements:

1. Modify the type definition for the user context in `cmq_typs.h`. The user context is used to identify a device in your application callbacks.

2. Modify the value of the base error code (`CMQ_ERR_BASE`) in `cmq_err.h`. This ensures that the driver error codes do not overlap other error codes used in your application.

3. Define the application-specific constants for your hardware configuration in `cmq_defs.h`:

Device Constant	Description	Default
`CMQ_MAX_DEVS`	The maximum number of COMET or COMET-QUAD devices on each card	5
`CMQ_MAX_INIT_PROFS`	The maximum number of COMET or COMET-QUAD initialization profiles	5

4. Define the following application-specific constants for your RTOS-specific services in `cometq_rtos.h`:

Task Constant	Description	Default
`CMQ_MAX_ISV_BUF`	The maximum number of ISV buffers	50
`CMQ_MAX_DPV_BUF`	The maximum number of DPV buffers	950
`CMQ_MAX_MSGS`	The maximum number of messages in the message queue	250
`CMQ_DPR_TASK_PRIORITY`	Deferred Task (DPR) task priority	85
`CMQ_DPR_TASK_STACK_SZ`	DPR task stack size, in bytes	8192

5. Code the callback functions according to your application. There are sample callback functions in `cmq_app.c`. The driver will call these callback functions when an event occurs on the device. These functions must conform to the following prototype (`cback` should be replaced with your callback function name):

```
void cback(sCMQ_USR_CTXT usrCtxt, sCMQ_DPV *pdpv)
```

Step 4: Building the Driver

Step 4 describes how to build the COMET or COMET-QUAD driver.

To build the driver:

1. Ensure that the directory variable names in the makefile reflect your actual driver and directory names.

2. Compile the source files and build the COMET or COMET-QUAD driver using your make utility.

3. Link the COMET or COMET-QUAD driver to your application code.

APPENDIX A: CODING CONVENTIONS

This section of the manual describes the coding conventions used to implement PMC driver software.

Variable Type Definitions

***Table 50**: Variable Type Definitions*

Type	Description
UINT1	unsigned integer – 1 byte
UINT2	unsigned integer – 2 bytes
UINT4	unsigned integer – 4 bytes
INT1	signed integer – 1 byte
INT2	signed integer – 2 bytes
INT4	signed integer – 4 bytes

Naming Conventions

Table 51 summarizes the naming conventions followed by PMC-Sierra driver software. Detailed descriptions are then provided in the following sub-sections.

The names used in the drivers are detailed enough to make their purpose fairly clear. Note that the device name appears in prefix.

Table 51: Naming Conventions

Type	Naming convention	Examples
Macros	Uppercase, prefix with "m" and device abbreviation	mCMQ_REG_ADDR mCMQ_[BLK]_<PURPOSE>
Enumerated Types	Uppercase, prefix with "e" and device abbreviation	eCMQ_MOD_STATE eCMQ_<OBJECT>
Constants	Uppercase, prefix with device abbreviation	CMQ_SUCCESS CMQ_[CATEGORY]_<OBJECT>

Type	Naming convention	Examples
Structures	Uppercase, prefix with "s" and device abbreviation	sCMQ_DDB sCMQ_<PURPOSE>_<BLK>_[OBJECT]
API Functions	Hungarian notation, prefix with device abbreviation	cometqAdd() cometqTestReg()
Porting Functions and Macros	Hungarian notation, prefix with "sys" and device abbreviation	sysCometqRead() sysCometqISVBufferGet() sysCometq[Object]<Action>()
Non-API Functions	Hungarian notation	utilCometqReset() <blk>Cometq<Action>[Object]()
Variables	Hungarian notation	maxDevs
Pointers to variables	Hungarian notation, prefix variable name with "p"	pmaxDevs
Global variables	Hungarian notation, prefix with device abbreviation	cometqMdb

File Organization

Table 52 presents a summary of the file naming conventions. All file names start with the device abbreviation, followed by an underscore and the actual file name. File names convey their purpose with a minimum number of characters.

Table 52: File Naming Conventions

File Type	File Name	Description
API (Module and Device Management)	cmq_api.c	Generic driver API block, contains Module & Device Management API such as installing/de-installing driver instances, read/writes, and initialization profiles.
API (ISR)	cmq_isr.c	Interrupt processing is handled by this block. This includes both ISR and DPR management
API (Diagnostics)	cmq_diag.c	Device diagnostic functions

File Type	File Name	Description
API (Status and statistics)	`cmq_stats.c`	Data collection block for all device results/counts not monitored through interrupt processing
API (Interface configuration)	`cmq_intf.c`	T1/E1 line and transmit and receive backplane configuration functions
API (Device specific blocks)	`cmq_frm.c`, `cmq_ser.c`, `cmq_inbd.c`, `cmq_sig.c`	Device specific configuration functions defined in the driver architecture
Hardware Dependent	`cmq_hw.c`, `cmq_hw.h`	Hardware specific functions, constants and macros
RTOS Dependent	`cmq_rtos.c`, `cmq_rtos.h`	RTOS specific functions, constants and macros.
Other	`cmq_util.c`	Utility functions
Header file	`cmq_api.h`	Prototypes for all the API functions of the driver
Header file	`cmq_err.h`	Return codes
Header file	`cmq_defs.h`	Constants and macros, registers and bitmaps, enumerated types
Header file	`cmq_typs.h`	Standard types definition (UINT1, UINT2, etc.)
Header file	`cmq_fns.h`	Prototypes for all the non-API functions used in the driver
Header file	`cmq_strs.h`	Structures definitions

APPENDIX B: COMET-QUAD ERROR CODES

This section of the manual describes the error codes used in the COMET-QUAD device driver.

Table 53: COMET-QUAD Error Codes

Error Code	Description
CMQ_SUCCESS	Success
CMQ_FAILURE	Failure
CMQ_ERR_MEM_ALLOC	Memory allocation failure
CMQ_ERR_INVALID_ARG	Invalid argument
CMQ_ERR_INVALID_MODULE_STATE	Invalid module state
CMQ_ERR_INVALID_MIV	Invalid Module Initialization Vector
CMQ_ERR_PROFILES_FULL	Maximum number of profiles already added
CMQ_ERR_INVALID_PROFILE	Invalid profile
CMQ_ERR_INVALID_PROFILE_NUM	Invalid profile number
CMQ_ERR_INVALID_DEVICE_STATE	Invalid device state
CMQ_ERR_DEVS_FULL	Maximum number of devices already added
CMQ_ERR_DEV_ALREADY_ADDED	Device already added
CMQ_ERR_INVALID_DEV	Invalid device handle
CMQ_ERR_INVALID_DIV	Invalid Device Initialization Vector
CMQ_ERR_INT_INSTALL	Error while installing interrupts
CMQ_ERR_INVALID_MODE	Invalid ISR/polling mode
CMQ_ERR_INVALID_REG	Invalid register number
CMQ_ERR_POLL_TIMEOUT	Time-out while polling
CMQ_ERR_FIFO_OVERRUN	RDLC FIFO has overrun.

Error Code	Description
CMQ_ERR_PACKET_COMPLETE	A complete HDLC packet has been written into the buffer.
CMQ_ERR_CHANGE_OF_LINK_STATE	A change of HDLC link state event has been detected by the RDLC.
CMQ_ERR_FIFO_UNDERRUN	A TDPR underrun event has occurred.

APPENDIX C: COMET-QUAD EVENTS

This section of the manual describes the events used in the COMET-QUAD device driver. Table 54 below describes the masks that are required to interpret the bit fields within a DPV structure. Table 55 to Table 60 describe the events associated with each callback function.

Table 54: COMET-QUAD DPV Event bit masks

DPV Event Field	Bit	Event
event1	0	CMQ_EVENT_CDRC_LCV
	1	CMQ_EVENT_CDRC_LOS
	2	CMQ_EVENT_CDRC_LINE_CODE_SIG
	3	CMQ_EVENT_CDRC_CON_16ZERO
	4	CMQ_EVENT_CDRC_ALT_LOS
	5	CMQ_EVENT_RJAT_FIFO_UNDRUN
	6	CMQ_EVENT_RJAT_FIFO_OVRRUN
	7	CMQ_EVENT_TJAT_FIFO_UNDRUN
	8	CMQ_EVENT_TJAT_FIFO_OVRRUN
	9	CMQ_EVENT_PDVD_CON_16ZERO_VIOLT
	10	CMQ_EVENT_PDVD_PULSE_DENSITY_VIOLT
	11	CMQ_EVENT_XPDE_BIT_STUFF
	12	CMQ_EVENT_XPDE_CON_16ZERO_VIOLT
	13	CMQ_EVENT_XPDE_PULSE_DENSITY_VIOLT
	14	CMQ_EVENT_RLPS_ALOS
	15	CMQ_EVENT_RX_ELST_SLIP_EMPTY
	16	CMQ_EVENT_RX_ELST_SLIP_FULL
	17	CMQ_EVENT_TX_ELST_SLIP_EMPTY
	18	CMQ_EVENT_TX_ELST_SLIP_FULL

DPV Event Field	Bit	Event
	19	CMQ_EVENT_BTIF_DATA_PAR_ERR
	20	CMQ_EVENT_BTIF_SIG_PAR_ERR
	21	CMQ_EVENT_RX_ELST_CCS_SLIP_FULL
	22	CMQ_EVENT_RX_ELST_CCS_SLIP_EMPTY
	23	CMQ_EVENT_TX_ELST_CCS_SLIP_FULL
	24	CMQ_EVENT_TX_ELST_CCS_SLIP_EMPTY
	25	CMQ_EVENT_SIGX_COS_STATE
	26	CMQ_EVENT_APRM_DATA_RDY
	27	CMQ_EVENT_PMON_XFER_CNT_UPD
	28	CMQ_EVENT_PMON_XFER_CNT_OVRRUN
	29	CMQ_EVENT_PRBS_PAT_SYNC
	30	CMQ_EVENT_PRBS_BIT_ERR
	31	CMQ_EVENT_PRBS_XFER_UPD
event2	0	CMQ_EVENT_E1_FRMR_RAI_ALARM
	1	CMQ_EVENT_E1_FRMR_RMAI_ALARM
	2	CMQ_EVENT_E1_FRMR_AIS_ALARM
	3	CMQ_EVENT_E1_FRMR_AISD_ALARM
	4	CMQ_EVENT_E1_FRMR_FEBE_ALARM
	5	CMQ_EVENT_E1_FRMR_CRC_ALARM
	6	CMQ_EVENT_E1_FRMR_OOF_ALARM
	7	CMQ_EVENT_E1_FRMR_RAI_CONT_CRC_ALARM
	8	CMQ_EVENT_E1_FRMR_CONT_FEBE_ALARM
	9	CMQ_EVENT_E1_FRMR_V52LINKID_ALARM
	10	CMQ_EVENT_E1_FRMR_BR_FRM_PLS_ALARM

DPV Event Field	Bit	Event
	11	CMQ_EVENT_E1_FRMR_CRC_SUBMFRM_PLS_ALARM
	12	CMQ_EVENT_E1_FRMR_CRC_MFRM_PLS_ALARM
	13	CMQ_EVENT_E1_FRMR_MFRM_PLS_ALARM
	14	CMQ_EVENT_E1_FRMR_RED_ALARM
	15	CMQ_EVENT_E1_FRMR_CRC2NCRC
	16	CMQ_EVENT_E1_FRMR_OOF
	17	CMQ_EVENT_E1_FRMR_OOF_SMFRM
	18	CMQ_EVENT_E1_FRMR_OOF_CRC_MFRM
	19	CMQ_EVENT_E1_FRMR_COFA
	20	CMQ_EVENT_E1_FRMR_ERR
	21	CMQ_EVENT_E1_FRMR_SMFRM_ERR
	22	CMQ_EVENT_E1_FRMR_CRC_MFRM_ERR
	23	CMQ_EVENT_E1_FRMR_SA4_IND
	24	CMQ_EVENT_E1_FRMR_SA5_IND
	25	CMQ_EVENT_E1_FRMR_SA6_IND
	26	CMQ_EVENT_E1_FRMR_SA7_IND
	27	CMQ_EVENT_E1_FRMR_SA8_IND
	28	Unused
	29	Unused
	30	Unused
	31	Unused
event 3	0	CMQ_EVENT_E1_TRAN_SIGMFRM_BNDRY
	1	CMQ_EVENT_E1_TRAN_NFAS_BNDRY
	2	CMQ_EVENT_E1_TRAN_MFRM_BNDRY

DPV Event Field	Bit	Event
	3	CMQ_EVENT_E1_TRAN_SUBMFRM_BNDRY
	4	CMQ_EVENT_E1_TRAN_FRM_BNDRY
	5	CMQ_EVENT_T1_FRMR_COFA
	6	CMQ_EVENT_T1_FRMR_ERR
	7	CMQ_EVENT_T1_FRMR_BIT_ERR
	8	CMQ_EVENT_T1_FRMR_SER_FRM
	9	CMQ_EVENT_T1_FRMR_MIMIC_FRM
	10	CMQ_EVENT_T1_FRMR_INFRM
	11	CMQ_EVENT_IBCD_LPBCK_ACT_CODE
	12	CMQ_EVENT_IBCD_LPBCK_DEACT_CODE
	13	CMQ_EVENT_T1_RBOC_IDLE
	14	CMQ_EVENT_T1_RBOC_DETECT
	15	CMQ_EVENT_T1_XBOC_REPEAT
	16	CMQ_EVENT_RDLC_EVENT
	17	CMQ_EVENT_TDPR_FIFO_FILL_LOWLVL_THRESH
	18	CMQ_EVENT_TDPR_FIFO_UNDRUN
	19	CMQ_EVENT_TDPR_FIFO_OVRRUN
	20	CMQ_EVENT_TDPR_FIFO_FULL
	21	CMQ_EVENT_TDPR_PMON_RPT_RDY
	22	CMQ_EVENT_ALMI_YELLOW_ALARM
	23	CMQ_EVENT_ALMI_RED_ALARM
	24	CMQ_EVENT_ALMI_AIS_ALARM
	25	Unused
	26	Unused

DPV Event Field	Bit	Event
	27	Unused
	28	Unused
	29	Unused
	30	Unused
	31	Unused

Table 55: COMET-QUAD Events for Interface Callbacks

Event Name	Field Description
CMQ_EVENT_CDRC_LCV	Clock & data recovery line code violation
CMQ_EVENT_CDRC_LOS	Clock & data recovery loss of signal
CMQ_EVENT_CDRC_CON_16ZERO	Clock & data recovery 16 consecutive zeros detected
CMQ_EVENT_CDRC_LINE_CODE_SIG	Clock & data recovery line code signature detected
CMQ_EVENT_CDRC_ALT_LOS	Alternate loss of signal detected
CMQ_EVENT_RJAT_FIFO_OVRRUN	Receive jitter attenuation FIFO overrun
CMQ_EVENT_RJAT_FIFO_UNDRUN	Receive jitter attenuation FIFO underrun
CMQ_EVENT_TJAT_FIFO_OVRRUN	Transmit jitter attenuation FIFO overrun
CMQ_EVENT_TJAT_FIFO_UNDRUN	Transmit jitter attenuation FIFO underrun
CMQ_EVENT_PDVD_PULSE_DENSITY_VIOLT	Receive pulse density rule violation
CMQ_EVENT_PDVD_CON_16ZERO_VIOLT	Receive pulse density 16 consecutive zeros violation
CMQ_EVENT_XPDE_BIT_STUFF	Transmit pulse density bit stuff
CMQ_EVENT_XPDE_PULSE_DENSITY_VIOLT	Transmit pulse density rule violation
CMQ_EVENT_XPDE_CON_16ZERO_VIOLT	Transmit pulse density 16 consecutive zeros violation

Event Name	Field Description
CMQ_EVENT_RLPS_ALOS	Receive line analog signal loss
CMQ_EVENT_RX_ELST_SLIP_EMPTY	Receive elastic store slip buffer is empty
CMQ_EVENT_RX_ELST_SLIP_FULL	Receive elastic store slip buffer is full
CMQ_EVENT_RX_ELST_CCS_SLIP_EMPTY	Receive elastic store CCS slip buffer is empty (COMET-QUAD only)
CMQ_EVENT_RX_ELST_CCS_SLIP_FULL	Receive elastic store CCS slip buffer is full (COMET-QUAD only)
CMQ_EVENT_TX_ELST_SLIP_EMPTY	Transmit elastic store slip buffer is empty
CMQ_EVENT_TX_ELST_SLIP_FULL	Transmit elastic store slip buffer is full
CMQ_EVENT_TX_ELST_CCS_SLIP_EMPTY	Transmit elastic store CCS slip buffer is empty (COMET-QUAD only)
CMQ_EVENT_TX_ELST_CCS_SLIP_FULL	Transmit elastic store CCS slip buffer is full (COMET-QUAD only)
CMQ_EVENT_BTIF_DATA_PAR_ERR	Backplane transmit data parity error
CMQ_EVENT_BTIF_SIG_PAR_ERR	Backplane transmit signal parity error

Table 56: COMET-QUAD Events for Framer Callbacks

Event Name	Field Description
CMQ_EVENT_T1_FRMR_COFA	T1 receive framer change of frame alignment indicator
CMQ_EVENT_T1_FRMR_ERR	T1 receive framer bit Error
CMQ_EVENT_T1_FRMR_BIT_ERR	T1 receive framer payload bit error
CMQ_EVENT_T1_FRMR_SER_FRM	T1 receive framer severely errored frame
CMQ_EVENT_T1_FRMR_MIMIC_FRM	T1 receive framer mimic framing bits detected
CMQ_EVENT_T1_FRMR_INFRM	T1 receive framer established frame sync
CMQ_EVENT_E1_TRAN_SIGMFRM_BNDRY	E1 transmit signal multiframe boundary
CMQ_EVENT_E1_TRAN_NFAS_BNDRY	E1 transmit national frame alignment signal boundary alignment achieved
CMQ_EVENT_E1_TRAN_MFRM_BNDRY	E1 transmit multiframe boundary alignment achieved
CMQ_EVENT_E1_TRAN_SUBMFRM_BNDRY	E1 transmit multiframe boundary alignment achieved
CMQ_EVENT_E1_TRAN_FRM_BNDRY	E1 transmit frame boundary alignment achieved
CMQ_EVENT_E1_FRMR_RAI_ALARM	E1 Remote Alarm Indication
CMQ_EVENT_E1_FRMR_RMAI_ALARM	E1 Remote Multiframe Alarm Indication
CMQ_EVENT_E1_FRMR_AIS_ALARM	E1 Alarm Indication signal (all ones)

Event Name	Field Description
CMQ_EVENT_E1_FRMR_AISD_ALARM	E1 Alarm Indication signal (max zero density)
CMQ_EVENT_E1_FRMR_FEBE_ALARM	E1 Far End Block Error
CMQ_EVENT_E1_FRMR_CRC_ALARM	E1 Frame CRC error
CMQ_EVENT_E1_FRMR_OOF_ALARM	E1 Out of Frame
CMQ_EVENT_E1_FRMR_RAI_CONT_CRC_ALARM	E1 RAI continuous CRC
CMQ_EVENT_E1_FRMR_CONT_FEBE_ALARM	E1 continuous Far End Block Error
CMQ_EVENT_E1_FRMR_V25LINKID_ALARM	E1 V2.5 Link ID detection alarm
CMQ_EVENT_E1_FRMR_BR_FRM_PLS_ALARM	E1 framer backplane receive frame pulse indication alarm
CMQ_EVENT_E1_FRMR_CRC_SUBMFRM_PLS_ALARM	E1 Framer CRC SubMultiFrame pulse indicator alarm
CMQ_EVENT_E1_FRMR_CRC_MFRM_PLS_ALARM	E1 framer CRC MultiFrame pulse indicator alarm
CMQ_EVENT_E1_FRMR_MFRM_PLS_ALARM	E1 framer MultiFrame pulse indicator alarm
CMQ_EVENT_E1_FRMR_RED_ALARM	E1 framer red alarm
CMQ_EVENT_E1_FRMR_CRC2NCRC	E1 receive framer CRC to non-CRC network or non-CRC to CRC network mode switch
CMQ_EVENT_E1_FRMR_OOF	E1 receive framer out of frame alignment
CMQ_EVENT_E1_FRMR_OOF_SMFRM	E1 receive framer signaling out of frame
CMQ_EVENT_E1_FRMR_OOF_CRC_MFRM	E1 receive framer CRC multiframe out of frame

Event Name	Field Description
CMQ_EVENT_E1_FRMR_COFA	E1 receive framer Change of Frame Alignment
CMQ_EVENT_E1_FRMR_ERR	E1 receive framer error
CMQ_EVENT_E1_FRMR_SMFRM_ERR	E1 receive framer signaling multiframe error
CMQ_EVENT_E1_FRMR_CRC_MFRM_ERR	E1 receive framer CRC MultiFrame error
CMQ_EVENT_E1_FRMR_SA4_IND	E1 receive framer national bit Sa4 updated
CMQ_EVENT_E1_FRMR_SA5_IND	E1 receive framer national bit Sa5 updated
CMQ_EVENT_E1_FRMR_SA6_IND	E1 receive framer national bit Sa6 updated
CMQ_EVENT_E1_FRMR_SA7_IND	E1 receive framer national bit Sa7 updated
CMQ_EVENT_E1_FRMR_SA8_IND	E1 receive framer national bit Sa8 updated

Table 57: COMET-QUAD Events for Alarm and InBand Communications Callbacks

Event Name	Field Description
CMQ_EVENT_IBCD_LPBCK_ACT_CODE	Inband Communications detect loopback activate code
CMQ_EVENT_IBCD_LPBCK_DEACT_CODE	Inband Communications detect loopback deactivate code
CMQ_EVENT_ALMI_YELLOW_ALARM	T1 Alarm Management Interface yellow alarm
CMQ_EVENT_ALMI_RED_ALARM	T1 Alarm Management Interface red alarm
CMQ_EVENT_ALMI_AIS_ALARM	T1 Alarm Management Interface alarm indication signal

Event Name	Field Description
CMQ_EVENT_T1_RBOC_IDLE	T1 receive Bit Oriented Code Idle detect
CMQ_EVENT_T1_RBOC_DETECT	T1 receive Bit Oriented Code match detect
CMQ_EVENT_T1_XBOC_REPEAT	T1 Bit Oriented Code consecutively repeated (COMET-QUAD only)
CMQ_EVENT_TDPR_FIFO_FILL_LOWLVL_THRESH	Transmit Datalink Performance Report below FIFO low fill level threshold
CMQ_EVENT_TDPR_FIFO_UNDRUN	Transmit Datalink Performance Report FIFO underrun
CMQ_EVENT_TDPR_FIFO_OVRRUN	Transmit Datalink Performance Report FIFO overrun
CMQ_EVENT_TDPR_FIFO_FULL	Transmit Datalink Performance Report FIFO full
CMQ_EVENT_TDPR_PMON_RPT_RDY	Transmit Datalink Performance Monitor Report ready
CMQ_EVENT_RDLC_EVENT	Receive Data Link event

Table 58: COMET-QUAD Events for Signal Extraction Callbacks

Event Name	Field Description
CMQ_EVENT_SIGX_COS_STATE	Receive signaling change of state

Table 59: COMET-QUAD Events for Performance Monitoring Callbacks

Event Name	Field Description
CMQ_EVENT_PMON_XFER_CNT_UPD	Performance monitoring count transfer update
CMQ_EVENT_PMON_XFER_CNT_OVRRUN	Performance monitoring count transfer overrun

Event Name	Field Description
CMQ_EVENT_APRM_DATA_RDY	Automatic Performance Report Management data ready

Table 60: COMET-QUAD Events for Serial Controller Callbacks

Event Name	Field Description
CMQ_EVENT_PRBS_PAT_SYNC	Pseudo Random Binary sequence pattern sync
CMQ_EVENT_PRBS_BIT_ERR	Pseudo Random Binary sequence pattern bit error detected
CMQ_EVENT_PRBS_XFER_UPD	Pseudo Random Binary sequence pattern count update

LIST OF TERMS

APPLICATION: Refers to protocol software used in a real system as well as validation software written to validate the COMET-QUAD driver on a validation platform.

API (Application Programming Interface): Describes the connection between this MODULE and the USER's Application code.

ISR (Interrupt Service Routine): A common function for intercepting and servicing DEVICE events. This function is kept as short as possible because an Interrupt preempts every other function starting the moment it occurs, and gives the service function the highest priority while running. Data is collected, Interrupt indicators are cleared and the function ended.

DPR (Deferred Processing Routine): This function is installed as a task, at a USER configurable priority, that serves as the next logical step in Interrupt processing. Data that was collected by the ISR is analyzed and then calls are made into the Application that inform it of the events that caused the ISR in the first place. Because this function is operating at the task level, the USER can decide on its importance in the system, relative to other functions.

DEVICE: ONE COMET-QUAD Integrated Circuit. There can be many Devices, all served by this ONE Driver MODULE:

- DIV (DEVICE Initialization Vector): Structure passed from the API to the DEVICE during initialization; it contains parameters that identify the specific modes and arrangements of the physical DEVICE being initialized.
- DDB (DEVICE Data Block): Structure that holds the Configuration Data for each DEVICE.

MODULE: All of the code that is part of this driver. There is only ONE instance of this MODULE connected to ONE OR MORE COMET-QUAD chips.

- MIV (MODULE Initialization Vector): Structure passed from the API to the MODULE during initialization. It contains parameters that identify the specific characteristics of the Driver MODULE being initialized.
- MDB (MODULE Data Block): Structure that holds the Configuration Data for this MODULE.

RTOS (Real Time Operating System): The host for this Driver.

ACRONYMS

API: Application programming interface

DDB: Device data block

DIV: Device initialization vector

DPR: Deferred processing routine

DPV: Deferred processing (routine) vector

FIFO: First in, first out

MDB: Module data block

MIV: Module initialization vector

ISR: Interrupt service routine

ISV: Interrupt service (routine) vector

RTOS: Real-time operating system

INDEX