BROADCOM®

Drivers To OMCI Performance Monitoring Application Program Interface Specification

For BCM968xx GPON Linux

Version 0.3

5300 California Avenue Irvine, CA 92617 USA • Phone: (949) 450-8700 • Fax: (949) 450-8710

BCM968xx Drivers To OMCI PM API Specification For GPON Linux

REVISION HISTORY

Revision Number	*Date*	*Change Description*
V0.3	03/02/2009	Add txOctets, txPackets, txBroadcastPackets, and txMulticastPackets counters to BCM_OMCI_PM_ETHERNET_3_COUNTER struct
V0.2	02/05/2009	• Add hec counters to FEC struct. • Change type of all counters in GEM port struct to UINT32.
V0.1	11/26/2008	Initial Release.

Table Of Contents

1 INTRODUCTION

1.1 OVERVIEW

The purpose of this document is to describe the interface between drivers API and OMCI (ONT Management and Control Interface) PM (Peformance Monitoring) Application for the Broadcom GPON modem (BCM6xx) software configuration. A fundamental goal for the development of the API is to provide a simple and efficient method to extract counters information from all the drivers API. The Drivers To OMCI PM API allows application developers to develop proprietary applications for retrieving counters in the GPON modem. Apart from vendor application developers, the Drivers To OMCI PM API is used by ONT Management and Control Interface Performance Monitoring Daemon.

The Drivers To OMCI PM API allows software applications running on the BCM68xx GPON platform to programmatically retrieve the performance monitoring counters using simple programming interfaces defined. It supports the following features:

- Exports counters information to the applications as objects. Each object is uniquely identified and its relation to other objects is specified unambiguously.
- Provides seamless access across all applications that need to access counters information from all the drivers.
- Hides the implementation details and provides flexibility to the applications to retrieve the counters information from all the drivers.
- It also allows to separate the Operating System (OS) dependent and OS independent sections of the code to make future porting effort of Drivers To OMCI PM API library much more simpler without any modifications to the applications that use the API.

A high-level view of the implementation and usage of the Drivers To OMCI PM API and the applications using the API is as shown in Fig. 1. More details of the transactions are specified in later sections of this document. A sample configuration sequence is also specified in the later sections of the document.

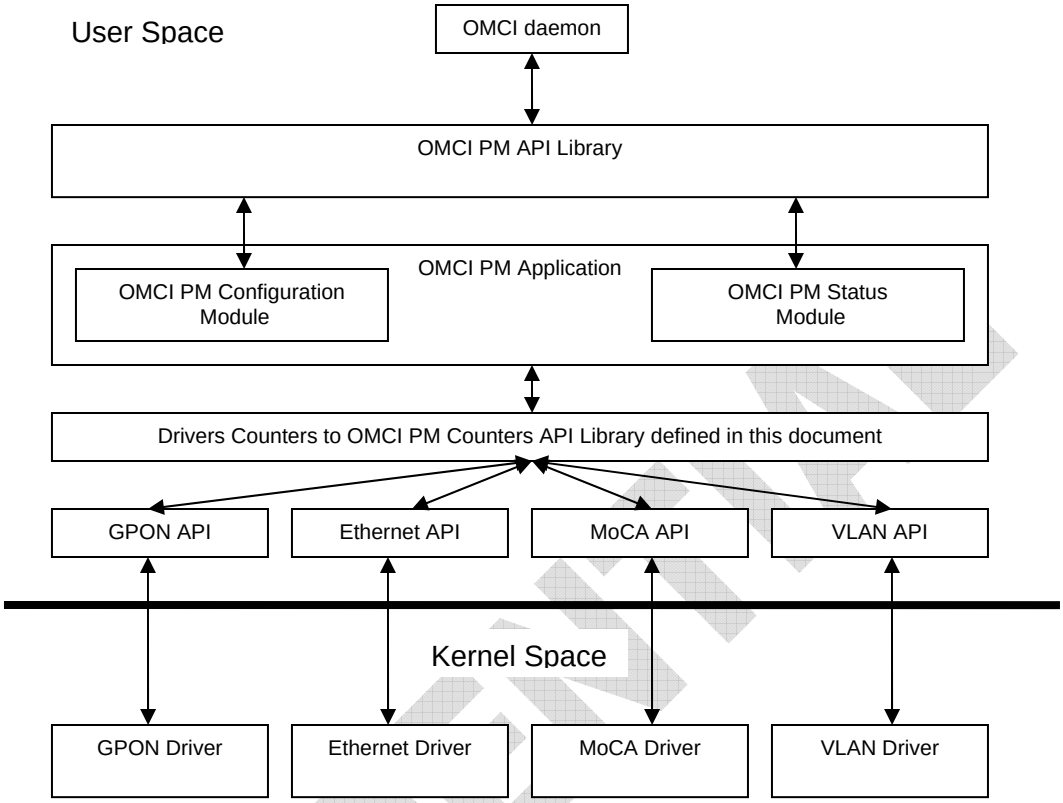

Fig. 1 OMCI PM Architecture

1.2 DISTRIBUTION

The Drivers To OMCI PM API consists of a set of C language functions. These functions are distributed in object form as a library or in source code form. C language header files are also distributed that must be included in application source files that call Drivers To OMCI PM API functions. The header files contain constant definitions, data structures and function prototypes described in this document.

1.3 PORTABILITY

The Drivers To OMCI PM API will be supported as an application on Linux. Since its interaction with the operating system is not very extensive, all operating system dependent functionality will be clearly isolated to make porting to other operating systems easy.

1.4 TERMINOLOGY

- **Object:** An object is a logical or physical entity which consists of a collection of attributes. Objects in Drivers To OMCI PM API library can be read-only. Since these objects contain only counters information, the attributes of these objects are read-only and cannot be modified. An object defines the columns of the table.
- **Object ID:** Object identifier is unique and is defined in Table 1 below. The object ID of an object denotes a particular class of performance monitoring managed entity.
- **Physical port ID:** Physical port identifier is also unique. It is used to identify which physical port of the driver that counters information should be retrieved.

2 DRIVERS TO OMCIPM API SUMMARY

2.1 DRIVERS TO OMCIPM OBJECTS

The following is the list of all the counters objects available in the system at the time of writing this document. This list must be updated with each new release.

Table 1.

Object ID	Object Structure
GPON Performance Monitoring Group	
BCM_OMCI_PM_CLASS_GEM_PORT	BCM_OMCI_PM_GEM_PORT_COUNTER
BCM_OMCI_PM_CLASS_FEC	BCM_OMCI_PM_FEC_COUNTER
Ethernet Performance Monitoring Group	
BCM_OMCI_PM_CLASS_ENET	BCM_OMCI_PM_ETHERNET_COUNTER
BCM_OMCI_PM_CLASS_ENET2	BCM_OMCI_PM_ETHERNET_2_COUNTER
BCM_OMCI_PM_CLASS_ENET3	BCM_OMCI_PM_ETHERNET_3_COUNTER
MoCA Performance Monitoring Group	
BCM_OMCI_PM_CLASS_MOCA_ENET	BCM_OMCI_PM_MOCA_ETHERNET_COUNTER
BCM_OMCI_PM_CLASS_MOCA_INTF	BCM_OMCI_PM_MOCA_INTERFACE_COUNTER
GAL Ethernet Performance Monitoring Group	
BCM_OMCI_PM_CLASS_GAL_ENET	BCM_OMCI_PM_GAL_ETHERNET_COUNTER
Bridge Performance Monitoring Group	
BCM_OMCI_PM_CLASS_BRIDGE	BCM_OMCI_PM_BRIDGE_COUNTER
BCM_OMCI_PM_CLASS_BRIDGE_PORT	BCM_OMCI_PM_BRIDGE_PORT_COUNTER

2.1.1 GPON PERFORMANCE MONTIORING GROUP

The GPON performance monitoring group is a collection of all the objects that support retrieving operation for GPON physical interface. The following is a brief description of each object and its usage:

Object ID: BCM_OMCI_PM_CLASS_GEM_PORT

Description: This object collects performance monitoring data associated with a GEM port network CTP. Instances of this object can be retrieved by the Drivers To OMCI PM API. An instance of this object is associated with an instance of the physical path termination point GEM Port UNI.

Object ID: BCM_OMCI_PM_CLASS_FEC

Description: This object collects performance monitoring data associated with Forward Error Correction counters for GPON physical interface. Instances of this object can be retrieved by the OMCI PM API.

2.1.2 ETHERNET PERFORMANCE MONITORING GROUP

The Ethernet performance monitoring group is a collection of all the objects that support configuration and status retrieving operations for Ethernet physical interface. The following is a brief description of each object and its usage:

Object ID: BCM_OMCI_PM_CLASS_ENET

Description: This object collects some of the performance monitoring data for an Ethernet interface. Instances of this object can be retrieved by the Drivers To OMCI PM API. An instance of this object is associated with an instance of the physical path termination point Ethernet UNI.

Object ID: BCM_OMCI_PM_CLASS_ENET2

Description: This object collects additional performance monitoring data for an Ethernet interface. Instances of this object can be retrieved by the Drivers To OMCI PM API. An instance of this object is associated with an instance of the physical path termination point Ethernet UNI.

Object ID: BCM_OMCI_PM_CLASS_ENET3

Description: This object collects performance monitoring data associated with an Ethernet interface. It includes parameters defined in the Ethernet statistics group of RFC 2819 that are not already covered by previously defined Ethernet monitoring MEs. The received direction is from the CPE toward the network (upstream). Instances of this object can be retrieved by the OMCI PM API. An instance of this object is associated with an instance of the physical path termination point Ethernet UNI.

2.1.3 MOCA PERFORMANCE MONITORING GROUP

The MoCA performance monitoring group is a collection of all the objects that support configuration and status retrieving operations for MoCA physical interface. The following is a brief description of each object and its usage:

Object ID: BCM_OMCI_PM_CLASS_MOCA_ENET

Description: This object collects the performance monitoring data for an MoCA Ethernet interface. Instances of this object can be retrieved by the Drivers To OMCI PM API. An instance of this object is associated with an instance of the physical path termination point MoCA UNI.

Object ID: BCM_OMCI_PM_CLASS_MOCA_INTF

Description: This object collects the performance monitoring data for an MoCA interface. Instances of this object can be retrieved by the Drivers To OMCI PM API. This object has only current values, which are retrievable by get and get next operations; no history is retained. An instance of this object is associated with an instance of the physical path termination point MoCA UNI.

2.1.4 GAL ETHERNET PERFORMANCE MONITORING GROUP

The GAL Ethernet group has a single object which is used to configure the GAL Ethernet GEM interworking termination point when the GEM layer provides Ethernet service. The following is a brief description and its usage:

Object ID: BCM_OMCI_PM_CLASS_GAL_ENET

Description: This object collects performance monitoring data associated with a GEM interworking termination point when the GEM layer provides Ethernet service. Instances of this object can be retrieved by the Drivers To OMCI PM API. An instance of this object is associated with an instance of the GEM interworking termination point.

2.1.5 BRIDGE PERFORMANCE MONITORING GROUP

The Bridge performance monitoring group is a collection of all the objects that support configuration and status retrieving operations for the MAC bridge. The following is a brief description of each object and its usage:

Object ID: BCM_OMCI_PM_CLASS_BRIDGE

Description: This object collects the performance monitoring data associated with the MAC bridge. Instances of this object can be retrieved by the Drivers To OMCI PM API. An instance of this object is associated with one instance of a MAC bridge service profile.

Object ID: BCM_OMCI_PM_CLASS_BRIDGE_PORT

Description: This object collects the performance monitoring data associated with the MAC bridge port. Instances of this object can be retrieved by the Drivers To OMCI PM API. An instance of this object is associated with an instance of the a MAC bridge port.

3 DRIVERS TO OMCIPM API

This section describes the Drivers To OMCI PM API and its usage. These API are used by applications such as OMCIPMD to get, and get next counters objects.

3.1 FUNCTION: BCM_OMCIPM_GETCOUNTERS

Description: This API call is used by the applications to retrieve the PM counters information of any object from the OMCIPM. The application user of this API will pass the object ID, a physical port ID to identify which port that counters information should be retrieved, and a pointer to a void type that can hold the counters structure. The applications must allocate the appropriate counters structure corresponding with the given object ID, and free this structure if necessary after using it. The API will return the entry if found with success status, otherwise error status will be returned.

3.2 FUNCTION: BCM_OMCIPM_GETCOUNTERSNEXT

Description: This API call is special function that can only be used with OMCI PM object that supports get next operation such as BCM_OMCI_PM_CLASS_MOCA_INTF. It is used by the applications to retrieve the next PM counters information. The application user of this API will pass the object ID, a pointer to a physical port ID, and a pointer to a void type that can hold the counters structure. This API will search the object table starting from the physical port ID plus one, and return the next available counters information. This API also returns the corresponding physical port ID if success. Otherwise it returns error status, and values in physical port ID and counters structure are undefined. The applications must allocate the appropriate counters structure corresponding with the given object ID, and free this structure if necessary after using it.

4 DRIVERS TO OMCIPM API DATA STRUCTURES

4.1 DRIVERS TO OMCI PM CLASS STRUCTURE

This section defines enum structure that is used to identify OMCI PM class objects.

typedef enum {

BCM_OMCI_PM_CLASS_GPON=0,

BCM_OMCI_PM_CLASS_FEC,

BCM_OMCI_PM_CLASS_ENET,

BCM_OMCI_PM_CLASS_ENET2,

BCM_OMCI_PM_CLASS_ENET3,

BCM_OMCI_PM_CLASS_MOCA_ENET,

BCM_OMCI_PM_CLASS_MOCA_INTF,

BCM_OMCI_PM_CLASS_GAL_ENET,

BCM_OMCI_PM_CLASS_BRIDGE,

BCM_OMCI_PM_CLASS_BRIDGE_PORT,

BCM_OMCI_PM_CLASS_MAX

} BCM_OMCI_PM_CLASS_ID;

4.2 DRIVERS TO OMCIPM COUNTERS STRUCTURES

This section defines data structures that are used to retrieve the counters information of OMCI PM objects. The only operation is permitted on the counters objects is read. The following standard types definitions are assumed:

- UINT8 – unsigned 8 bit integer or unsigned char
- UINT16 – unsigned 16 bit integer or unsigned short
- UINT32 – unsigned 32 bit integer or unsigned int

4.2.1 BCM_OMCI_PM_GEM_PORT_COUNTER

typedef struct {

OUT UINT32 lostPackets;

OUT UINT32 receivedPackets;

OUT UINT32 receivedBlocks;

OUT UINT32 receivedFragments;

OUT UINT32 transmittedPackets;

OUT UINT32 transmittedBlocks;

OUT UINT32 transmittedFragments;

OUT UINT32 misinsertedPackets;

OUT UINT32 impairedBlocks;

} BCM_OMCI_PM_GEM_PORT_COUNTER, *PBCM_OMCI_PM_GEM_PORT_COUNTER;

`lostPackets` - Partially Compliant: counts aggregate uncorrectable HEC. Not buffer overflows

`receivedPackets` - Compliant

`receivedBlocks` - Compliant

`receivedFragments` – Compliant, Non-standard Counter

`transmittedPackets` – Compliant, Non-standard Counter

`transmittedBlocks` - Compliant

`transmittedFragments` – Compliant, Non-standard Counter

`misinsertedPackets` - Not Compliant, Always returns 0

`impairedBlocks` - Not Compliant, Always returns 0

4.2.2 BCM_OMCI_PM_FEC_COUNTER

typedef struct {

OUT UINT32 correctedBytes;

OUT UINT32 correctedCodeWords;

OUT UINT32 uncorrectedCodeWords;

OUT UINT32 totalCodeWords;

OUT UINT32 fecSeconds;

OUT UINT32 hecCorrectedCodeWords;

OUT UINT32 hecUncorrectedCodeWords;

} BCM_OMCI_PM_FEC_COUNTER, *PBCM_OMCI_PM_FEC_COUNTER;

`correctedBytes` - Not Compliant, Always returns 0

`correctedCodeWords` - Compliant

`uncorrectedCodeWords` - Compliant

`totalCodeWords` - Compliant

`fecSeconds` - Compliant

`hecCorrectedCodeWords` - Compliant, Non-standard Counter

`hecUncorrectedCodeWords` - Compliant, Non-standard Counter

4.2.3 BCM_OMCI_PM_ETHERNET_COUNTER

typedef struct {

OUT UINT32 fcsErrors;

OUT UINT32 excessiveCollisionCounter;

OUT UINT32 lateCollisionCounter;

OUT UINT32 frameTooLongs;

OUT UINT32 bufferOverflowsOnReceive;

OUT UINT32 bufferOverflowsOnTransmit;

OUT UINT32 singleCollisionFrameCounter;

OUT UINT32 multipleCollisionsFrameCounter;

OUT UINT32 sqeCounter;

OUT UINT32 deferredTransmissionCounter;

OUT UINT32 internalMacTransmitErrorCounter;

OUT UINT32 carrierSenseErrorCounter;

OUT UINT32 alignmentErrorCounter;

OUT UINT32 internalMacReceiveErrorCounter;

} BCM_OMCI_PM_ETHERNET_COUNTER, *BCM_OMCI_PM_ETHERNET_COUNTER;

`fcsErrors` -

excessiveCollisionCounter -

lateCollisionCounter -

frameTooLongs -

bufferOverflowsOnReceive -

singleCollisionFrameCounter -

multipleCollisionsFrameCounter -

sqeCounter -

deferredTransmissionCounter -

internalMacTransmitErrorCounter -

carrierSenseErrorCounter -

alignmentErrorCounter -

internalMacReceiveErrorCounter -

4.2.4 BCM_OMCI_PM_ETHERNET_2_COUNTER

typedef struct {

OUT UINT32 pppoeFilterFrameCounter;

} BCM_OMCI_PM_ETHERNET_2_COUNTER, *PBCM_OMCI_PM_ETHERNET_2_COUNTER;

pppoeFilterFrameCounter -

4.2.5 BCM_OMCI_PM_ETHERNET_3_COUNTER

typedef struct {

OUT UINT32 dropEvents;

OUT UINT32 octets;

OUT UINT32 packets;

OUT UINT32 broadcastPackets;

OUT UINT32 multicastPackets;

OUT UINT32 undersizePackets;

OUT UINT32 fragments;

```
    OUT UINT32 jabbers;
    OUT UINT32 packets64Octets;
    OUT UINT32 packets127Octets;
    OUT UINT32 packets255Octets;
    OUT UINT32 packets511Octets;
    OUT UINT32 packets1023Octets;
    OUT UINT32 packets1518Octets;
    OUT UINT32 txOctets;
    OUT UINT32 txPackets;
    OUT UINT32 txBroadcastPackets;
    OUT UINT32 txMulticastPackets;
} BCM_OMCI_PM_ETHERNET_3_COUNTER, *PBCM_OMCI_PM_ETHERNET_3_COUNTER;
```

dropEvents -

octects -

packets -

broadcastPackets -

multicastPackets -

undersizePackets -

fragments -

jabbers -

packets64Octects -

packets127Octects -

packets255Octects -

packets511Octects -

packets1023Octects -

packets1518Octects -

txOctects -

txPackets -

txBroadcastPackets -

txMulticastPackets -

4.2.6 BCM_OMCI_PM_MOCA_ETHERNET_COUNTER

typedef struct {

OUT UINT32 incomingUnicastPackets;

OUT UINT32 incomingDiscardedPackets;

OUT UINT32 incomingErroredPackets;

OUT UINT32 incomingUnknownPackets;

OUT UINT32 incomingMulticastPackets;

OUT UINT32 incomingBroadcastPackets;

OUT UINT32 incomingOctets;

OUT UINT32 outgoingUnicastPackets;

OUT UINT32 outgoingDiscardedPackets;

OUT UINT32 outgoingErroredPackets;

OUT UINT32 outgoingUnknownPackets;

OUT UINT32 outgoingMulticastPackets;

OUT UINT32 outgoingBroadcastPackets;

OUT UINT32 outgoingOctets;

} BCM_OMCI_PM_MOCA_ETHERNET_COUNTER, *PBCM_OMCI_PM_MOCA_ETHERNET_COUNTER;

incomingUnicastPackets -

incomingDiscardedPackets -

incomingErroredPackets -

incomingUnknownPackets -

incomingMulticastPackets -

incomingBroadcastPackets -

incomingOctets -

outgoingUnicastPackets -

outgoingDiscardedPackets -

outgoingErroredPackets -

outgoingUnknownPackets -

outgoingMulticastPackets -

outgoingBroadcastPackets -

outgoingOctets -

4.2.7 BCM_OMCI_PM_MOCA_INTERFACE_COUNTER

typedef struct {

OUT UINT32 phyTxBroadcastRate;

} BCM_OMCI_PM_MOCA_INTERFACE_COUNTER, *PBCM_OMCI_PM_MOCA_INTERFACE_COUNTER;

phyTxBroadcastRate -

4.2.8 BCM_OMCI_PM_MOCA_INTERFACE_ENTRY_COUNTER

typedef struct {

OUT UINT32 phyTxRate;

OUT UINT32 txPowerControlReduction;

OUT UINT32 phyRxRate;

OUT UINT32 rxPowerLevel;

OUT UINT32 phyRxBroadcastRate;

OUT UINT32 rxBroadcastPowerLevel;

OUT UINT32 txPackets;

OUT UINT32 rxPackets;

OUT UINT32 erroredMissedRxPackets;

OUT UINT32 erroredRxPackets;

} BCM_OMCI_PM_MOCA_INTERFACE_ENTRY_COUNTER, *PBCM_OMCI_PM_MOCA_INTERFACE_ENTRY_COUNTER;

`phyTxRate` -

`txPowerControlReduction` -

`phyRxRate` -

`rxPowerLevel` -

`phyRxBroadcastRate` -

`rxBroadcastPowerLevel` -

`txPackets` -

`rxPackets` -

`erroredMissedRxPackets` -

`erroredRxPackets` -

4.2.9 BCM_OMCI_PM_GAL_ETHERNET_COUNTER

typedef struct {

OUT UINT32 discardedFrames;

OUT UINT32 transmittedFrames;

OUT UINT32 receivedFrames;

} BCM_OMCI_PM_GAL_ETHERNET_COUNTER, *PBCM_OMCI_PM_GAL_ETHERNET_COUNTER;

`discardedFrames` - Partialy Compliant, do not count erroneous FCS

`transmittedFrames` - Compliant, Non-standard Counter

`receivedFrames` - Compliant, Non-standard Counter

4.2.10 BCM_OMCI_PM_MAC_BRIDGE_COUNTER

typedef struct {

OUT UINT32 learningDiscaredEntries;

} BCM_OMCI_PM_MAC_BRIDGE_COUNTER, *PBCM_OMCI_PM_MAC_BRIDGE_COUNTER;

`learningDiscaredEntries` - Not Compliant, Always returns 0.

4.2.11 BCM_OMCI_PM_MAC_BRIDGE_PORT_COUNTER

typedef struct {

OUT UINT32 forwardedFrames;

OUT UINT32 delayDiscardedFrames;

OUT UINT32 mtuDiscardedFrames;

OUT UINT32 receivedFrames;

OUT UINT32 receivedDiscardedFrames;

} BCM_OMCI_PM_MAC_BRIDGE_PORT_COUNTER, *PBCM_OMCI_PM_MAC_BRIDGE_PORT_COUNTER;

`forwardedFrames` - Compliant

`delayDiscardedFrames` - Compliant

`mtuDiscardedFrames` - Compliant

`receivedFrames` - Compliant

`receivedDiscardedFrames` - Compliant

5 DRIVERS TO OMCIPM API FUNCTIONS

The Drivers To OMCIPM API is used between an OMCIPM daemon and the various drivers API such as the GPON API, Ethernet API, MoCA API, etc.

5.1 BCM_OMCIPM_GETCOUNTERS

Syntax

```
BCM_OMCI_PM_STATUS
   bcm_omcipm_getCounters(BCM_OMCI_PM_CLASS_ID classId,
   UINT16 physPortId, void *counters);
```

Parameters

[IN] BCM_OMCI_PM_CLASS_ID classId
Constant that represents an object. For values see table 1.
[IN] UINT16 physPortId
Physical port identifier of the driver.
[OUT] void *counters
Pointer to the object that contains counters information. It's casted to a void type.

5.2 BCM_OMCIPM_GETCOUNTERSNEXT

Syntax

```
BCM_OMCI_PM_STATUS
   bcm_omcipm_getCountersNext(BCM_OMCI_PM_CLASS_ID
   classId, UINT16 *physPortId, void *counters);
```

Parameters

[IN] BCM_OMCI_PM_CLASS_ID classId
Constant that represents an object. For values see table 1.
[INOUT] UINT16 *physPortId
Physical port identifier of the driver.
[OUT] void *counters
Pointer to the object that contains counters information. It's casted to a void type.