



# OMCI 의 이해

---

## Understanding of OMCI

*Release 1.0*

2008/10/7

Copyright 2008 novo networks. All rights reserved.

All information contained herein is the property of novo networks. No part of this publication (whether in hardcopy or electronic form) may be reproduced or transmitted, in any form or by any means, electronic, mechanical, photocopying, recording, or otherwise, without the prior written permission of novo networks.

No responsibility is assumed by novo networks for the use thereof nor for the rights of third parties which may be effected in any way by the use thereof. Any representation(s) in this document concerning performance of novo networks' product(s) are for informational purposes only and are not warranties of future performance

All other trademarks, service marks, registered trademarks, or registered service marks may be the property of their respective owners. All specifications are subject to change without prior notice.

Figures from ITU-T G.984.4 belongs to ITU.

**목 차 (Table of Contents)**

1. AUDIENCE .....	4
2. INTRODUCTION.....	5
3. OMCI PROTOCOL .....	6
4. MIB DESCRIPTION.....	11
5. OPERATIONS .....	21
6. SPECIAL CONCEPTS.....	39
7. MIB DESCRIPTION의 작성 및 확장 .....	43
APPENDIX A. 문서정보.....	45
A.1 문서이력 .....	45
APPENDIX B. REFERENCES.....	46

## 1. Audience

본 자료는 GPON 시스템의 OMCI Agent Toolkit 을 개발하면서 얻은 경험을 공유하고자 작성되었다. OMCI 관련 작업을 하거나, OMCI 에 대한 기본적인 이해를 하고 싶은 분들을 위한 것이다. 사실 OMCI 의 규격 자체를 처음 접하면서, 그 내용을 완전하게 이해하는 것은 쉽지 않은 일이다. 게다가 OMCI 는 특수 분야이기 때문에 물어볼 전문가를 찾기도 쉽지 않기 때문에 모든 것을 스스로 해결해야만 한다. 이러한 어려움에 도움을 주고자 하는 것이 이 문서의 목적이며, 가능하면 쉬운 설명을 사용하였는데, 어떨지 모르겠다. 아무쪼록 OMCI 의 이해에 도움이 되면 좋겠다는 바람이다.

특별한 background 는 필요하지 않으나, protocol 의 기본적인 이해는 하고 있다는 가정에서 설명하였다.

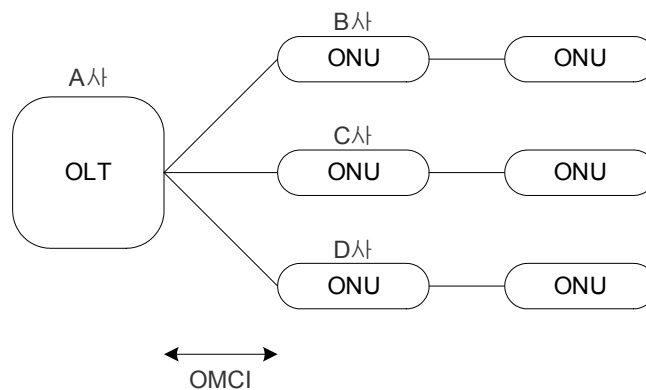
## 2. Introduction

### ■ 범위

본 문서가 다루고 있는 주요내용은 ITU-T 의 G.984.4 의 Chapter 11, Appendix I 및 II 이며, Class 의 설명을 위하여 Chapter 9 부분도 일부분 포함되어 있다.

### ■ OMCI 를 왜 만들었을까?

FTTH 을 구성할 수 있는 주요 시스템으로는 E-PON, WDM-PON, 그리고 G-PON 과 같은 시스템이 있다. 이중에 GPON 시스템에서 채택하여 만든 것이 OMCI 인데, OMCI 의 목적은 아주 간단하다. 우선 FTTH 시스템은 전화국에 설치하는 OLT 와 가입자 단의 ONU/ONT 로 (아래부터는 ONU 로 부른다) 구성이 되는데, 일반적으로 OLT 를 한 번 설치하면 그 제조 회사의 ONU 만 사용해야 한다. 이른바 제조사에 대한 Lock-in 이 발생하게 되는데, 장비를 구매하는 FTTH 사업자에게는 썩 기분 좋은 일이 아니다. GPON 의 OMCI 는 타사간의 OLT-ONU 연동을 보장하여, FTTH 사업자가 마음에 드는 OLT 제품과 ONU 제품을 마음대로 사용할 수 있도록 보장 하겠다는 것이다.



### 3. OMCI Protocol

G-PON 은 B-PON 에서 발전한 시스템이며, 이는 ATM 을 기반으로 하기 있기 때문에, OMCI 의 PDU 는 모두 53 byte 으로 되어 있다. 왼쪽부터 하나씩 알아보자.

GEM header (5 bytes)	Transaction correlation identifier (2 bytes)	Message type (1 byte)	Device identifier (1 byte)	Message identifier (4 bytes)	Message contents (32 bytes)	OMCI trailer (8 bytes)
-------------------------	---	--------------------------	-------------------------------	---------------------------------	--------------------------------	---------------------------

Figure 11-1/G.984.4

■ **GEM Header**

GEM Header 는 GPON 의 H/W 부분에서 사용하므로 OMCI 에서는 큰 의미가 없다.

■ **Transaction correlation identifier**

Transaction correlation identifier 는 간단하게 TID 라고 부르는데, 용도는 간단하다. OLT 에서 ONU 로 여러 개의 Request 를 동시에 보내는 경우, 돌아온 Response 가 어떤 Request 의 것인지 알 수 있도록 OLT 나름대로의 숫자를 적어 보내는 것이다. 이를 위해 ONU 는 OLT 로부터 받은 숫자를 그대로 돌려주도록 되어 있다. 약간의 기능을 추가하여 TID 의 첫 번째 Bit 를 Priority 용도로 사용하고 있는데, 1 인 경우에 High Priority 로 ONU 가 처리하도록 되어 있다. 그러나 대부분의 경우, ONU 에서 오래 걸리는 OMCI Request 가 별로 없기 때문에, 실제로 의미는 크게 없어 보인다.

(Retransmission 와 관련된 TID 에 대한 설명은 6.Special concepts에서 이어진다.)

■ **Message Type**

Message Type 은 아래와 같이 4 가지로 구분되어 있는데, 이 또한 아주 간단하다.

8	7	6	5	1
DB	AR	AK	MT	

Figure 11-2/G.984.4

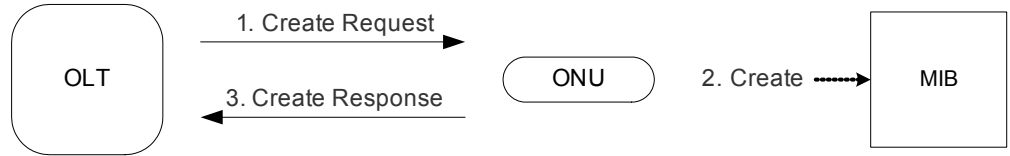
오른쪽부터 설명을 하면 MT 는 Message type 이다 (또? = 그렇다. 이름이 동일하다. 이런) MT 는 5 bit 이니 총 32 개의 종류를 나타낼 수 있는데 4 부터 28 까지 사용한다. 다음의 Table 은 Message type 의 encoding 값이다.

**Table 11-1/G.984.4 – OMCI message types**

MT	Type	Purpose	AK	Inc MIB data sync.
4	Create	Create a managed entity instance with its attributes	yes	yes
5	Create complete connection	Deprecated	–	–
6	Delete	Delete a managed entity instance	yes	yes
7	Delete complete connection	Deprecated	–	–
8	Set	Set one or more attributes of a managed entity	yes	yes
9	Get	Get one or more attributes of a managed entity	yes	no
10	Get complete connection	Deprecated	–	–
11	Get all alarms	Latch the alarm statuses of all managed entities and reset the alarm message counter	yes	no
12	Get all alarms next	Get the active alarm status of the next managed entity	yes	no
13	MIB upload	Latch the MIB	yes	no
14	MIB upload next	Get latched attributes of a managed entity instance	yes	no
15	MIB reset	Clear the MIB and re-initialize it to its default and reset the MIB data sync counter to 0	yes	no
16	Alarm	Notification of an alarm	no	no
17	Attribute value change	Notification of an autonomous attribute value change	no	no
18	Test	Request a test on a specific managed entity	yes	no
19	Start software download	Start a software download action	yes	yes
20	Download section	Download a section of a software image	yes/no	no
21	End software download	End of a software download action	yes	yes
22	Activate software	Activate the downloaded software image	yes	yes
23	Commit software	Commit the downloaded software image	yes	yes
MT	Type	Purpose	AK	Inc MIB data sync.
24	Synchronize Time	Synchronize the time between OLT and ONT	yes	no
25	Reboot	Reboot ONT or circuit pack	yes	no
26	Get next	Get the latched attribute values of the managed entity within the current snapshot	yes	no
27	Test result	Notification of test result that is initiated by "Test"	no	no
28	Get current data	Get current counter value associated with one or more attributes of a managed entity	yes	no
NOTE – The "Download section" action is only acknowledged for the last section within a window. See Appendix I.2.15/G.984.4.				

언뜻 보면 어려워 보이지만, 하나씩 살펴보면 매우 간단하다. 예를 들어 MT = 4 라고 하는 것은 Create 을 의미하고, 이는 OLT 가 ONU 에 뭔가를 만들라고 하는 것이다.

다음은 6 번 bit 인데, 이를 이해하기 위해 Create operation 의 동작을 살펴보자.



그림과 같이 OLT 가 ONU 에 CREATE Request 를 보내고, ONU 는 MIB 에 생성한 이후에 OLT 에게 CREATE Response 를 보내 transaction 이 완료되는 형태이다.

Message Type 만 보면 CREATE Request 와 CREATE Response 가 동일하기 때문에, 이를 구분하여 주기 위하여 AK bit 를 사용한다. AK 가 0 이면 Request 이고, 1 이면 Response 가 되는 것이다. 규격에서는 Response 와 Acknowledgement 를 동일한 의미로 함께 사용하고 있다.

7 번 bit 인 AR 은 Acknowledgement Request 의 약자로, Acknowledgement 즉 response 의 요청여부를 나타낸다. AR 이 1 이면 ONU 는 Response 를 보내고, 0 이면 보내지 않는다.

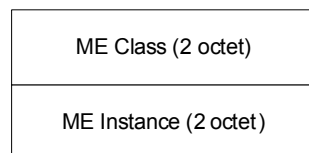
마지막으로 8 번 bit 인 DB 는 Reserved bit 로 항상 0 을 사용한다.

■ **Device Identifier**

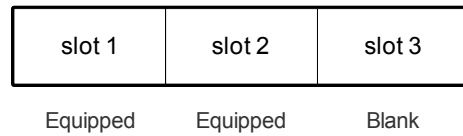
Device Identifier 는 0x0A 로 fix 되어 있다.

■ **Message Identifier**

Message Identifier 는 4 byte 으로 되어 있으며 이는 아래와 같이 ME (Managed Entity)의 Class 와 Instance ID 로 구분된다.



우선 ME Class 와 Instance 의 개념을 설명하기 전에 실제 시스템의 모습과 MIB 의 형태에 대해서 알아보자.



위 시스템을 보면 Slot 이 3 개 있고, 1 번과 2 번에는 Circuit pack, 즉 유닛이 실장 되어 있다. 3 번은 비워져 있는 상태이다. 이를 MIB 에서 나타내면 아래와 같이 표현된다.

Class	Instance		
2. ONT Data	0x0000		
5. Cardholder	0x0001	0x0002	0x0003
6. Circuit pack	0x0001	0x0002	

참고로, MIB 은 Instance 의 모임을 말하는 것으로 ONU 내부에는 오른쪽 Box 부분만 구현된다.

2. ONT Data 는 나중에 설명하기로 하고, 5. Cardholder 를 보면 실제 시스템과 같이 Instance 가 3 개 존재한다. 6. Circuit pack 은 Cardholder 에 실장 되는 유닛을 말하는데, 시스템과 같이 Instance 가 2 개 존재함을 알 수 있다. Slot 3 는 Blank 이다.

이번에는 MIB 이 위와 같이 구성되어 있다는 가정아래, OLT 에서 ONU 의 정보를 읽어 오는 경우를 살펴보자. 그 중에서 Slot 1 에 어떤 Circuit Pack 이 실장 되어 있는지를 알아보자. OLT 에서 Slot1 의 Circuit pack 에 해당하는 Instance 에 GET Request 를 보내게 된다. 그렇다면 Circuit pack 의 Instance 0x0001 을 지칭은 어떻게 하는가? 간단하다. ME Class ID 인 6 과 ME Instance ID 인 0x0001 을 주면 해결된다. 이 두 가지를 합친 것이 바로 Message Identifier 이다.

쉽게 말해, Message Identifier 는 ONU 의 MIB 상에서 하나의 ME Instance 를 지칭하기 위한 것이다.

**■ Message Contents**

Message Contents 부분은 Message Type 과 AK field 에 따라서 그 형태가 달라진다. 예를 들어 Create Request, Create Response, Get Request, Get Response 별로 다른 형태를 가지게 된다. 이 부분은 양이 많기 때문에 5.Operations에서 알아보도록 하자.

**■ OMCI Trailer**

OMCI Trailer 는 8 byte 으로써, 앞의 4 byte 은 0x00000028 을 사용하고, 나머지 4 byte 은 32-bit CRC 를 사용한다. 여기의 CRC 는 ITU-T Rec. I.363.5 임으로 주의하도록 한다.

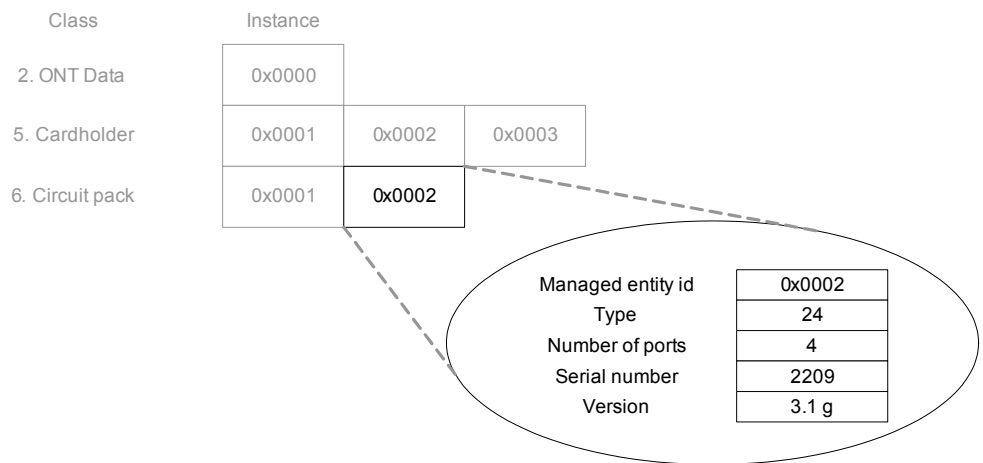
이로써 OMCI 의 53 byte 의 각 부분별의 의미에 대해 이해가 되었을 것으로 예상된다. 결국 중요한 부분은 TID, Message Type, Message Identifier, Message Contents 의 4 부분에 불과하다.

## 4. MIB Description

이제는 ONU 의 MIB 정보에 대해 조금 더 자세히 알 필요가 있다.

MIB description 이란 MIB 이 어떤 식으로 구성이 되어 있는지에 대한 표준 규격을 말한다. 이름은 거창하지만 하나씩 살펴보면 아주 쉽다.

전 Section 에서 살펴본 MIB Instance 의 내부를 열어보면 아래의 그림과 같이 되어 있다.



Instance 는 하나 이상의 attribute 로 구성이 되어 있다. 위의 그림에서는 6. Circuit Pack 의 Instance 0x0002 의 내부의 모습을 본 것인데 attribute 가 5 개 표시되어 있다. 동그라미 안에서 왼쪽에 있는 것은 attribute 이름이고 오른쪽에 있는 것은 attribute 의 값이다. 예를 들어 2 번째 Type 의 값은 24 이고, 3 번째 Number of ports 는 4 이다. Type 에서 24 의 의미는 10/100 Base-T type 임으로, 결과적으로 이 Circuit pack 은 4-port 용 10/100 Base-T Card 임을 알 수 있다.

이와 같이 각 Class 별로 Attribute 의 정보를 서술해 놓은 것이 바로 MIB description 이다. 아래는 Circuit Pack class 의 MIB description 이다. 이해를 돕기 위해 주요 부분만 포함하였다. 이제, 각 부분을 살펴보기로 하자.

### 9.1.6 Circuit Pack

This managed entity models a circuit pack that is equipped in an ONT slot. For ONTs with integrated interfaces, this managed entity may be used to distinguish available types of interfaces.

...

*Relationships*

An instance of this managed entity is contained by an instance of the cardholder managed entity

*Attributes*

**Managed entity id:** This attribute uniquely identifies each instance of this managed entity.

Its value is the same as that of the cardholder managed entity containing this circuit pack instance. (R, Set-by-create (if applicable)) (mandatory) (2 bytes)

**Type:** This attribute identifies the circuit pack type. This attribute is a code as defined in table 9.1.5-1/G.984.4. The value 255 means unknown or undefined, i.e., the inserted circuit pack is not recognized by the ONT or is not mapped to an entry in table 9.1.5-1/G.984.4. In the latter case, the equipment ID attribute may contain inventory information. Upon autonomous ME instantiation, the ONT sets this attribute to 0 or to the type of the circuit pack that is physically present. (R, Set-by-create (if applicable)) (mandatory) (1 byte)

**Number of ports:** This attribute is the number of access ports on the circuit pack. If the port mapping package-G is supported for this circuit pack, this attribute should be set to the total number of ports of all types. (R) (optional) (1 byte)

**Serial number:** The serial number is unique for each circuit pack. Note that the serial number may contain the vendor id and/or version number. For integrated ONTs, this value is identical to the value of the serial number attribute of the ONT-G managed entity. Upon creation in the absence of a physical circuit pack, this attribute comprises all spaces. (R) (mandatory) (8 bytes)

**Version:** This attribute is a string that identifies the version of the circuit pack as defined by the vendor. The value 0 indicates that version information is not available or applicable. For integrated ONTs, this value is identical to the value of the version attribute of the ONT-G managed entity. Upon creation in the absence of a physical circuit pack, this attribute comprises all spaces. (R) (mandatory) (14 bytes)

**Vendor id:** This attribute identifies the ...

**Administrative state:** This attribute locks (1) and unlocks (0) the functions ...

**Operational state:** This attribute indicates whether or not the circuit pack is capable of

...

...

*Actions*

**Get, set**

**Create, delete:** Optional, only when plug-and-play is supported.

**Reboot:** Reboot the circuit pack.

*Notifications***Attribute value change**

Number	Attribute value change	Description
1..6	N/A	
7	Op state	Operational state change
8..14	N/A	
15..16	Reserved	

**Alarm**

Number	Alarm	Description
0	Equipment alarm	A failure on an internal interface or failed self test
1	Powering alarm	Fuse failure or failure of DC/DC converter
2	Self test failure	Failure of circuit pack autonomous self test
3	Laser end of life	Failure of transmit laser imminent
4	Temperature yellow	No service shutdown at present, but the circuit pack is operating beyond its recommended range.
5	Temperature red	Service has been shut down to avoid equipment damage. The operational state of the affected PPTPs indicates the affected services.
6..207	Reserved	
208..223	Vendor specific alarms	Not to be standardized

(ITU-T G.984.4 p61)

■ **Class**

9.1.6 Circuit Pack 은 Class 의 이름이다. 그 아래에는 관련 설명이 있다.

사실 Class 이름은 사람을 위한 것이며 OLT 와 ONU 간의 통신에는 Class ID 만을 사용한다. Class ID 는 아래의 table 에 정의되어 있는데, 예제인 Circuit Pack 은 6 번임으로 알 수 있다.

Table 11-2/G.984.4 – Managed entity identifiers

Managed entity class value	Managed entity
1	ONT <sub>B-PON</sub>
2	ONT data
3	PON IF line cardholder
4	PON IF line card
5	Cardholder
6	Circuit pack
7	Software image
8	UNI <sub>B-PON</sub>
9	TC Adapter <sub>B-PON</sub>
10	Physical path termination point ATM UNI
11	Physical path termination point Ethernet UNI
12	Physical path termination point CES UNI
13	Logical Nx64 kbit/s sub-port connection termination point
14	Interworking VCC termination point
15	AAL1 profile <sub>B-PON</sub>

#### ■ Relationships

Relationships 부분은 다른 Class 와의 관계를 나타내는 것으로, Cardholder class 와 연결된다고 적혀 있다.

#### ■ Attributes

Attributes 부분에는 각 attribute 에 대한 description 이 포함되어 있다. 모든 Class 에는 Managed entity id 가 포함되며, 항상 맨 처음에 서술되어 있다.

Attribute 에는 이름과 설명이 포함되어 있다. Attribute 의 이름 또한, Class 의 이름 과 마찬가지로 사람을 위해서 만든 것이다. Class 와 다른 점은 attribute 의 encoding 은 MIB description 의 순서대로 하는 것이다. 예를 들어, Circuit Pack class 의 첫 번째 attribute 는 Type 이고, 두 번째 attribute 는 Number of Ports 이다. (참고로 Managed Entity id 는 Message Identifier field 를 사용한다) Type 1 byte 가 오고, 그 다음 Number of Ports 의 1 byte 가 오게 된다. 이를 해석하기 위해서는 OLT 와 ONU 가 동일한 MIB description 규격을 가지고 있어야 한다. 이에 는 길이, optional 여부 등이 포함되며 각각의 설명은 아래에 있다.

참고로, OMCI 에서 사용할 수 있는 attribute 의 최대 수는 16 개이기 때문에 MIB description 의 attribute 는 항상 16 개를 넘지 않는다. 물론 Managed Entity id 는 제외하고이다.

각 Attribute 의 마지막 부분을 보면 3 개의 괄호 ()가 항상 표시되어 있다.

이 부분 → (R, Set-by-create (if applicable)) (mandatory) (2 bytes)

우선 맨 마지막의 괄호는 해당 attribute 의 길이를 나타낸다. (2 byte)는 attribute 의 길이가 2 byte 라는 것을 의미한다. 그 앞의 (mandatory)는 이 attribute 가 항상 존재해야 한다는 것을 의미한다. 이에 반해 (optional)은 존재할 수도 있고 존재하지 않을 수도 있음을 의미한다. 이 부분에 대해서는 Optional attribute 부분의 설명을 참조한다.

### Permission

(R, Set-by-create (if applicable))부분이 permission 인데 여기에서 (if applicable)은 큰 의미가 없기 때문에, (R, Set-by-create)으로 이해하면 된다. R and Set-by-create 이라는 의미인데, 여기서 R 은 Read 할 수 있음을 나타내고, Set-by-create 은 ME Instance 를 Create 시 포함하는 attribute 를 의미한다. W 는 Write 할 수 있는 attribute 이다. 예를 들어, Read 를 할 수 없는 attribute 에 대해 OLT 가 GET request 를 내리면, ONU 는 error 를 return 하게 된다. Set 을 할 수 없는 attribute 에 대해 SET request 를 내리면 마찬가지로 error 를 return 하게 된다. 비슷하게 CREATE Request 에는 모든 Set-by-create attribute 가 포함되어야 한다.

### ■ Actions

해당 Class 에서 사용할 수 있는 Operation 을 의미한다.

위의 예제는 GET, SET, CREATE, DELETE, REBOOT operation 이 가능하다는 것을 알 수 있다. 각 Operation 의미는 5.Operations부분에서 소개된다.

### ■ Notifications

Notification 에는 3 가지가 있다 - AVC, Alarm, TCA.

AVC 는 attribute 의 값이 변경된 경우에 OLT 에게 이를 알려주는 것이다. Alarm 은 경보이고, TCA 는 PM 의 값이 threshold 를 넘어설 때 발생하는 notification 을 말한다. 위의 예제를 보면 Circuit Pack class 는 AVC 와 Alarm 을 지원함을 알 수 있다.

자 이제, AVC, Alarm, TCA 에 대해서 조금 더 자세히 알아보자.

**AVC**

AVC 는 attribute 의 값이 변경되는 경우에 이를 OLT 에 알려주는 Notification 이다. (OLT 에 의해서 변경된 경우에는 이를 알려주지 않는다) Circuit Pack 예제의 MIB description 을 보면 7 번째의 attribute 인 Operational State 의 값이 변경되는 경우에 이를 OLT 에게 알려주도록 되어 있다. AVC 의 내용은 변경된 attribute list 와 attribute 의 새로운 값을 보내준다. Attribute list 는 mask 를 사용하는데 16 개를 표현하기 위하여 총 2 byte 를 사용한다. 아래 그림과 같이 해당 bit 가 1 이면 관련 attribute 가 포함됨을 의미한다.

이 attribute mask 는 차후에 설명될 GET, SET, CREATE 에서도 동일하게 사용된다.

Byte	Bit							
	8	7	6	5	4	3	2	1
1	Attribute 1	Attribute 2	Attribute 3	Attribute 4	Attribute 5	Attribute 6	Attribute 7	Attribute 8
2	Attribute 9	Attribute 10	Attribute 11	Attribute 12	Attribute 13	Attribute 14	Attribute 15	Attribute 16

예를 들어, 0x80 0x00 은 Attribute 1 번이라는 것을 의미하고, (1000 0000 0000 0000 b)  
 0x81 0x10 은 attribute 1 번, 8 번 그리고 12 번이라는 것을 의미한다.  
 (1000 0001 0001 0000 b)

**Alarm**

Alarm 은 ONU 경보의 발생과 해제를 OLT 에게 알려주는 역할을 한다. Circuit Pack 의 경우를 보면 0 부터 5 까지 총 6 개의 Alarm 이 정의되어 있으며, 208 부터 223 까지는 비 표준화된 부분으로 배정되어 있다. Alarm 의 PDU 는 아래 그림과 같이 28 byte 를 사용하여 총 224 개의 alarm 의 ON/OFF 를 표시할 수 있다.

Byte	Bit							
	8	7	6	5	4	3	2	1
1	Alarm 0	Alarm 1	Alarm 2	Alarm 3	Alarm 4	Alarm 5	Alarm 6	Alarm 7
2	Alarm 8	Alarm 9	Alarm 10	Alarm 11	Alarm 12	Alarm 13	Alarm 14	Alarm 15
...								
28	Alarm 216	Alarm 217	Alarm 218	Alarm 219	Alarm 220	Alarm 221	Alarm 222	Alarm 223

Alarm PDU 에는 Alarm Sequence Number 가 포함되는데, 처음에는 1 부터 시작하여 발생할 때마다 1 씩 증가시킨다. 255 가 넘는 경우에는 1 부터 다시 시작하게 된다.

**TCA**

마지막인 TCA 는 다음 예제에서 설명하기로 한다. TCA 를 이해하기 위해서는 PM 을 먼저 알아야 하기 때문이다.

**PM**

여기서 MIB description 의 예제를 하나 더 살펴보기로 하자.

PM (Performance Monitoring) 관련된 GEM port performance monitoring history data 이다.

### 9.2.6 GEM port performance monitoring history data

This managed entity collects performance monitoring data associated with a GEM port network CTP. Instances of this managed entity are created and deleted by the OLT. For a complete discussion of generic PM architecture, refer to clause I.1.9/G.984.4.

#### *Relationships*

An instance of this managed entity is associated with an instance of the GEM port network CTP managed entity.

#### *Attributes*

**Managed entity id:** This attribute uniquely identifies each instance of this managed entity.

Through an identical ID, this managed entity is implicitly linked to an instance of the GEM port network CTP. (R, Set-by-create) (mandatory) (2 bytes)

**Interval end time:** ... This attribute identifies the most recently finished 15-minute interval.

(R) (mandatory) (1 byte)

**Threshold data 1/2 id:** This attribute points to an instance of the threshold data 1 managed entity that contains PM threshold values. Since no threshold value attribute number exceeds 7, a threshold data 2 ME is optional. (R, W, Set-by-create) (mandatory) (2 bytes)

**Lost packets:** This attribute counts background GEM frame loss. It does not distinguish between packets lost because of header bit errors or buffer overflows; it records only loss of information. (R) (mandatory) (4 bytes)

**Misinserted packets:** This attribute counts GEM frames misrouted to this GEM port. (R) (mandatory) (4 bytes)

**Received packets:** This attribute counts GEM frames that were received correctly at the monitored GEM port. (R) (mandatory) (5 bytes)

**Received blocks:** This attribute counts GEM blocks that were received correctly at the monitored GEM port. (R) (mandatory) (5 bytes)

**Transmitted blocks:** This attribute counts GEM blocks originated by the transmitting end point (i.e., backward reporting is assumed). (R) (mandatory) (5 bytes)

**Impaired blocks:** This severely errored data block counter is incremented whenever one of the following events takes place: the number of misinserted packets reaches its threshold, the number of bipolar violations reaches its threshold, or the number of lost packets reaches its threshold. Threshold values are based on vendor-operator negotiation. (R) (mandatory) (4 bytes)

*Actions*

**Create, delete, get, set**

**Get current data** (optional)

*Notifications*

**Threshold crossing alert**

Number	Threshold crossing alert	Threshold value attribute # (Note)
0	Lost packets	1
1	Misinserted packets	2
2	Impaired blocks	3

다행히도 Circuit Pack 의 예제와 크게 다르지 않다. 차이점이라고 하면 단지 PM 이라는 특성뿐이고, 이제부터 이를 하나씩 알아보도록 하자. PM 이라는 것은 port 의 상태를 말한다. 예를 들어 수신 받은 packet 의 수와 같은 것이다. PM history 란 이 port 의 상태의 기록을 저장하는 곳이다. 일반적으로 15 분 단위로 내용을 PM history 에 저장하게 된다.

**ID**

PM history 는 port 당 하나씩 존재하기 때문에 PM history 의 ID 는 port 의 ID 를 그대로 사용한다. PM history 의 ID 를 보고 어떤 port 의 것인지 알 수 있도록 말이다. 문서에서는 GEM port network CTP 라는 거창한 용어를 사용하고 있는데, 이는 결국 port 를 의미하는 것이다.

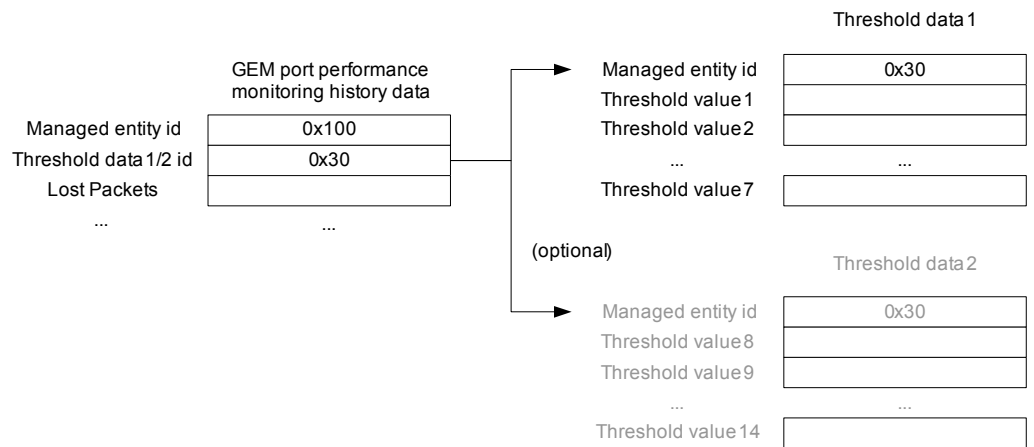
**TCA (again)**

Notifications 부분을 보면 지난 예제에서 없었던 TCA 가 포함되어 있다. Table 의 모양은 Alarm 과 비슷한데 Threshold value attribute # 라는 것이 추가되어 있다. 자세한 의미를 설명하기 전에 우선 TCA 가 무엇인지 이해를 해보자. 매우 간단하다.

Lost packets attribute 는 port 에서 잃어버린 packet 의 수를 count 하는 곳이다. 예를 들어, 이 count 가 1000 을 넘어서는 경우에 OLT 가 알고 싶다고 하자. 이때, 이런 일을 부탁할 수 있는 곳이 바로 TCA 이다. 1000 이라는 것을 threshold 라고 부르는 것이고, TCA 는 바로 이 threshold 를 넘어섰다는 의미의 경보이다. (Threshold Crossing Alarm)

**그러면 threshold 값은 MIB instance 의 어디에 있는 것인가?**

우선 table 의 0 번째 TCA 를 살펴보자. Lost Packets 라고 쓰여 있다. 그 다음 column 에 Threshold value attribute #라고 되어 있는데, threshold 값이 아닌 index 같은 것이 있는 것 같다. 도대체 threshold 값은 어디 있는 것인가?  
아래의 그림을 보면 이해가 될 것이다.



**Threshold data 1 / Threshold data 2**

예제 Class MIB description 의 attribute 3 번째를 보면 Threshold data 1/2 id 라는 attribute 가 보인다. 그림에서와 같이 이 attribute 는 새로운 Class 인 Threshold data 1 과 Threshold data 2 의 instance 를 가리킨다. 바로 이곳이 TCA 에 사용되는 Threshold 값을 저장되어 있는 곳이다. TCA notification table 의 1 번째 entry 의 Threshold value attribute #에서 1 이라는 의미는 바로 threshold 의 값이 Threshold value 1 attribute 에 있다는 것을 의미한다. 아하.

Class 가 2 개로 나누어져 있는 이유는 CREATE 으로 한번에 설정할 수 있도록 attribute 가 7 개로 한정되어 있기 때문이다.

이제 다시 TCA 로 돌아가자.

흥미롭게도 TCA 의 전송은 Alarm 의 PDU 를 그대로 사용한다. 이유는 TCA 와 Alarm 을 동시에 사용하는 ME 가 거의 없기 때문이다. 혹 있다면 Number 를 겹치지 않게 사용하게 된다. 이러한 이유로 일반적으로 발생만 있는 TCA 와는 달리 OMCI 에서는 TCA 의 발생/해제의 상태가 존재하게 된다. TCA 는 15 분 단위로 Clear 된다.

이로써 MIB description 을 어떻게 읽는지 살펴보았다. 자신이 생겼다면 G.984.4 의 Chapter 9 를 섭렵해보기 바란다.

## 5. Operations

이제 각 Operation 에 대해 알아보기로 하자.

OMCI Protocol 에서 잠시 미루었던 Message Contents 부분에 대한 설명도 포함된다. Appendix II.2 Message Layout 부분을 함께 보면서 아래의 내용을 확인하는 것을 추천한다.

### ■ CREATE

CREATE operation 은 OLT 가 ONU 에 ME instance 를 생성할 때 사용한다. 각 ME Class 의 설명을 보면, ONU 에서 자동으로 create 되는지, OLT 가 create 을 하는지 여부가 설명되어 있다. 마찬가지로 Class 의 Actions 부분을 보면 CREATE 이 포함되어 있다. 만약, CREATE action 이 없는 instance 가 CREATE 요청을 받으면 ONU 는 error 를 return 한다.

CREATE request 를 살펴보면, ME Class ID 와 Instance ID 로 생성할 instance 를 지정한다. 아울러 Create 에 필요한 attribute 의 내용을 보내주게 된다. ME Class 의 설명을 보면 Set-by-Creat e 이라고 되어 있는 부분이 있는데 이 attribute 의 값을 모두 포함하여 전달하게 된다. 설령 그 attribute 가 optional 이고 ONU 에서 구현하지 않는 것이라 할지라도 포함되어야 하며, 이러한 경우 해당 attribute 의 영역은 0 으로 채워지게 된다.

CREATE request 를 받은 ONU 는, 해당 ME Instance 를 생성한 후에 CREATE response 를 보낸다.

Message contents	14	0	0	0	0	x	x	x	x	Result, reason 0000 = command processed successfully 0001 = command processing error 0010 = command not supported 0011 = parameter error 0100 = unknown managed entity 0101 = unknown managed entity instance 0110 = device busy 0111 = instance exists
	15									Attribute execution mask (attributes 1-8), used with 0011 encoding: 0 = Attribute ok, 1 = illegal value attribute,
	16									Attribute execution mask (attributes 9-16), used with 0011 encoding: 0 = Attribute ok, 1 = illegal value attribute,
	17-45	0	0	0	0	0	0	0	0	Padding

CREATE Response 의 message contents 부분을 보면 Result field 가 있다. 성공인 경우에는 0000 을 사용하여 정상적으로 생성되었다는 것을 나타낸다. 나머지는 error 를 나타내는데, 그 의미는 아래와 같으며 다른 Operation 에서 동일하게 사용한다.

**[ Error Result ]**

**Command processing error**

아래의 'Command not supported'나 'Parameter error'에 해당하지 않는 error 가 발생하였음을 의미한다.

**Command not supported**

CREATE action 이 지원되지 않는 ME instance 를 생성하는 경우이다. Get, Set, Delete, Test, Alarm, Reboot, Get Next 등의 command 도 동일하게 적용된다.

**Parameter error**

수신한 Request 의 정보가 올바르지 않은 경우이다. 예를 들어, Create request 에서 사용된 attribute 의 값이 정상적이지 않은 경우가 이에 해당된다. Error 가 발생하는 경우 어떤 attribute 가 원인인지를 표시하기 위하여 Attribute execution mask 가 사용되며, 이에 대한 설명은 바로 아래에 있다.

**Unknown managed entity**

지원되지 않는 ME Class ID 를 사용한 경우이다.

**Unknown managed entity instance**

Create 이외의 경우에는, 지정한 ME Instance ID 가 존재하지 않는다는 것을 의미한다. 예를 들어, 존재하지 않는 ME Instance 의 attribute 를 Get 하는 경우가 이에 해당한다. CREATE request 의 경우에는 Valid 하지 않은 Class 의 ID 를 생성하고자 하는 경우에 이 error 가 발생하게 된다.

**Device busy**

ONU 의 처리 용량이 넘쳐 timeout 이 발생하는 경우이다.

**Attribute(s) failed or unknown**

Create 에서는 사용하지 않고, Get, Set, Get current 에서 발생하는 error 이다. 구현되지 않은 optional attribute 를 access 하는 경우에 해당 error 가 발생한다. 혹은 invalid 한 값을 set 하는 경우에 발생한다. 이 error 가 발생하는 경우에, 16 개중 어떤 attribute 가 문제인지 알 수 없기 때문에 그 정보를 알려주는 Optional attribute mask 와 Attribute execution mask 를 사용한다. Attribute Execution mask 는 바로 아래에서, Optional attribute mask 는 Set operation 부분에서 설명이 이어진다.

**Instance exists**

Create 하려는 Instance 가 이미 존재하는 경우이다. Create 이외에는 사용하지 않는다.

**[ Attribute execution mask ]**

Attribute execution mask field 는 Result code 가 0011 혹은 1001, 즉 parameter error 혹은 attribute(s) failed or unknown 인 경우에만 의미를 가진다. CREATE request 의 경우에 Parameter error 는 수신한 attribute 의 값이 잘못되었다는 것을 의미하며 Attribute execution mask 는 어떤 attribute 가 잘못 되었는지는 알려준다. 2 byte 를 사용하고 있는데 이는 총 16 개의 attribute 를 표시하기 위함이다. 문제가 있는 attribute 의 bit 1 의 값을 가지고, 정상인 attribute 는 0 을 가지게 된다. 동시에 여러 개의 attribute 에 문제가 있는 경우에는, 문제 있는 모든 bit 에 1 이 표시된다.

### ■ DELETE

Delete 는 OLT 가 ONU 의 instance 를 삭제하는 경우에 사용한다. ME Class 에 Delete action 이 포함되어 있어야 정상적으로 동작한다.

### ■ SET

Set 은 OLT 가 ONU 의 Attribute 값을 변경하는 operation 이다. 다른 Operation 과 마찬가지로 instance 를 지정하기 위하여 ME Class 와 Instance ID 를 사용한다. Attribute 의 Set 은 하나 이상의 attribute 를 동시에 할 수 있다. 이를 위하여 어떤 attribute 들을 Set 하는지 표현할 수 있도록 Attribute mask 2 byte 를 사용한다. 1 로 설정된 attribute 의 값들이 다음 부분에 순서대로 encoding 된다.

ONU 는 해당 Instance 를 찾아 원하는 attribute 의 값을 변경하고 SET response 를 보내준다. 만약, 문제가 발생하면 Create 과 비슷하게 Result code 에 error code 가 전달된다.

#### Optional Attribute mask

Optional attribute 중에서 ONU 에서 구현되지 않은 attribute 를 SET 하는 경우에는 attribute(s) failed or unknown 의 error 가 발생하며, 어떤 attribute 가 문제인지를 표시하기 위하여 Optional Attribute mask 를 사용한다. 문제가 있는 attribute 의 bit 가 1 로 표시된다.

Attribute Execution mask 는 Create 에서와 동일하게 OLT 로부터 invalid 한 값을 받은 경우에 이를 알리기 위하여 사용한다.

### ■ GET

Get operation 은 ONU 의 attribute 값을 읽어올 때 사용한다. 요청하는 attribute 의 list 는 Attribute mask 를 사용하여 설정하게 된다. ONU 는 요청 받은 attribute 들의 값을 GET response 에 담아 보내준다.

그런데, 혹시 OLT 에서 attribute mask 를 잘못 설정해서 Attribute 들의 전체 크기가 GET response 에서 담을 수 있는 Data 보다 큰 경우에는 어떻게 하는가?

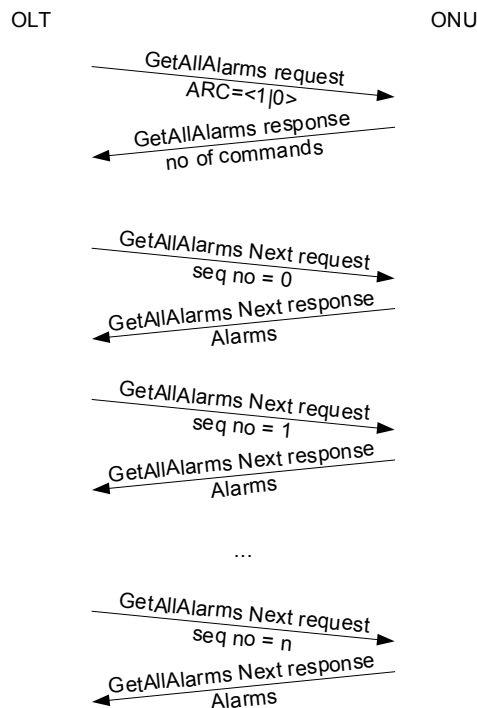
두 가지 방법이 있다.

ONU 는 담을 수 있는 만큼의 Attribute 들의 값만 GET response 로 보내주는 것이

한 가지 방법이다. OLT 는 Get request 의 attribute mask 와 비교하여 어떤 attribute 가 빠져 있는지 알 수 있으므로 필요한 부분을 추가로 요청하여 처리하게 된다. 다른 방법은 error 를 return 하는 방법이다. OLT 는 자신이 보낸 attribute mask 에 문제가 있는 것을 인식하고, 이를 줄여서 GET request 를 다시 보내는 방식이다. 선택은 구현하는 사람에게 맡겨져 있다.

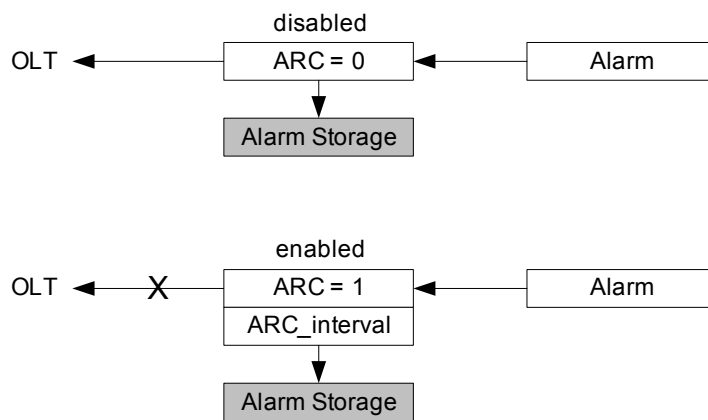
■ GET ALL ALARMS

GetAllAlarms 는 OLT 가 ONU 의 Alarm 상태에 대한 Sync 를 다시 맞추는 데 사용한다. OLT 는 ONU 의 경보에 대한 상태를 항상 유지하고 있어야 하는데, 이 상태를 Alarm Sequence number 를 사용하여 유지하게 된다. ONU 는 경보를 보내면서 Alarm Sequence number 를 증가시킨다. OLT 도 수신할 때마다 값을 증가시키는데 통신 장애 등의 이유로 이 Alarm Sequence number 가 달라지는 경우에, Alarm 의 sync 가 깨졌다고 판단 한다. 이러한 경우에 OLT 는 ONU 의 alarm 정보를 다시 가져오기 위한 동작을 한다. 이때 사용하는 것이 GetAllAlarms 이다. OLT 는 아래의 그림과 같이, ONU 에게 GetAllAlarms request 를 보낸다. 이때 ARC 를 선택할 수 있다. ARC 에 대해서는 아래를 참조한다. ONU 는 GetAllAlarms response 로 commands 의 개수를 보내준다. Command 의 개수는 발생한 경보가 있는 ME instance 의 개수를 의미한다. OLT 는 이 개수만큼 GetAllAlarms Next Request 를 보내고 ONU 는 각 sequence no 에 해당하는 Alarm 의 정보를 보내준다.



**ARC**

ARC 란 Alarm Report Control 의 약자로서, 아래의 그림과 같이 ARC 가 enabled 되면 Alarm 이 발생해도 OLT 에게 보고하지 않는 기능을 말한다. ARC 는 ARC\_interval 과 함께 사용하는데 ARC timer 라고 이해하면 된다. 1~254 분까지 설정 가능하며, timeout 이 발생한 다음, Alarm 이 없는 상태가 되면 ARC 가 0 으로 자동 변경된다. ARC\_interval 을 0 으로 설정하면 Alarm 이 clear 될 때, ARC 를 0 으로 변경한다. ARC\_interval 이 255 이면 timeout 기능을 사용하지 않는다는 의미이다.



ONU 는 GetAllAlarms 에서 ARC 에 관계없이 모든 Alarm 을 요청할 수 있기 때문에, 이런 경우를 위해 Alarm 의 상태를 늘 유지하고 있어야 한다. (Alarm Storage 의 용도)

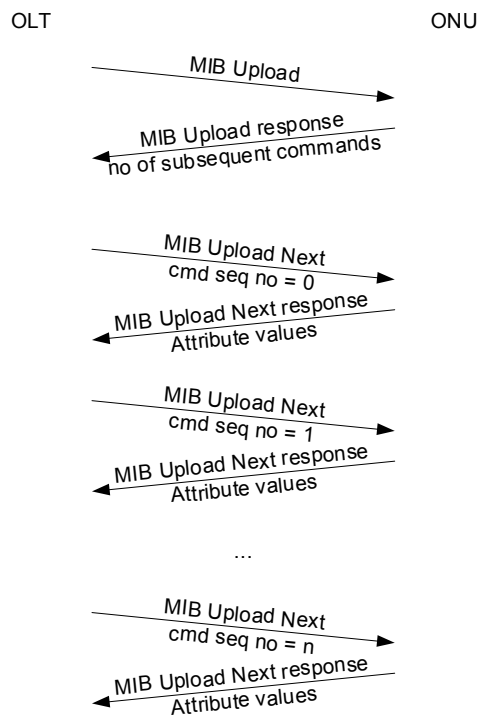
**Alarm Sequence number 와의 관계**

Alarm Sequence number 는 alarm 이 발생할 때마다 증가하는 sequence number 를 말한다. 이 Alarm Sequence number 는 GetAllAlarms request 를 받으면 1 부터 다시 reset 하여 사용한다. 이렇게 하는 이유는 간단하다. GetAllAlarms 를 수행하고 있는 동안에 새로운 Alarm 이 발생할 수 있는데, OLT 는 새로 발생한 Alarm notification 을 GetAllAlarms 가 끝난 후에 Alarm MIB 에 적용시켜야 하기 때문이다. 그러기 위해서는 어떤 notification 부터가 새로 시작한 것인지 알아야 하는데, 그런 이유로 Alarm Sequence number 를 다시 1 부터 사용하도록 하는 것이다.

■ **MIB Upload**

MIB Upload 는 OLT 가 ONU 의 MIB 정보를 모두 가져올 때 사용한다. ONU 의 모든 Instance 를 OLT 에게 알려주는 것이다. (보고하지 않도록 되어 있는 Class 는 제외이다. 예, SIP config portal)

Protocol 은 간단하다. OLT 는 ONU 에게 Instance 가 총 몇 개이냐고 물어본다. 예를 들어, ONU 가 10 개라고 답을 하면, OLT 는 1 번째 것 줘, 2 번째 것 줘, 하면서 10 번을 물어보면서 모든 Instance 의 값을 읽어오게 된다. 이러한 Protocol 이 아래의 그림이다.



그런데, 그림을 보면 MIB Upload response 에서 no of instances 대신에 no of subsequence commands 라고 표시되어 있다. 왜 다른 용어를 사용했을까? 이것 또한 간단하다. MIB Upload Next response 에 담을 수 있는 크기는 26 byte 인데, instance 의 attribute 들의 총 합이 이것보다 크다면 여러 번에 나누어서 담아야 한다. 그래서 MIB Upload Next command 를 불러야 하는 숫자라는 의미로서 no of instances 대신에 사용하는 것이다.

### ■ MIB Reset

MIB Reset 은 ONT 의 MIB 상태를 초기화 하는 것이다. OLT 가 생성한 Instance 들은 모두 초기화 된다. 일반적으로 OLT 는 최초에 ONU 에게 MIB Reset 를 보낸 후에, MIB Upload 를 수행하는 방식을 사용한다.

### ■ ALARM

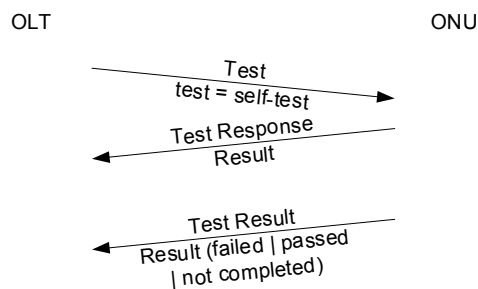
Alarm 은 ONU 에서 OLT 로 경보의 발생 및 해제를 알려주는 notification 이다. 자세한 설명을 이미 ME Class 부분에서 하였기 때문에 PDU 를 이해하는데 큰 문제는 없을 것으로 보인다.

### ■ Attribute Value Change (AVC)

AVC 는 Attribute 값의 변경을 ONU 가 OLT 에게 알려주는 notification 이다. 자세한 설명을 이미 ME Class 부분에서 하였기 때문에 PDU 를 이해하는데 큰 문제는 없을 것으로 보인다.

### ■ TEST

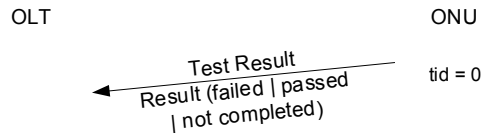
TEST operation 은 말 그대로 시험을 할 때 사용한다. 예를 들어 Circuit Pack 이 정상적으로 동작하는지 여부를 시험하거나, ping test 를 하거나, 특정 port 의 voltage 를 측정하는 경우이다. Test 는 주로 OLT 가 시작하는데, 간혹 ONU 가 스스로 시험을 하는 경우도 있다. OLT 가 시작하는 시험의 procedure 는 아래와 같다.



OLT 는 ONT 에게 TEST request 를 보낸다. 이때 시험의 종류를 적어서 보낸다. ONU 는 TEST Response 에서 정상적으로 시험이 시작되었음을 알린다. 시험의 결과는 차후에 TEST Result Notification 으로 전달된다. Test PDU 의 format 은 현재 3 가지를 사용하고 있다. ONT-G 및 Circuit Pack 용, IP host config data 용 그리고 POTS UNI and PPTP ISDN UNI 용이다. TEST Response 는 format 에 상관없이 동일

한 것을 사용하지만 TEST Result notification 은 test 종류에 따라서 5 가지 형태를 띈다. 자세한 내용은 규격을 참고하길 바란다.

위와 같이 일반적으로 Test 는 OLT 의 요청에 의해 시작된다. 그러나, self-test 의 경우에는 ONU 에 의해 스스로 시작될 수도 있는데, 아래와 같이 ONU 가 시험을 하고 그 결과를 OLT 에게 알려주는 방식을 사용한다.



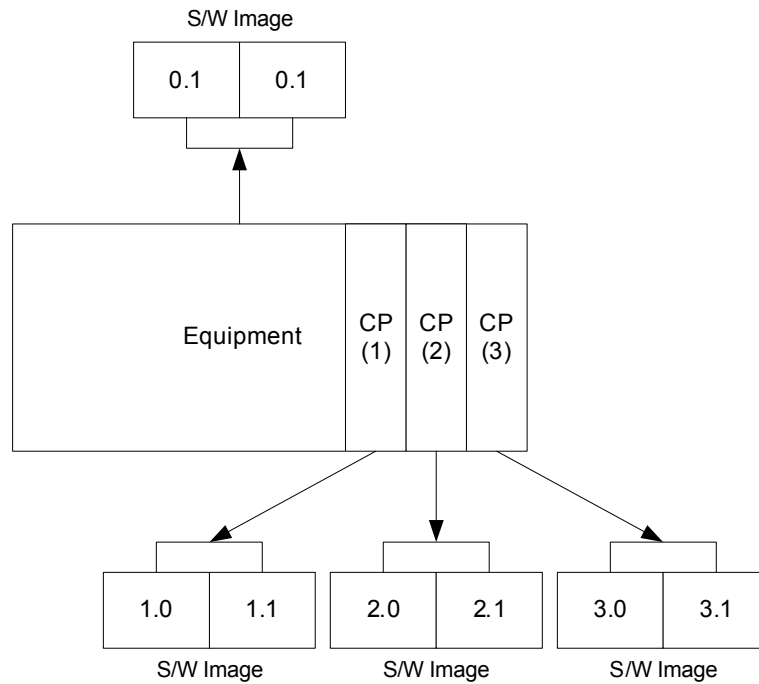
TEST Result notification 의 format 은 동일한 것을 사용하기 때문에 ONU 가 스스로 발생시켰다는 의미를 표시하기 위하여 TID (Transaction correlation identifier)에 0 을 사용한다. OLT 에서 시작된 Test 의 경우에는 TEST Request 에 사용한 TID 를 TEST Result notification 에서 그대로 사용하게 된다.

■ SW Image Download

S/W Image

0	1
---	---

S/W Image 는 2 개가 존재한다고 가정하고 있다. 0 번과 1 번이 있는데, 이중에 하나를 현재 수행하고 있다는 가정이다. 독특한 개념은 현재 수행하고 있는 Image 와 Reboot 이 되었을 때 수행할 Image 의 선택이 따로 되어 있다는 점이다. 공식 용어를 살펴보면, 현재 수행하고 있는 것을 Active 라고 하고, Reboot 시 수행할 Image 를 Commit 이라고 부른다. 현재 실행 중인 Image 와 다음에 실행할 Image 가 다를 수 있는 것이다. 여기에 Valid 라는 간단한 개념이 하나 더 사용되는데, Image 의 상태가 정상 여부를 나타낸다. 특이한 점은 ONU 의 memory 가 작다는 가정아래 Download 가 시작되면 Image 의 상태는 invalid 로 된다는 사실이다. (구현상으로 보면 8K 단위로 Flash 에 구우라는 이야기로 들린다)



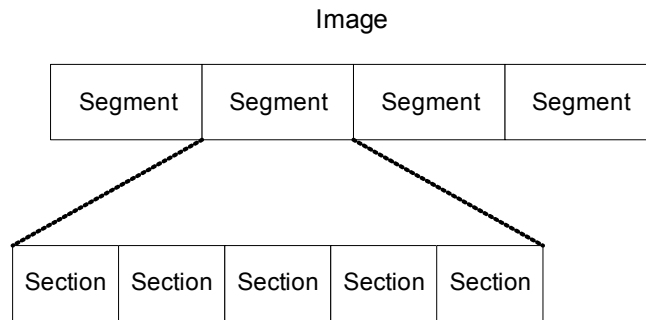
위의 그림은 Equipment 에서 사용하는 S/W Image 를 보여준다. 상단에 Equipment(ONT-G)에서 사용하는 Main S/W Image 는 물론이고, 하단에는 각 Circuit Pack 의 S/W Image 도 download 할 수 있도록 되어 있다. Circuit Pack 은 단위로 각 2 개의 S/W Image Instance 를 사용하고 있다. Circuit Pack 이 3 개임으로 2 X 3 = 총 6 개의 S/W Image ME Instance 를 사용하고 있다. Instance ID 는 2 byte 를 사용하는데, 앞의 것이 ONT-G, Slot no 와 같이 장소를 의미하고, 뒤의 것이 Instance 의 번호를 의미한다. (0 혹은 1)

**Multiple target download**

S/W Image download 는 일반적으로 한번에 하나의 instance 를 받도록 되어 있다. 그러나, 종류가 같은 Circuit Pack 의 경우에는 동일한 S/W Image 를 반복적으로 down 받아야 하는 문제가 있다. 이러한 문제를 해결하기 위하여 한번만 S/W Image 를 전송하고 여러 개의 Circuit Pack 을 동시에 Upgrade 할 수 있는 기능을 제공한다. 이것을 바로 Multiple download 라고 한다. (하나만 받는 것은 single download 이다) 최대 9 개의 Instance 를 동시에 Upgrade 를 할 수 있으며, 동일한 Circuit Pack 의 instance 0 과 instance 1 을 동시에 받을 수도 있도록 되어 있다. Multiple download 는 ONU 의 Optional feature 이다.

이제 S/W Download Protocol 을 살펴보도록 하자.

Protocol 이 복잡해 보이지만, 사용하는 개념을 이해하면 의외로 간단하다.



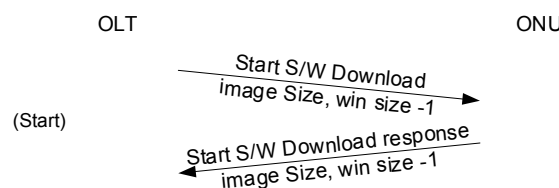
Protocol 에서, Image 는 여러 개의 Segment 로 이루어 지며, Segment 는 다시 여러 개의 Section 으로 이루어진다고 가정한다. 각 Section 의 크기는 Download Section PDU 인 31 byte 으로 fix 되어 있다. 이에 반해 몇 개의 Section 을 모아 하나의 Segment 를 만들지는 OLT 와 ONU 가 Negotiation 을 통하여 Run-time 상에 정하게 된다. 이를 Window Size 혹은 Section Size 라고 부른다. 최대 값은 256 를 넘을 수 없게 되어 있다. 참고로, 각 Section 의 크기는 31 byte 임으로 Segment 의 최대 크기는  $31 \times 256 = 7936$  byte 이다 (규격에는 8K = 8192 라고 되어 있다)

Image 의 전송은 Segment 단위로 일어난다. OLT 는 하나의 Segment 의 전송이 완료된 것을 확인하고 다음 Segment 의 전송을 시작한다. Segment 내의 Section 은 처음부터 순차적으로 ONU 로 보내진다. ONU 는 ACK 를 맨 끝의 section 에 대해 한 번만 하는데, 이때 실패를 하게 되면 해당 Segment 의 Section 을 처음부터 다시 보내는 방식을 사용한다. 결국 Packet 의 재전송은 Segment 내에서만 일어난다.

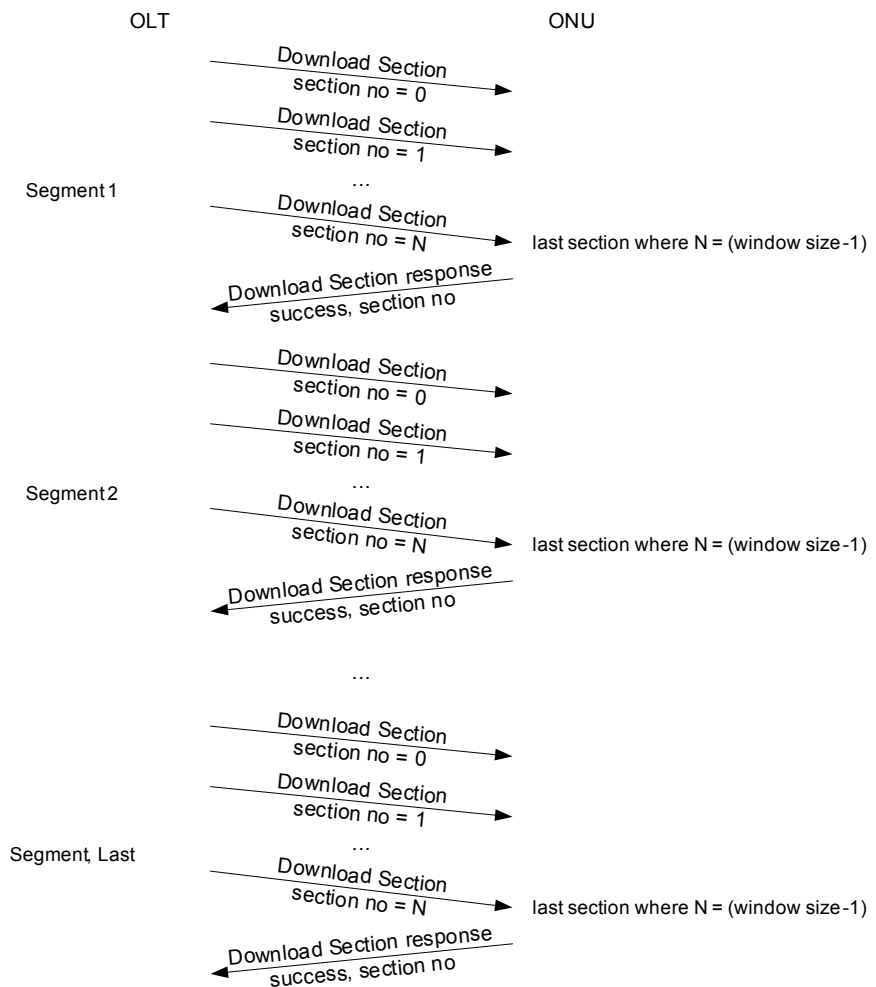
모든 Segment 의 내용이 ONU 에 전달되면 OLT 는 End Software download 를 보내 마무리를 한다. CRC-32 의 값이 정상적이면 ONU 는 성공하였음을 OLT 에 알려준다.

자 이제 Protocol 을 조금 더 자세히 알아보자.

Protocol 은 크게 3 단계로 나뉜다 - Start 단계, Download 단계, End 단계.



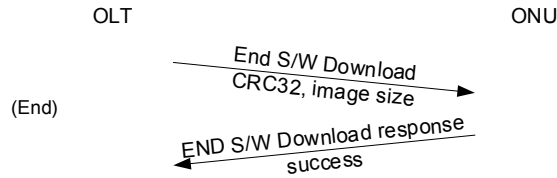
우선 Start 단계이다. Start S/W Download 를 ONU 에 보낸다. 이때 Image 의 크기와 Windows Size 를 보낸다. 이 windows size 가 ONU 의 처리 능력보다 크다면 ONU 는 더 작은 숫자를 OLT 에게 전달하고, OLT 는 이것을 windows size 로 사용한다. 이 Procedure 를 windows size negotiation 이라고 부른다. 그림에 보면 win size 대신에 win size -1 을 사용하는데, 이것 또한 간단하다. win size 가 0 이라는 것은 사용하지 않기 때문에 PDU encoding 시에는 0~255 대신에 1~256 을 사용한다는 의미이다.



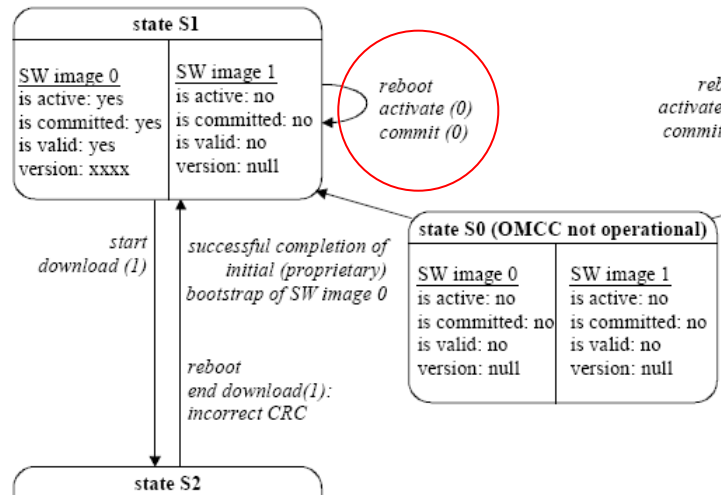
다음은 Download 단계이다.

Segment 1 부터 마지막 Segment 까지 순서대로 보낸다. 각 Segment 단위를 보면 Window size 만큼 data 를 잘라서 Download section PDU 에 실어 ONU 로 보낸다. ONU 는 맨 마지막 section data 를 수신하면, 성공여부를 Download section response 로 알려준다. 성공하면 OLT 는 다음 Segment 를 보낸다. 혹시라도 실패하게 되

면, OLT 는 해당 Segment 의 section 을 처음부터 다시 보내게 된다. 이렇게 하여, 마지막 Segment 를 다 보내면 End 단계로 접어든다.



End 단계에서는 OLT 가 End S/W Download 를 보내게 된다. 이때 CRC32 와 image size 를 보내주는데, ONU 는 계산한 CRC32 와 수신한 값을 비교하여 성공 여부를 OLT 에게 알려준다. 성공하면 ONU 는 해당 Image 를 실제로 upgrade 하게 된다.



참고) Figure 9.1.6-1/G.984.4 의 그림은 S/W Download 관련 State diagram 을 나타낸 것이다. 개인적으로 이 그림에서 이해가 잘 되지 않는 부분이 있었는데, 그것은 위의 동그라미 친 action 부분이였다. 처음에는 이것이 reboot 하고 activate(0) 하고 commit(0)한다는 '순서'의 의미라고 생각을 해서 이해가 안되었었는데, 결국 이는 reboot 혹은 activate(0) 혹은 commit(0)인 OR 를 나타내는 것이였다.

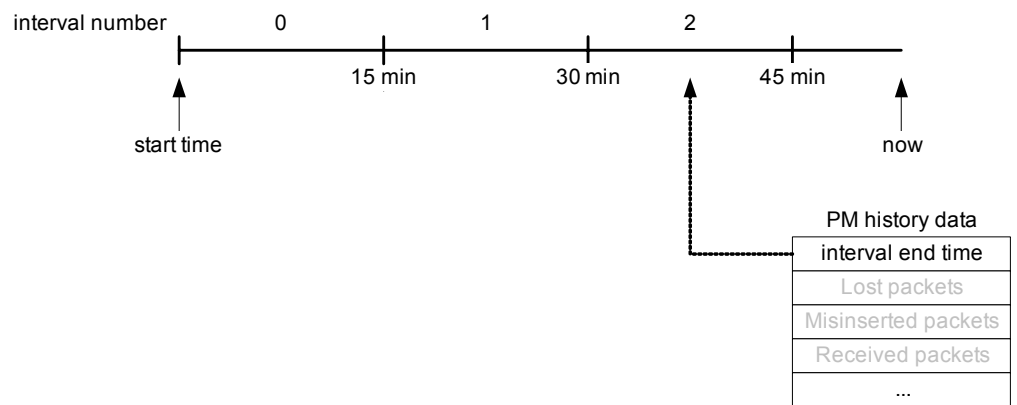
■ **Activate Image**

OMCI 의 S/W Image 에서 Active 란 의미는 현재 실행 중인 S/W Image 를 말한다. Activate Image 는 선택한 instance 의 image 를 실행하라는 의미이다. Image 는 당연히 valid 상태이어야 한다.

■ **Commit Image**

Commit 는 다음에 reboot 혹은 초기화 시에 실행할 Image 를 나타낸다. Commit Image 는 선택한 instance 의 image 를 다음에 실행하도록 설정하라는 의미이다. Image 는 당연히 valid 상태이어야 한다.

■ **Synchronize Time**



Synchronize Time 하면 장비간의 시간을 맞추는 것으로 들리지는 실제로는 PM 과 관련된 내용이다. ONU 는 15 분 단위로 PM history 를 저장하게 된다. 그러면 언제부터 15 분을 측정하는 것인가 하는 의문이 생긴다. 이를 PM 의 Start time 이라고 부르는데, 그 기준을 정하는 것이 바로 Synchronize Time 이다. OLT 가 ONU 에게 Synchronize Time 을 보내면 그때부터 15 분 단위로 PM 을 저장하게 된다.

**Interval end time (Interval number/count)**

PM Class 의 attribute 를 보면 interval end time 이 항상 포함되어 있다. 이것은 counter 들의 내용이 언제의 15min 것인지 나타내는 것이다. 위의 그림에서 보면 now 가 현재 시점일 경우 interval end time 은 바로 전 15min 을 가리키게 된다. 적혀진 숫자는 각 15 분을 구분하기 위해 사용하는 interval count 의 값이다. Synchronize Time 을 ONU 가 받으면 interval count 를 0 으로 clear 시킨다.

■ **Reboot**

Reboot 은 Equipment(ONT-G)나 Circuit Pack 을 재 시작하라는 요청이다. ONT 는 Reboot Response 를 보내고 해당 equipment 혹은 Circuit Pack 을 재 시작 한다.

## ■ Get Next

Get Next 를 이해하기 위해서는 우선 Table attribute 의 특성을 이해해야 한다.

### Table attribute

Table attribute 는 MIB description 의 attribute 길이에 8N 과 같이 되어 있는 attribute 를 말한다. Attribute 이름에는 일반 attribute 와 구분하기 위하여 항상 table 을 포함하고 있다.

Table attribute 는 OMCI operation 에서 허용하는 Data 의 크기보다 커질 수 있기 때문에, (예를 들어 Get operation 의 Max 는 25 byte 이다) 약간의 다른 방법으로 attribute 의 값을 access 한다.

### [ GET ]

GET 하는 방법은 간단하다. GET operation 의 mask 에 Table attribute 가 포함되어 있는 경우에는 attribute 의 값을 전달하는 대신에 총 길이를 return 한다. 예를 들어 8N 의 table 에 5 개의 entry 가 있다면  $8 \times 5 = 40$  이 return 된다. OLT 는 GET-NEXT 를 반복적으로 불러 29 byte 만큼씩 data 를 받는 방식으로 Table attribute 의 값을 읽어 들인다. OLT 는 ONU 의 memory 소진을 막기 위하여 하나의 GET-NEXT 가 종료된 후에, 다른 table 의 GET-NEXT 를 시작하는 것을 권장한다.

### [ SET ]

SET 하는 방법은 약간 더 복잡하다. Table 을 변경할 수 있는 operation 은 3 가지가 있는데, 이는 Modify, Add, Delete 이다. 그러나 OMCI Operation 에는 SET 밖에 없기 때문에 조금 독특한 방법으로 처리한다. Permission 이 W 가 포함되어 있는 table attribute 의 경우를 보면, 항상 index 가 맨 처음에 포함되어 있다. 이를 통해서 table 의 entry 를 선택하게 된다. 즉 Modify 에 해당한다. Add/Delete 가 필요한 경우에는 index 와 함께 혹은 그 다음의 attribute 내에 add/delete 여부를 적을 수 있도록 되어 있다. 가끔씩 add 와 modify 가 함께 되어 있는 경우도 있는데, 사용한 index 의 entry 가 존재하면 modify 이고, 그렇지 않으면 add 로 인식하는 방식이다. 그러나 이러한 SET 의 방식은 표준이라기 보다는 ME 마다 다르게 구현되어 있기 때문에 MIB description 을 참조하여 하나씩 구현해야 한다.

### [ CREATE ]

Table attribute 는 CREATE operation 으로 생성할 수 없기 때문에 Set-by-create

permission 을 가지지 못한다.

[ AVC ]

AVC 의 경우에 Table attribute 는 mask 에 표시만 하고 값은 보내지 않는다. OLT 가 값이 필요한 경우에는 GET operation 을 사용하여 값을 가져가도록 되어 있다. (GET-NEXT 이용)

아래는 MAC bridge port filter table data 의 MAC filter table 의 예제이다,

**MAC filter table:** This attribute lists MAC destination addresses associated with the bridge port, each with an allow/disallow forwarding indicator for traffic flowing out of the bridge port. In this way, the upstream traffic is filtered on the ANI-side bridge ports, and the downstream traffic is filtered on the UNI-side bridge ports. Each entry contains:

- The entry number, an index into this attribute list (1 byte)
- Filter byte (1 byte)
- MAC address (6 bytes)

The bits of the filter byte are assigned as follows:

Bit	Name	Setting
1	Filter/forward	0: forward 1: filter
2..7	Reserved	0
8	Add/remove	0: remove this entry (set operation) 1: add this entry

Upon ME instantiation, the ONT sets this attribute to an empty table.

One OMCI set message can convey a maximum of three table entries. However, OMCI does not provide robust exception handling when more than one entry is included in a set command, and multiple entries per set operation are not recommended.

(R, W) (mandatory) (8N bytes, where N is the number of entries in the list)

맨 끝의 8N 은 8 X N 의 table attribute 라는 것은 나타낸다. Attribute 의 이름에도 권장대로 table 이 포함되어 있다. (MAC filter table) Entry 를 구성하는 8 byte 를 살펴보면, SET operation 을 위해 첫 번째 byte 를 index 로 사용하고 있다. Add/Delete 를 위하여 2 번째 Filter byte 의 8 번 bit 를 사용하고 있음을 알 수 있다.

### Get Next 의 Protocol

위에서 간단하게 설명한 Get Next 의 Procedure 를 알아보면 아래와 같다.

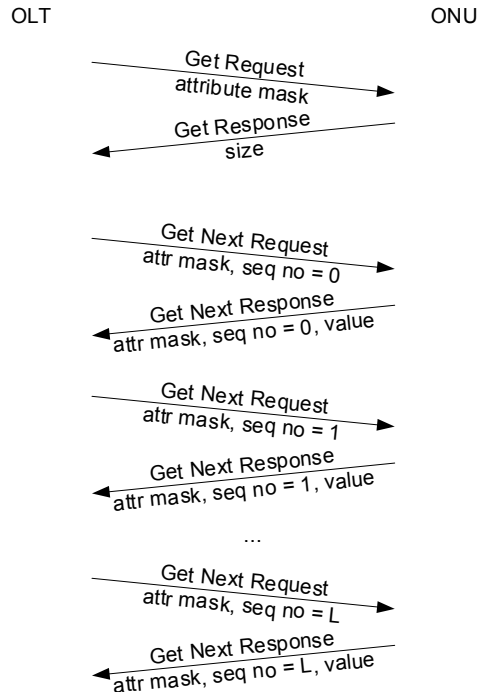
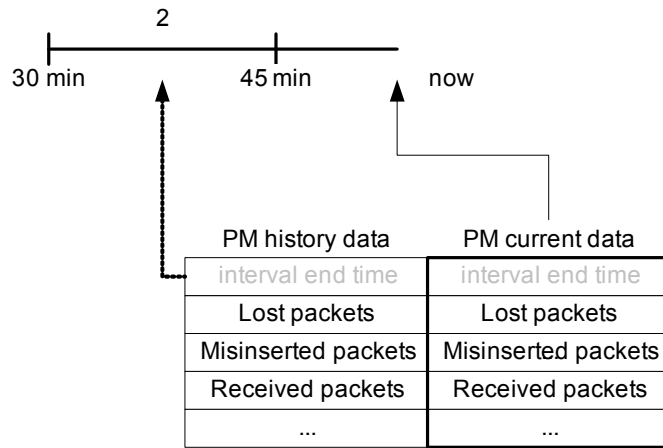


Table attribute 의 값을 읽으면 attribute 값 대신에 길이를 보내 준다. 길이는 항상 4 byte 이다. OLT 는 반복적으로 Get Next Request 를 sequence no 0 부터 순차적으로 보내고 ONU 는 그에 대한 값을 보내준다. Get Next 를 수행하고 있는 동안 Table attribute 의 값이 변할 수 있기 때문에 ONU 는 이를 복사하여 사용한다. Protocol 상에는 attribute mask 가 포함되어 있어, 여러 개의 table attribute 를 동시에 가져 올 수 있도록 되어 있으나, ONU 입장에서 보면 이를 복사하여야 되는 부담 때문에 동시에 여러 개의 table attribute 를 가져오는 것은 권장하지 않고 있다.

■ Get Current



Get Current 는 PM 과 관련된 Operation 이다. Synchronize Time 에서 본 PM history data 를 한번 더 살펴보자. 그림에서와 같이 PM history data 는 15 분 전의 counter 값을 가지고 있다. 이에 반해 현재의 counter 값 또한 가지고 있을 수 있다. 바로 PM current data 로 표시된 오른쪽 부분인데, OMCI 에서는 PM current data 를 access 하기 위해 class 나 instance 를 추가로 만들지 않고 Get current 라는 operation 을 사용하여 이를 해결하고 있다. 쉽게 말해 PM history data history 에 대해 Get current operation 을 내리면 PM history data 가 아닌 PM current data 를 돌려주게 되는 방식이다. 일반적으로 Get current 는 optional 기능이다.

이로써 OMCI 의 모든 operation 에 대한 이해를 마쳤다.

어느 정도 자신감이 생겼다면, 흩어져 있는 생각들의 조각을 잘 맞추어서 커다란 그림을 그려보기 바란다. 조각을 잘 이해했다고 생각하더라도 전체로 만들면 말이 안 되는 경우가 종종 있기 때문이다.

## 6. Special concepts

OMCI 전반에서 사용되는 독특한 개념들에 대해서 알아보기로 하자.

### ■ Optional Attribute 의 의미

Optional Attribute 란 implementation 시에 구현해도 되고 안 해도 되는 attribute 를 의미한다. 어떤 장비에서는 존재하고, 어떤 장비에서는 존재하지 않을 수 있는 attribute 를 말하는 것이다. 마찬가지로 ONU 장비 내에서도 시간이나 상황에 따라서 attribute 가 존재할 수도 존재하지 않을 수도 있다.

여기서는 이러한 Optional attribute 의 구현여부가 OLT-ONU 의 연동에 어떤 영향을 미치는지 살펴보자. 가장 기본적인 것은 Optional attribute 의 구현여부가 Attribute mask 에서 사용하는 Attribute number 에 대한 영향이다. 결론은 간단하다. 아무 영향이 없다는 것이다.

Class			Instance	
Mandatory		1	Mandatory	
Optional		2	Optional	
Mandatory		3	Mandatory	

예를 들어, 어떤 Class 가 왼쪽의 그림과 같이 3 개의 attribute 를 가지고 있다고 하자. Mandatory attribute 이후에 Optional attribute 가 있으며 그 다음에 Mandatory attribute 가 있다. Optional field 를 구현한 경우에 MIB 은 오른쪽과 같은 모습을 가지게 된다. Attribute mask 에서 사용하는 Attribute number 는 각각 1, 2, 3 을 사용한다.

그렇다면 Optional field 를 구현하지 않은 경우에 3 번째의 attribute number 에 어떤 영향이 있을까? 아래의 그림을 보면 간단하다.

1	Mandatory	1	Mandatory
2	Mandatory	2	
3		3	Mandatory

왼쪽의 경우에는 Attribute number 가 2 번이 된다는 가정이고, 오른쪽은 그대로 3 번이 된다는 가정이다. 어떤 것이 답일까?

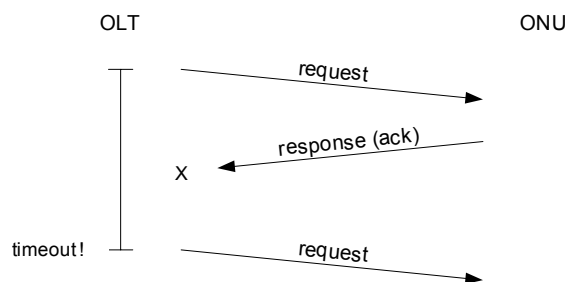
정답은 오른쪽이다. Attribute 의 구현 여부에 상관없이 Attribute 의 number 는 변경되지 않는다.

이런 이유로 CREATE request 때 Optional Set-by-Create attribute 부분은 구현여부와 관계없이 항상 사용되어서 보내진다. GET/SET request 의 경우에는 구현하지 않는 Optional attribute 를 요청하면 fail 이 발생하는 것과 대조된다.

■ **Retransmission**

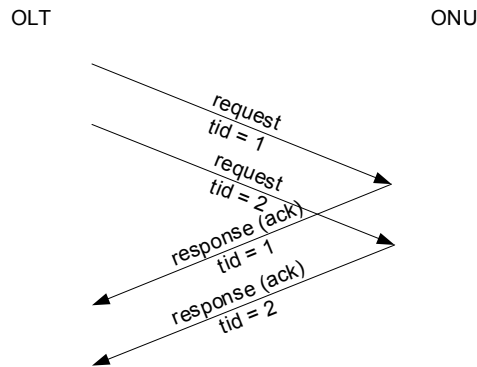
OMCI 에서 재전송은 간단한 구조로 되어 있다.

OLT 가 ONU 에게 Request 를 보내면, ONU 는 Acknowledge 인 Response 를 보낸다. OLT 가 주어진 시간 내에 Response 를 받지 못하면 OLT 는 다시 Request 를 보내게 된다.



Retransmission timeout 이나 몇 번의 재전송을 할지는 모두 implementation-dependent 한 부분이다.

추가적으로 OLT 는 response 가 도착하기 전에 새로운 request 를 보낼 수 있도록 설계 되어있다. 결국, 여러 개의 request 가 동시에 ONU 에서 처리될 수 있다는 의미이다. 그런데 이런 경우 수신한 response 가 어떤 request 의 결과인지 알 수 없기 때문에, 이를 구분하기 위하여 TID (Transaction correlation identifier)를 사용한다. OLT 는 각 request 별로 다른 TID 를 보내고, ONU 는 response 에 수신 받은 TID 를 돌려주게 된다.



그러면 재전송이 일어나는 경우에는 OLT 는 TID 에 어떤 값을 사용할까? 이전의 TID 를 사용할 수도 있고, 새로운 TID 를 부여할 수도 있을 것이다. 답은 CREATE 이나 SET 과 같이 MIB 을 변경하는 operation 의 경우를 고려해보면 알 수 있다.

만약 새로운 TID 를 사용하는 경우에서의 CREATE operation 을 살펴보자. Loss 가 발생하여 CREATE Response 가 없어졌다고 가정해보자. OLT 는 timeout 발생 후 CREATE 을 다시 보내게 되는데, ONU 입장에서는 두 번 받았기 때문에 실패라고 결과를 보내게 된다. OLT 는 CREATE 이 실패되었다고 생각하고 ONU 는 CREATE 이 성공되었다고 생각하게 되는 것이다.

이러한 상황을 막기 위하여 OLT 는 재전송 시에 항상 이전의 TID 를 그대로 사용하게 된다. ONU 는 이미 처리한 Request 의 TID 이면 실제로 처리는 하지 않고 이전에 보낸 Response 를 보내주는 방식으로 동작한다. 이를 위해서 ONU 는 실행한 TID 와 Response 의 내용을 일정 기간 동안 기억하고 있어야 한다. ONU 에서는 중복 실행을 막아주는 중요한 부분이다.

■ MIB data sync

OLT 는 ONU 의 MIB 정보를 가지고 있어야 하고, 이를 항상 ONU 의 MIB 과 동일하게 유지하고 있어야 한다. 그러면, OLT 는 어떻게 MIB 이 동기화 되었다는 것을 알 수 있을까?

ONT Data 의 MIB data sync attribute 의 값이 그 역할을 한다.

MIB data sync attribute 는 OLT 에 의해서 Instance 가 생성되거나 삭제되는 경우에 증가하게 된다. 예를 들어, OLT 가 하나의 Instance 를 생성하면 자신이 가지고 있는 MIB data sync 값을 증가시킨다. 마찬가지로 ONU 는 Instance 생성 한 후 자신의 MIB data sync 값을 증가시키게 된다. OLT 는 ONU 의 MIB data sync 값을 읽어

와서 이 두 값이 다른 경우에는 MIB 의 sync 가 깨진 것으로 보고 MIB sync procedure 인 MIB upload 를 수행하게 된다.

MIB data sync 값은 OLT 에 의해 attribute 의 값이 변경되는 경우에도 증가하며, S/W Download 의 시작과 정상 종료시점에서도 증가하게 된다. MIB 이 변경되어도 MIB data sync 값이 증가하지 않는 경우도 있는데, ONU 에서 자동으로 instance 가 생성 혹은 삭제되는 경우가 이에 해당한다. 마찬가지로 attribute 의 값이 ONU 에서 자동으로 변경되는 경우에는 MIB data sync 값이 증가하지 않는다. 단, 이 경우에 AVC 는 발생하게 된다. 문제는 이 AVC 가 Loss 되는 경우인데, 규격에서는 OLT 가 변할 수 있는 attribute 를 주기적으로 polling 을 해서 알아 내야 된다고 되어 있다. 조금 성의 없는 규격이다.

MIB data sync 는 1 에서부터 255 까지 증가한 후에 다시 1 로 돌아온다. 맨 처음 초기화 되었을 때 0 을 가지고 있으며, 그 외의 경우에 0 은 오류의 의미로 사용된다.

## 7. MIB description 의 작성 및 확장

OMCI MIB description 을 추가하거나 Private 용도로 확장하는 경우에 그 의미와 주의할 점에 대해서 알아보기로 하자.

### ■ OMCI 규격 만들 때 제한사항

Attribute의 개수는 총 16 개를 넘을 수 없다.

Managed Entity Id 는 제외이다.

각 Attribute의 최대 크기는 25 를 넘을 수 없다.

GET Operation 을 할 수 없으니까.

만약 넘는다면 Table Attributes 를 사용해야 한다.

구분을 위해 attribute name 에 table 을 사용한다.

Set-by-Create된 attribute의 총 길이의 합계는 32 를 넘을 수 없다. (ME id제외)

Create operation의 한계 때문이다.

Table Attribute는 Set-by-Create을 할 수 없다

Table Attribute의 각 entry의 W attribute의 총 크기는 30 를 넘을 수 없다.

Set operation 의 한계 때문이다.

### ■ Attribute 확장과 연동의 관계

그렇다면 OMCI 기본 규격에서 attribute 를 확장하여 사용하면 어떤 연동의 문제가 발생할까?

OLT 에서 확장하는 경우에 ONU 에서 확장하는 경우로 구분할 수 있겠다.

물론 둘 다 확장하는 경우도 배제 할 수 없으나, 이 부분은 나중에 고려해 보자.

OLT 가 확장하는 경우에는 GET/SET request 에서 추가된 attribute mask 를 사용할

가능성이 높다. ONU 에는 이 attribute 가 존재하지 않기 때문에 error 가 발생하게 된다. OLT 는 attribute 의 error 에 대한 어느 정도 예상을 하고 있기 때문에 대처가 가능할 것이다.

ONU 가 확장되는 경우에는 MIB upload 시에 해당 attribute 가 OLT 에 보고되게 된다. OLT 에서 이를 수신하여 무시할 가능성이 높으므로 문제가 되지 않는다. 그 외에는 OLT 의 확장과 같이 큰 문제가 없어 보인다.

문제가 되는 부분은 OLT 와 ONU 가 동일한 attribute 를 확장하는 경우이다. 이와 유사한 것이 private 하게 확장한 attribute 의 부분이 OMCI 규격이 확장되면서 사용되는 경우이다. 이러한 경우가 발생하면 동일한 attribute number 에 대해서 길이 및 의미가 달라지기 때문에 호환성의 문제가 발생하게 된다.

이를 방지하기 위해서 권장하는 방법은 Private attribute 를 16 번부터 앞으로 사용하는 방법이다. OMCI 의 차후 규격이 해당 Attribute 의 장소를 사용할 때까지는 안전한 방법이다. 하지만 이런 방법보다는 Private Class 를 추가하여 사용하는 방법이 연동에는 더 바람직한 방법이라고 생각한다.

이로써 OMCI 의 간단한 설명이 완료되었습니다.

더 깊이 공부하실 분은 규격을 보면, 더 많은 정보를 얻으실 수 있을 겁니다. 도움이 되기를 바라고, 끝까지 읽어주셔서 감사합니다.

## Appendix A. 문서정보

### A.1 문서이력

- 문서목적  
OMCI 규격의 이해를 돕는다.
  
- 작성자  
제임스 / [james@novonetworks.com](mailto:james@novonetworks.com)
  
- 도움주신 분  
김 흥원, 김 용석, 최 형주, 엄 지성, 백 승우, 김 인태, 김 정미  
그 외 크고 작은 도움을 주신 분들께 감사 드립니다.
  
- 작성이력  
최초작성 : 2008-10

## Appendix B. References

ITU-T G.984.1  
Gigabit-capable Passive Optical Networks (GPON): General characteristics

ITU-T G.984.4  
Gigabit-capable Passive Optical Networks  
(G-PON): ONT management and control interface specification

omcstk 1.1  
OMCI agent toolkit manual, novo networks