Direct Lake Messaging – 3rd party protocol for PLM™ & LM™ Series

Purpose

The Direct Lake Messaging provides an Ethernet 3rd party protocol suitable for integration with third party control and monitoring applications such as AMX®/Crestron®/etc. This document provides all of the reference information required to implement a control interface for an end user's custom application.

Support contact

support@lakeprocessing.com

History

Rev.	Date	Comment
1.0	2009-09-23	First public release
2.0	2010-10-11	Added PLM firmware 2.74 and LM firmware 0.32 commands.
2.1	2010-11-05	Added a Release note section with Known issues and Disclaimer
2.2	2010-11-19	Fixed PSU Mains Input section.
2.3	2010-11-22	Improved information about PLM 20000Q unique features.
2.4	2010-12-01	Added probe parameters to Dev.Dante.BreakIn
2.5	2010-12-08	Fixed typo in Amp Info 9.2.2.
2.6	2010-12-21	Fixed typo in Mod.Out.Delay section.
2.7	2011-01-18	Made *Sensitivity* an optional parameter of Dev.Router.InputTypSel.
2.8	2011-02-21	Updated FAQ

Release notes

1.1 Known Issues

- *[PLM & LM]* The **Mod.In.Mute and Mod.Out.Mute** command does not synchronize correct LED indications to the front panel user interface. The correct mute status is valid on the output(s).
- *[PLM & LM]* If any random input is sent instead of the correct "1" or "0", that syntax will be interpreted as a "0".

1.2 Disclaimer

- Via the DLM it is possible to set illegal parameters for many settings. It is advisable to implement user limitations in a 3rd party UI. Example of possible illegal settings:
    - Have an input routing priority list with analog higher prioritized than for example AES or Dante.
    - Set gain, delay etc. parameters outside the normal borders.

Index

2 Version requirements

This document applies to;

PLM Series products firmware versions:

DLM protocol version 1:	2.58 and above, or until further notice.
DLM protocol version 1 and 2:	2.74 and above, or until further notice (i.e. firmware 2.74 and above are fully backwards compatible with DLM protocol version 1). All mains input monitoring is PLM 20000Q unique features. **Some** commands need other firmware, be sure to read the exact requirement in the heading of each command.

LM Series products firmware versions:

DLM protocol version 2:	0.32 and above, or until further notice. **Some** commands need other firmware, be sure to read the exact requirement in the heading of each command.

3 Important Communication information

The DLM protocol can be used between a software application and a device using Ethernet UDP/IP messaging.

3.1 Description

Unicast can fail when having more than one network adapter. The code will bind a UDP socket to the specific adapter's IP address specifically and this will guarantee that the packets originate from that interface. Still, routing may decide to route the packets through the other interface based on destination address and costs associated with each interface. Type 'route print' in the command prompt in order to see the routing.

3.2 Workaround

3.2.1 Limit the number of network adapters

If possible, use only one network adapter. This will force the routing to use it.

3.2.2 Edit routing table (advanced)

Use the 'route' command in the command prompt.

4 Usage of ports

The following ports shall be used by the application for communication:

UDP Port	Purpose
6004	Application listening port for receiving packets
6015	Device destination port for transmitting packets

5 Packet format

In order to implement external control the controlling application must transmit and receive properly formed UDP packets to and from a network of devices.

Every packet contains a header, a variable length payload and footer:

Header Payload Footer

5.1 Packet Header

The following table depicts the structure of the packet header:

Field	Name		Comment
lSrcIDHi	**Source ID**	**4 Bytes**	**Hi and Lo 32 bits of the Src unique 64 bit ID**
LSrcIDLo	**Source ID**	**4 Bytes**	
lDestIDHi	**Destination ID**	**4 Bytes**	**Hi and Lo 32 bits of the Dest unique 64 bit ID**
lDestIDLo	**Destination ID**	**4 Bytes**	
sSrcClass	**Source Class**	**2 Bytes**	**Host class for this application**
sDestClass	**Destination Class**	**2 Bytes**	**Module class for this application**
sLength	**Length**	**2 Bytes**	**Total length of packet including header and footer in bytes. Maximum length is 560 bytes.**
sPacketType	**Packet Type**	**2 Bytes**	**(see 5.1.6 below)**
lMsgID	**Message ID**	**4 Bytes**	**A unique number for every packet. If the sender of the packet does not require a response, set this to 0xFFFFFFFF (-1)**

5.1.1 Source ID

The Source ID identifies the sending (source) application and can be any value. The device verifies that the Message ID is incremented for each packet from the same source.

5.1.2 Destination ID

In order to send Ethernet packets to a specific device on the network, each device's unique hardware ID must be determined. See "Addressing a Device" in section 7.

When building an application, the hardware ID will be required for use in the destination ID of the packet header.

Alternatively, if the application does not require addressing each hardware processor individually, the application doesn't need to set this. Broadcasts from the processors use the BROADCAST preprocessor for class and device IDs.

```
const int C_BROADCAST_IDHI = -2;
const int C_BROADCAST_IDLO = -3;
```

5.1.3 Source Class

The source class should be set to 6 indicating a host.

```
const int C_HOST_CLASS_ID = 6;
```

5.1.4 Destination Class

If communicating with a specific PLM the destination class should be set to 5.

```
const int C_PLM_MODULE_CLASS_ID = 5;
```

Alternatively, if the application does not require addressing each hardware processor individually, the application can use the BROADCAST preprocessor macros for class and device IDs.

```
const int C_BROADCAST_CLASS_ID = 0;
```

5.1.5 Packet Length

The maximum length of a packet is 560 bytes, including the header and footer. The hardware will not accept larger packets. The sender must split data into smaller packets if the payload makes the packet larger than the maximum length.

5.1.6 Packet type

The packet type is a number which is unique for every type of packet that can be sent between devices. The two types available are "DLM message" for sending a message to the device and "Acknowledge" which indicates a reply to the previous DLM message (see 6).

```
const int C_ACK_MSG = 2;       // Ack message
const int C_DLM_MSG = 701;     // DLM message
```

5.1.7 Message ID

The message ID should be unique for every packet sent from a device that is expecting a response. When the receiving device sends a response back to the sender it puts the sender's Message ID into the header of the packet of its reply, this allows the sender to match the response with the request that it made.

If the sending device sets the Message ID to 0xFFFFFFFF (-1), the receiver will not send a response.

5.2 Packet Footer

The footer contains a reserved 4 byte (32 bit) value.

6 Messages

6.1 Msg_DLMMsg

Packet Name	Packet Type	Response	Comment
Msg_DLMMsg	701	Msg_DLMMsg or Msg_Ack	High Level command interface
Field	**Name**	**Size**	**Comment**
szMsg	Command text	Variable	Null terminated text string containing command

Msg_DLMMsg allows you to send high level commands as a text string. The command is parsed by the receiver. Data is either returned in an Msg_DLMMsg packet or an Msg_Ack that returns an error code or ACK_SUCCESS.

6.2 Msg_Ack

Packet Name	Packet Type	Response	Comment
Msg_Ack	2	N/A	Acknowledge sent from device in response to some packets
Field	**Name**	**Size**	**Comment**
lResult	Result	4 Bytes	A 32 bit return value (see table below)

The following table lists result codes for the Msg_Ack packet:

Ack Result	Value	Comment
ACK_SUCCESS	-2	Packet processed successfully
ACK_NOTMASTER	-3	Sender not master of the device
ACK_INVALID_PACKET	-4	Indicates a packet with bad checksum, or unsupported type
ACK_DSP_ERROR	-5	Communication with the DSP failed
ACK_BAD_PARAM	-6	Bad parameters in packet

To receive the responses to the Msg_DLMMsg packets, the listening application must receive packets on UDP port 6004.

6.3 Broadcast messages

PLM frames with firmware 2.58 are constantly broadcasting messages (e.g. packet type 4, 116, 165 etc) which is part of the proprietary control protocol and therefore not explained here. As of PLM firmware 2.74 and LM firmware 0.32 broadcast is no longer utilized and has been replaced by a unicast scheme.

7 Addressing a device

One can either address a specific device or all units at the same time.

7.1 Address a specific unit

There are two different ways of addressing a specific PLM unit.

- Send a unicast UDP with the specific device's hardware ID as Destination ID. This is the recommended addressing scheme.
  Please note that the default IP setting is Auto ZeroConf which means that a specific device might get different IP addresses every boot up.
- Send a broadcast UDP packet to the device with the specific device's hardware ID as Destination ID.

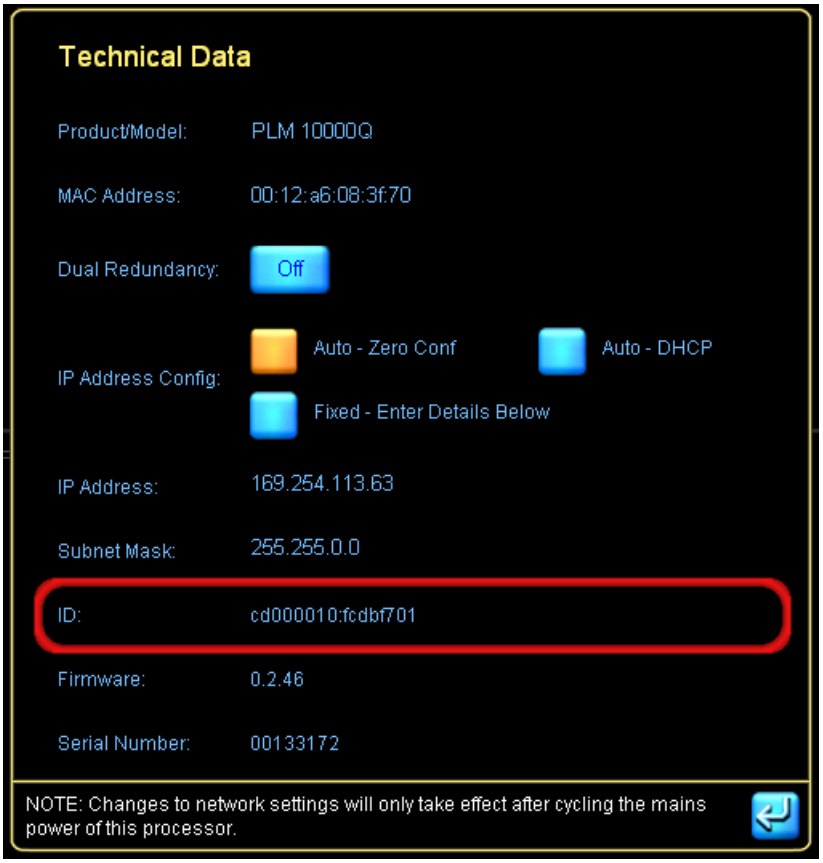
7.2 Address all units

If the application does not require addressing each hardware processor individually, the application can broadcast a UPD packet to all processors and set the Destination Class ID to broadcast.

7.3 Determining a unit's hardware ID

In order to send Ethernet packets to specific hardware processors on the network, each processor's unique hardware ID must be determined. These IDs are utilized as Destination ID within the packet header, as described in section 5.1.

The hardware ID is presented by the Lake Controller software within the I/O Config user interface display. From the Home page, navigate to Modules, select a module on the desired hardware processor and tap on the I/O Config button. Then click on the button entitled "Technical Data" to bring up the pop-up window as shown in the figure. The unique ID is highlighted below.



The hardware ID can also be found on the front panel by selecting MENU->Frame->Network.

8 Appendix A: Command reference

This section describes all commands in the 3rd party control protocol.

8.1 Nomenclature

These are the set of commands that will be supported to be sent as the szMsg in the Msg_DLMMsg. Each level in the command tree is separated with a '.' and all commands can be operated using one or many of the operation types:

- '!' is Do something
- '=' is Set/Store something
- '?' is Get something

Some commands have one or multiple parameters. In the commands below parameters are described within <> brackets. Each parameter is delimited with a single white space ' '. Formats and range of each parameter is described in the comments field.

8.2 Quick reference

Name	GET (?)	SET (=)	DO (!)	Supported by	Use / description
Device commands					
Dev.IsoFloat	X	X		PLM	Floating or grounded
Dev.IsoFloatInputs	X	X		LM	Inputs floating or grounded
Dev.IsoFloatOutputs	X	X		LM	Outputs floating or grounded
Dev.AesLoopTermination	X	X		All	Unterminated or terminated
Dev.Power	X	X		All	Standby or powered

Firmware					
Dev.BundleVer	X			All	Version of the bundle (.lkc file) loaded to host.
Preset					
Dev.Preset.Name	X			All	Names of stored frame presets
Dev.Preset.Recall			X	All	Recall a certain frame Preset
Dev.Preset.Store			X	All	Store a frame Preset
Network Settings					
Dev.Network.ID	X			All	Get Frames' HWID
Dev.Network.IPAddr	X			All	
Dev.Network.SubMask	X			All	
Dev.Network.MACAddr	X			All	
Reset					
Dev.Reset.Factory			X	All	Full factory reset
Dev.Reset Soft			X	All	Reset processing to default (Contour reset)
Breaker Emulation (Fuse)					
Dev.Fuse.NominalCurrent	X	X		PLM 20000Q	Breaker emulation current.
Dev.Fuse.Type	X	X		PLM 20000Q	Breaker emulation type.
Miscellaneous					
Dev.Route	X			PLM	Get how amp channels are routed from the Band DSP outputs
Dev.BridgeMode	X			PLM	Channel pair bridge mode.
Dev.Out.Route	X	X		LM	Get and set how channels are routed.
Dev.FrameLabel	X	X		All	
Dev.Latency	X			PLM	Amp latency + DLM latency
Dev.LatencyMatch	X	X		All	Match internal delay so that all platforms that are present in the system have the same inherent latency.
Dev.Speakers	X	X		PLM	Configure number of speakers for a physical channel
GPIO					
Dev.GPI.Config	X	X		LM	Get or set GPI configuration.
Dev.GPI.State	X			LM	Get current GPI state.
Dev.GPO.Config	X	X		LM	Get or set GPO configuration.
Dev.GPO.State	X			LM	Get current GPO state.
Dante					

Dev.Dante.BreakIn	X	X		PLM	Start transmitting/what is transmitted from the device out on the DANTE Network
Dev.Dante.Enabled	X			All	Is Dante Enabled
Dev.Dante.In.Label	X			LM	Dante input channels labels
Input type selection					
Dev.Router.InputTypSel	X	X		All	Configure inputs (prio, type, channel, sensitivity)
Dev.Router.InputAct	X			All	Is input active?
Dev.Router.ForceInputPriority	X	X		All	Select if auto selecting according to input priority settings should be used or if an input priority level should be forced
Dev.Router.InputSR	X			All	Input sample rate
Dev.Router.InputMute	X	X		All	Mute / unmute inputs
Meter Data					
Dev.MD.NoFaults	X			All	Checks if there is any faults present in the device
Dev.MD.FullBin	X			All	Retrieves a full meter data structure as binary data
Pilot tone generation					
Dev.PTG.Active	X	X		PLM	Activates / deactivates pilot tone generation for physical output channel
Dev.PTG.Impedance	X			PLM	Impedance measurement
Module commands					
Output Channel					
Mod.Out.Mute	X	X		All	Mute module output channel
Mod.Out.Gain	X	X		All	Control module output gain
Mod.Out.Delay	X	X		All	Control module output delay
Mod.Out.MaxRMSLvl	X	X		All	Max RMS Level
Mod.Out.MaxRMSCor	X	X		All	Max RMS Corner
Mod.Out.MaxRMSAtk	X	X		All	Max RMS Attack
Mod.Out.MaxRMSRel	X	X		All	Max RMS Release
Mod.Out.MaxPeakLvl	X	X		All	Max Peak level
Mod.Out.Phase	X	X		All	Output polarity
Mod.Out.Label	X	X		All	Module output label
Mod.Out.AmpGain	X	X		PLM	Module output amplifier gain
Mod.Out.AmpVPL	X	X		PLM	Voltage peak limit
Mod.Out.VPLProfile	X	X		PLM	Voltage peak limit profile
Mod.Out.Chans	X			All	Returns the number of output channels for a module
Mixer					

Mod.In.MixerGain	X	X		LM	Module input mixer gain
Input Channel					
Mod.In.Mute	X	X		All	Mute module input channel
Mod.In.Gain	X	X		All	Control module input gain
Mod.In.Delay	X	X		All	Control module input delay
Mod.In.Phase	X	X		All	Input polarity
Mod.In.Label	X	X		All	Module input label
Module					
Mod.Mod.Label	X			All	Module label
Mod.Mod.Selected	X	X		All	Use to indicate to PC and on front that module is selected.

8.3 Device Commands

ISO Float (Dev.IsoFloat)		**PLM 2.58**
Description		
Operation	**Parameters**	**Range / Resolution**
?		
Reply:	<Off/On>	0=Floating, 1=Grounded
Operation	**Parameters**	**Range / Resolution**
=	<Off/On>	0=Floating, 1=Grounded
Reply:	Ack	

ISO Float (Dev.IsoFloatInputs)		**LM 0.32**
Description		
Operation	**Parameters**	**Range / Resolution**
?		
Reply:	<Off/On>	0=Floating, 1=Grounded
Operation	**Parameters**	**Range / Resolution**
=	<Off/On>	0=Floating, 1=Grounded
Reply:	Ack	

ISO Float (Dev.IsoFloatOutputs)		**LM 0.32**
Description		
Operation	**Parameters**	**Range / Resolution**
?		
Reply:	<Off/On>	0=Floating, 1=Grounded
Operation	**Parameters**	**Range / Resolution**
=	<Off/On>	0=Floating, 1=Grounded
Reply:	Ack	

AES Loop termination (Dev.AesLoopTermination)		PLM 2.58, LM 0.32
Description		
Operation	**Parameters**	**Range / Resolution**
?		
Reply:	<Unconnected/Connected>	0=Unterminated, 1=Terminated
Operation	**Parameters**	**Range / Resolution**
=	<Unconnected/Connected>	0=Unterminated, 1=Terminated
Reply:	Ack	

Power (Dev.Power)		PLM 2.58, LM 0.32
Description		
Gets and sets desired power mode (on or standby)		
Operation	**Parameters**	**Range / Resolution**
?		
Reply:	<On/Standby>	1 = 0n, 0 = Standby
Operation	**Parameters**	**Range / Resolution**
=	<On/Standby>	1 = 0n, 0 = Standby
Reply:	Ack	
Example		
Additional information		
There are two additional acknowledge response codes that can occur with this command: #define ACK_SPI_PWRON -20L /* Command not sent because the frame is already on */ #define ACK_SPI_PWROFF -21L /* Command not sent because the frame is already off */		

8.3.1 Firmware

Bundle version (Dev.BundleVer)		PLM 2.58, LM 0.32
Description		
Version of the bundled (.lkc file) currently loaded.		
Operation	**Parameters**	**Range / Resolution**
?		
Reply:	<.lkc file Bundle Version>	szString

8.3.2 Preset

Preset Name (Dev.Preset.Name)		PLM 2.58, LM 0.32
Description		
Get the names of stored frame presets		
Operation	**Parameters**	**Range / Resolution**
?	<Preset #>	1-100
Reply:	<Preset Name>	szString[64]

Preset recall (Dev.Preset.Recall)		PLM 2.58, LM 0.32
Description		

Recall a certain frame Preset (delayed response)		
Operation	**Parameters**	**Range / Resolution**
!	<Preset #>	1-100
Reply:	Ack	
Example		
"Dev.Preset.Recall!1" - Recall preset 1.		

Preset Store (Dev.Preset.Store)		**PLM 2.74, LM 0.32**
Description		
Store a frame Preset		
Operation	**Parameters**	**Range / Resolution**
!	<Preset #>	1-100
	<Preset Name>	szString[64]
Reply:	Ack	
Example		
"Dev.Preset.Store!1 testpreset" - Stores a preset called testpreset on preset position 1		

8.3.3 Network settings

Frame ID (Dev.Network.ID)		**PLM 2.58, LM 0.32**
Description		
Get Frame Hardware ID		
Operation	**Parameters**	**Range / Resolution**
?		
Reply:	<DLM Frame ID>	szString xxxxxxxx:xxxxxxxx

IP Address (Dev.Network.IPAddr)		**PLM 2.58, LM 0.32**
Description		
Gets frame's IP address		
Operation	**Parameters**	**Range / Resolution**
?		
Reply:	<IP address>	szString xxx.xxx.xxx.xxx

Subnet mask (Dev.Network.SubMask)		**PLM 2.58, LM 0.32**
Description		
Get the frames subnet mask		
Operation	**Parameters**	**Range / Resolution**
?		
Reply:	<Subnet Mask>	szString xxx.xxx.xxx.xxx

MAC Address (Dev.Network.MACAddr)		**PLM 2.58, LM 0.32**
Description		
Get the frames MAC Address		
Operation	**Parameters**	**Range / Resolution**
?		

Reply:	<MAC Address>	szString xx:xx:xx:xx:xx:xx

8.3.4 Reset

Factory Reset (Dev.Reset.Factory)		**PLM 2.58, LM 0.32**
Description		
Do full factory reset		
Operation	**Parameters**	**Range / Resolution**
!		
Reply:	Ack	

Soft reset (Dev.Reset.Soft)		**PLM 2.58, LM 0.32**
Description		
Do soft reset		
Operation	**Parameters**	**Range / Resolution**
!		
Reply:	Ack	

8.3.5 Breaker Emulation (Fuse)

Nominal Current (Dev.Fuse.NominalCurrent)		**PLM 2.74**
Description		
Breaker emulation current. Only valid for PLM 20000Q.		
Type	**Parameters**	**Range/resolution**
?		
Reply:	<Nominal Current>	szString (x)x.x (range 0.1-50.0)
Type	**Parameters**	**Range/resolution**
=	<Nominal Current>	szString (x)x.x (range 0.1-50.0)
Reply:	Ack	

Type (Dev.Fuse.Type)		**PLM 2.74**
Description		
Breaker emulation type. Only valid for PLM 20000Q.		
Type	**Parameters**	**Range/resolution**
?		
Reply:	<Type>	0=Conservative, 1=Fast, 2=Universal
Type	**Parameters**	**Range/resolution**
=	<Type>	0=Conservative, 1=Fast, 2=Universal
Reply:	Ack	

8.3.6 Miscellaneous

Routing (Dev.Route)	**PLM 2.58**
Description	
Get how amp channels are routed from the Band DSP outputs	

Type	Parameters	Range/resolution
?	<Module> <Proc Channel>	A-B 1-6
Reply:	<Physical Outputs>	CsCsCsC (where C is X for position routed and – for not routed and s are space char)

Bridge Mode (Dev.BridgeMode)		**PLM 2.74**
Description		
Channel pair bridge mode.		
Type	**Parameters**	**Range/resolution**
?	<Channel Pair>	1-2
Reply:	<Bridge Mode>	0=Off, 1=On

Routing (Dev.Out.Route)		**LM 0.32**
Description		
Get and set how channels are routed.		
Type	**Parameters**	**Range/resolution**
?	<Source Type> <Channel Number> <Output Type>	PC, Analog, AES, Dante, Router (PC is Processing Channel) 1-x (x=6 for PC, 2 for Analog, 4 for AES and Dante, 6 for Router) Analog, AES, Dante
Reply:	<Routed Outputs>	CC..CC (where C is X for position routed and – for not routed) 6 C positions for Analog outputs and 8 C positions for AES and Dante outputs.
Type	**Parameters**	**Range/resolution**
=	<Source Type> <Channel Number> <Output Type> <Routed Outputs>	PC, Analog, AES, Dante, Router (PC is Processing Channel) 1-x (x=6 for PC, 2 for Analog, 4 for AES and Dante, 6 for Router) Analog, AES, Dante CC..CC (where C is X for position routed and – for not routed) 6 C positions for Analog outputs and 8 C positions for AES and Dante outputs.
Reply:	Ack	

Frame Label (Dev.FrameLabel)		**PLM 2.58, LM 0.32**
Description		
Type	**Parameters**	**Range/resolution**
?		
Reply:	<Frame Label>	szString[30]
Type	**Parameters**	**Range/resolution**
=	<Frame Label>	szString[30]
Reply:	Ack	

Inherrent latency (Dev.Latency)		**PLM 2.58**
Description		
Amp latency + DLM latency		
Type	**Parameters**	**Range/resolution**
?	<Physical Channel>	1-6
Reply:	<Inherent latency in ms>	szString (xxx)x.xxxx (range 0.000-1000.000)

Latency matching (Dev.LatencyMatch)		**PLM 2.58, LM 0.32**
Description		
Match internal delay so that all platforms that are present in the system have the same inherent latency.		
Type	**Parameters**	**Range/resolution**
?		
Reply:	<Match>	0=Off, 1=On
Type	**Parameters**	**Range/resolution**
=	<Match>	0=Off, 1=On
Reply:	Ack	

Number of Speakers (Dev.Speakers)		**PLM 2.58**
Description		
Type	**Parameters**	**Range/resolution**
?	< Physical Channel >	1-4
Reply:	<Number of Speakers>	
Type	**Parameters**	**Range/resolution**
=	<Physical Channel> <Number of Speakers>	1-4
Reply:	Ack	

8.3.7 GPIO

GPIO Input Configuration (Dev.GPI.Config)		**LM 0.32**
Description		
Type	**Parameters**	**Range/resolution**
?	<Input>	1-2
Reply:	<Action when Closed> <Action when Opened>	Actions for ProtectiveMuteState: ToggleMute, Mute, Unmute, NoAction Actions for StandbyState: ToggleStandby, Standby, On, NoAction Actions for PresetRecall: Recall99, Recall100, NoAction Action for NoAction: NoAction
Type	**Parameters**	**Range/resolution**
=	<Input> <Action when Closed> <Action when Opened>	1-2 Actions for ProtectiveMuteState: ToggleMute, Mute, Unmute, NoAction

		Actions for StandbyState: ToggleStandby, Standby, On, NoAction Actions for PresetRecall: Recall99, Recall100, NoAction Action for NoAction: NoAction Note that both closed and opened action must be of the same type, i.e. ProtectiveMuteState, StandbyState or PresetRecall.
Reply:	Ack	

GPIO Input State (Dev.GPI.State)		**LM 0.32**
Description		
Type	**Parameters**	**Range/resolution**
?	<Input>	1-2
Reply:	<Current State>	Open, Closed

GPIO Output Configuration (Dev.GPO.Config)		**LM 0.32**
Description		
Type	**Parameters**	**Range/resolution**
?	<Output>	1-2
Reply:	<Indication when Closed>	Indications for ProtectiveMuteState: Muted, Unmuted Indications for StandbyState: Standby, On Indications for Fault: NoFault, Fault Indications for Ready: NotReady, Ready Indications for NoIndication: NoInd
Type	**Parameters**	**Range/resolution**
=	<Output> <Indication when Closed>	1-2 Indications for ProtectiveMuteState: Muted, Unmuted Indications for StandbyState: Standby, On Indications for Fault: NoFault, Fault Indications for Ready: NotReady, Ready Indications for NoIndication: NoInd
Reply:	Ack	

GPIO Output State (Dev.GPO.State)		**LM 0.32**
Description		
Type	**Parameters**	**Range/resolution**
?	<Output>	1-2
Reply:	<Current State>	Open, Closed

8.3.8 Dante

Dante Break in Enable (Dev.Dante.BreakIn)		**PLM 2.79**
Description		
Start transmitting/what is transmitted from the device out on the DANTE Network		
Type	**Parameters**	**Range/resolution**
?	<Dante transmitter>	1-2
Reply:	<Type> <Ch> [Probe type] [Power output channel]	Analog, AES, Probe, Empty 1-2 U, I (Voltage or current probe, only returned when Type is Probe) 1-4 (only returned when Type is Probe)
Type	**Parameters**	**Range/resolution**
=	<Dante transmitter> <Type> <Ch> [Probe type] [Power output channel]	1-2 Analog, AES, Probe, Empty 1-2 U, I (Voltage or current probe, only valid when Type is Probe) 1-4 (only valid when Type is Probe)
Reply:	Ack	

ISSUE 1 - Input select not supported by Dante FW.

Dante Enabled (Dev.Dante.Enabled)		**PLM 2.58, LM 0.32**
Description		
Is Dante Enabled		
Type	**Parameters**	**Range/resolution**
?		
Reply:	<Enabled>	0=Off, 1=On

Dante Input Labels (Dev.Dante.In.Label)		**LM 0.32**
Description		
Type	**Parameters**	**Range/resolution**
?	<Dante input>	1-4
Reply:	<Label>	szString[32], Empty string if channel is unused

8.3.9 Input Type Selection

Input Type Settings (Dev.Router.InputTypSel)		**PLM 2.58 (PLM 2.80 for optional Sensitivity)**
Description		
Type	**Parameters**	**Range/resolution**
?	<Input> <Prio>	1-2 1-4
Reply:	<Type>	Analog, AES, Dante, Empty

	<Ch>	1-2
	<Sensitivity>	12.00 or 26.00 for Analog, -(xx)x.xx (range -100.00 - 15.00) for AES and Dante
	<Name>	Dante channel name (optional)
Type	**Parameters**	**Range/resolution**
=	<Input>	1-2
	<Prio>	1-4
	<Type>	Analog, AES, Dante, Empty
	<Ch>	1-2
	<Sensitivity>	12.00 or 26.00 for Analog, -(xx)x.xx (range -100.00 - 15.00) for AES and Dante ((Sensitivity is an optional parameter, when not submitted default values will be applied. 26 dBu for Analog and 0 dB for AES))
Reply:	Ack	

Note: The Sensitivity is in dBu for Analog and dB for AES.If an Analog input is selected at priority x then all priorities x+n for the same input must be Empty.

Input Type Settings (Dev.Router.InputTypSel)		**LM 0.32 (LM 0.35 for optional Sensitivity)**
Description		
Type	**Parameters**	**Range/resolution**
?	<Input>	1-6
	<Prio>	1-4
Reply:	<Type>	Analog, AES, Dante, Empty
	<Ch>	1-x (x=2 for analog, 4 for AES and Dante)
	<Sensitivity>	12.00 or 26.00 for Analog, -(xx)x.xx (range -100.00 - 15.00) for AES and Dante
Type	**Parameters**	**Range/resolution**
=	<Input>	1-6
	<Prio>	1-4
	<Type>	Analog, AES, Dante, Empty
	<Ch>	1-x (x=2 for analog, 4 for AES and Dante)
	<Sensitivity>	12.00 or 26.00 for Analog, -(xx)x.xx (range -100.00 - 15.00) for AES and Dante ((Sensitivity is an optional parameter, when not submitted default values will be applied. 26 dBu for Analog and 0 dB for AES))
Reply:	Ack	

Note: The Sensitivity is in dBu for Analog and dB for AES.If an Analog input is selected at priority x then all priorities x+n for the same input must be Empty.

Input active (Dev.Router.InputAct)		**PLM 2.58, LM 0.32**
Description		
Type	**Parameters**	**Range/resolution**
?	<Input>	1-x (2 for PLM, 6 for LM)
Reply:	<Priority>	1-4

Force Input Priority (Dev.Router.ForceInputPriority)		PLM 2.58, LM 0.32
Description		
Select if auto selecting according to input priority settings should be used or if an input priority level should be forced		
Type	**Parameters**	**Range/resolution**
?	<Input>	1-x (2 for PLM, 6 for LM)
Reply:	<Auto/Force>	0=Auto, 1-4=Force priority 1-4
Type	**Parameters**	**Range/resolution**
=	<Input> <Auto/Force>	1-x (2 for PLM, 6 for LM) 0=Auto, 1-4=Force priority 1-4
Reply:	Ack	

Input Sample Rate (Dev.Route.InputSR)		PLM 2.58
Description		
Type	**Parameters**	**Range/resolution**
?	<Type> <Ch>	AES 1-2
Reply:	<Sample rate in kHz>	00.0 for no input otherwise sample rate E.g. 44.1/48.0/88.2/96.0/176.4/192.0

Input Sample Rate (Dev.Route.InputSR)		LM 0.32
Description		
Type	**Parameters**	**Range/resolution**
?	<Type> <Ch>	AES, Dante 1-4
Reply:	<Sample rate in kHz>	00.0 for no input otherwise sample rate E.g. 44.1/48.0/88.2/96.0/176.4/192.0

Input Mute (Dev.Router.InputMute)		PLM 2.58, LM 0.32
Description		
Type	**Parameters**	**Range/resolution**
?	<Input>	1-x (2 for PLM, 6 for LM)
Reply:	<Mute>	1=Muted, 0=Unmuted
Type	**Parameters**	**Range/resolution**
=	<Input> <Mute>	1-2 1=Muted, 0=Unmute
Reply:	Msg_Ack	

8.3.10 Meter Data (Dev.MD)

Implementation note: Use "sAMPinfos" protection bit

No Faults (Dev.MD.NoFaults)		**PLM 2.58, LM 0.32**
Description		
Checks if there is any faults present in the device. Uses sAmpInfos protection bit and does not include bit 10 and 11 of Amp Status. *Note*: Frame must be **on** for this to be valid.		
Operation	**Parameters**	**Range / Resolution**
?		
Reply:	<Any faults present>	0 = Faults present, 1 = No faults present

Full Binary (Dev.MD.FullBin)		**PLM 2.58 & 2.74**
Description		
Retrieves a full meter data structure as binary data. *Note*: Frame must be **on** for this to be valid.		
Operation	**Parameters**	**Range / Resolution**
?	[DLM protocol Version]	1-2. Version 1 is assumed if the parameter is left out. Version parameter is only valid if FW is at least 2.74
Reply:	<Binary meter data structure>	See "Appendix B: Full meter data binary structure" for complete information.
Example		
"Dev.MD.Fullbin?" - retrieves a full version 1 meter data binary (works for PLM firmware 2.58 and above) "Dev.MD.Fullbin?2"- retrieves a full version 2 meter data binary (works for PLM firmware 2.74 and above)		

Full Binary (Dev.MD.FullBin)		**LM 0.32**
Description		
Retrieves a full meter data structure as binary data		
Operation	**Parameters**	**Range / Resolution**
?		
Reply:	<Binary meter data structure>	See "Appendix B: Full meter data binary structure" for complete information.

8.3.11 Pilot Tone Generation (Dev.PTG)

Active (Dev.PTG.Active)		**PLM 2.58**
Description		
Activates or deactivates pilot tone for a given physical output channel. Make sure that pilot tone is correctly configured before activation!		
Operation	**Parameters**	**Range / Resolution**
?	<Physical output channel>	1/2/3/4
Reply:	<PTG State>	1 = Active, 0 = Not active
Operation	**Parameters**	**Range / Resolution**
=	<Physical output channel> <PTG State >	1/2/3/4 1 = Activate, 0 = Deactivate
Reply:	Ack	
Example		
"Dev.PTG.Active = 2 1" - activates pilot tone for physical output 2		

Impedance (Dev.PTG.Impedance)		**PLM 2.58**
Description		
Gets the current impedance measurement for a physical output channel. The impedance is a ~1.5 seconds average absolute value of the complex impedance.		
Operation	**Parameters**	**Range / Resolution**
?	<Physical output channel>	1/2/3/4 This parameter is optional. If omitted, impedance is returned for all physical output channels.
Reply:	<Imp.> <*Imp.*> <*Imp.*> <*Imp.*>	In ohms with one decimal, formatted X.X ranging from 0.0 ohms to 6000.0 ohms. A '?' indicates that there are no value measured for a certain channel (PTG inactive).
Example		
"Dev.PTG.Impedance? 1" - Get impedance for physical output channel 1 "7.3" - Reply shows measured impedance to be 7.3 ohms		

8.4 Module commands

8.4.1 Output channel

Mute (Mod.Out.Mute)		**PLM 2.58, LM 0.32**
Description		
Type	**Parameters**	**Range/resolution**
?	<Module> <Ch>	A-B 1-6
Reply:	<Mute>	1=Muted, 0=Unmuted
Type	**Parameters**	**Range/resolution**
=	<Module> <Ch> <Mute>	A-B 1-6 1=Muted, 0=Unmute
Reply:	Msg_Ack	

Gain (Mod.Out.Gain)		**PLM 2.58, LM 0.32**
Description		
Type	**Parameters**	**Range/resolution**
?	<Module> <Ch>	A,B 1-6
Reply:	<Gain in dB> <GroupSum in dB> <SumMin in dB> <SumMax in dB>	szString (-xx)x.xx (range -100.00 - 20.00) szString (-xx)x.xx (range -100.00 - 20.00) szString (-xx)x.xx (range -100.00 - 20.00) szString (-xx)x.xx (range -100.00 - 20.00)
Type	**Parameters**	**Range/resolution**
=	<Module> <Ch>	A-B 1-6

	<Gain in dB>	szString (-xx)x.xx (range -100.00 - 20.00)
Reply:	Ack	

Delay (Mod.Out.Delay)		**PLM 2.58, LM 0.32**
Description		
Type	**Parameters**	**Range/resolution**
?	<Module>	A-B
	<Ch>	1-6
Reply:	<Delay in ms>	szString (xxx)x.xx (range 0.00-100.00)
	<GroupSum in ms>	szString (xxx)x.xx (range 0.00-100.00)
	<SumMin in ms>	szString (xxx)x.xx (range 0.00-100.00)
	<SumMax in ms>	szString (xxx)x.xx (range 0.00-100.00)
Type	**Parameters**	**Range/resolution**
=	<Module>	A-B
	<Ch>	1-6
	<Delay in ms>	szString (xxx)x.xx (range 0.00-100.00)
Reply:	Ack	

MaxRMS Level (Mod.Out.MaxRMSLvl)		**PLM 2.58, LM 0.32**
Description		
Type	**Parameters**	**Range/resolution**
?	<Module>	A-B
	<Ch>	1-6
Reply:	<MaxRMS Level in dB>	szString (-x)x.xx (range -30.00 - 30.00)
	<GroupSum in dB>	szString (-x)x.xx (range -30.00 - 30.00)
	<SumMin in dB>	szString (-x)x.xx (range -30.00 - 30.00)
	<SumMax in dB>	szString (-x)x.xx (range -30.00 - 30.00)
Type	**Parameters**	**Range/resolution**
=	<Module>	A-B
	<Ch>	1-6
	<MaxRMS Level in dB>	szString (-x)x.xx (range -30.00 - 30.00)
Reply:	Ack	

MaxRMS Corner (Mod.Out.MaxRMSCor)		**PLM 2.58, LM 0.32**
Description		
Type	**Parameters**	**Range/resolution**
?	<Module>	A-B
	<Ch>	1-6
Reply:	<MaxRMS Corner in dB>	szString (-xx)x.xx (range -100.00 – 0.00)
	<MaxRMS Corner Min in dB>	szString (-xx)x.xx (range -100.00 – 0.00)
	<MaxRMS Corner Max in dB>	szString (-xx)x.xx (range -100.00 – 0.00)
Type	**Parameters**	**Range/resolution**

=	<Module> <Ch> <MaxRMS Corner in dB>	A-B 1-6 szString (-xx)x.xx (range -100.00 – 0.00)
Reply:	Ack	

MaxRMS Attack (Mod.Out.MaxRMSAtk)		**PLM 2.58, LM 0.32**
Description		
Type	**Parameters**	**Range/resolution**
?	<Module> <Ch>	A-B 1-6
Reply:	<MaxRMS Attack in ms> <MaxRMS Attack Min in ms> <MaxRMS Attack Max in ms>	szString (xx)x.xx (range 1.00-500.00) szString (xx)x.xx (range 1.00-500.00) szString (xx)x.xx (range 1.00-500.00)
Type	**Parameters**	**Range/resolution**
=	<Module> <Ch> <MaxRMS Attack in ms>	A-B 1-6 szString (xx)x.xx (range 1.00-500.00)
Reply:	Ack	

MaxRMS Release (Mod.Out.MaxRMSRel)		**PLM 2.58, LM 0.32**
Description		
Type	**Parameters**	**Range/resolution**
?	<Module> <Ch>	A-B 1-6
Reply:	<MaxRMS Release in ms> <MaxRMS Release Min in ms> <MaxRMS Release Max in ms>	szString (xx)x.xx (range 1.00-500.00) szString (xx)x.xx (range 1.00-500.00) szString (xx)x.xx (range 1.00-500.00)
Type	**Parameters**	**Range/resolution**
=	<Module> <Ch> <MaxRMS Release in ms>	A-B 1-6 szString (xx)x.xx (range 1.00-500.00)
Reply:	Ack	

MaxPeak Level (Mod.Out.MaxPeakLvl)		**PLM 2.58, LM 0.32**
Description		
Type	**Parameters**	**Range/resolution**
?	<Module> <Ch>	A-B 1-6
Reply:	<MaxPeak Level in dB> <GroupSum in dB> <SumMin in dB> <SumMax in dB>	szString (-x)x.xx (range -30.00 - 30.00) szString (-x)x.xx (range -30.00 - 30.00) szString (-x)x.xx (range -30.00 - 30.00) szString (-x)x.xx (range -30.00 - 30.00)

Type	Parameters	Range/resolution
=	<Module>	A-B
	<Ch>	1-6
	<MaxPeak Level in dB>	szString (-x)x.xx (range -30.00 - 30.00)
Reply:	Ack	

Phase (Mod.Out.Phase)		**PLM 2.58, LM 0.32**
Description		
Type	**Parameters**	**Range/resolution**
?	<Module>	A-B
	<Ch>	1-6
Reply:	<Phase>	1=Positive,0=Negative
	<Lock>	1=Locked, 0=Not Locked
Type	**Parameters**	**Range/resolution**
=	<Module>	A-B
	<Ch>	1-6
	<Phase>	1=Positive,0=Negative
Reply:	Ack	

Output Label (Mod.Out.Label)		**PLM 2.58, LM 0.32**
Description		
Type	**Parameters**	**Range/resolution**
?	<Module>	A-B
	<Ch>	1-6
Reply:	<Label>	szString[32]
Type	**Parameters**	**Range/resolution**
=	<Module>	A-B
	<Ch>	1-6
	<Label>	szString[32]
Reply:	Ack	

Amp Gain (Mod.Out.AmpGain)		**PLM 2.58**
Description		
Type	**Parameters**	**Range/resolution**
?	<Module>	A-B
	<Ch>	1-6
Reply:	<AmpGain in dB>	szString xx (range 22-44)
	<AmpGain Min in dB>	szString xx (range 22-44)
	<AmpGain Max in dB>	szString xx (range 22-44)
Type	**Parameters**	**Range/resolution**
=	<Module>	A-B
	<Ch>	1-6

	<AmpGain in dB>	szString xx (range 22-44)
Reply:	Ack	

Amp VoltagePeakLimiter (Mod.Out.AmpVPL)		**PLM 2.58**
Description		
Type	**Parameters**	**Range/resolution**
?	<Module> <Ch>	A-B 1-6
Reply:	<Desired VPL in V> <VPL Min in V> <VPL Max in V> <Actual VPL in V>	szString (x)xx.x (range 17.8-193.0) szString (x)xx.x (range 17.8-193.0) szString (x)xx.x (range 17.8-193.0) szString (x)xx.x (range 17.8-193.0)
Type	**Parameters**	**Range/resolution**
=	<Module> <Ch> <Desired VPL in V>	A-B 1-6 szString (x)xx.x (range 17.8-193.0)
Reply:	Ack	

Amp VPL Profile (Mod.Out.VPLProfile)		**PLM 2.58**
Description		
Type	**Parameters**	**Range/resolution**
?	<Module> <Ch>	A-B 1-6
Reply:	<VPL Profile>	0=Universal, 1=Sub/LF, 2=Sub, 3=LF, 4=MF, 5=HF
Type	**Parameters**	**Range/resolution**
=	<Module> <Ch> <VPL Profile>	A-B 1-6 0=Universal, 1=Sub/LF, 2=Sub, 3=LF, 4=MF, 5=HF
Reply:	Ack	

Output Channels (Mod.Out.Chans)		**PLM 2.58, LM 0.32**
Description		
Returns the number of output channels for a module		
Type	**Parameters**	**Range/resolution**
?	<Module>	A-B
Reply:	<Channels>	1-6

8.4.2 Mixer

Mixer Gain (Mod.In.MixerGain)		**LM 0.32**
Description		
Type	**Parameters**	**Range/resolution**

?	<Module> <Channel>	A-B 1-4 (for input router 1-4 in mixer for module A or B)
Reply:	<Gain in dB>	szString (-xx)x.xx (range -100.00 - 15.00)
Type	**Parameters**	**Range/resolution**
=	<Module> <Channel> <Gain in dB>	A-B 1-4 szString (-xx)x.xx (range -100.00 - 15.00)
Reply:	Ack	

8.4.3 Input channel

Mute (Mod.In.Mute)		**PLM 2.58, LM 0.32**
Description		
Type	**Parameters**	**Range/resolution**
?	<Module>	A-B
Reply:	<Mute>	1=Muted, 0=Unmuted
Type	**Parameters**	**Range/resolution**
=	<Module> <Mute>	A-B 1=Muted, 0=Unmuted
Reply:	Ack	

Gain (Mod.In.Gain)		**PLM 2.58, LM 0.32**
Description		
Type	**Parameters**	**Range/resolution**
?	<Module	A-B
Reply:	<Gain in dB> <GroupSum in dB> <SumMin in dB> <SumMax in dB>	szString (-xx)x.xx (range -100.00 - 15.00) szString (-xx)x.xx (range -100.00 - 15.00) szString (-xx)x.xx (range -100.00 - 15.00) szString (-xx)x.xx (range -100.00 - 15.00)
Type	**Parameters**	**Range/resolution**
=	<Module> <Gain in dB>	A-B szString (-xx)x.xx (range -100.00 - 15.00)
Reply:	Ack	

Delay (Mod.In.Delay)		**PLM 2.58, LM 0.32**
Description		
Type	**Parameters**	**Range/resolution**
?	<Module>	A-B
Reply:	<Delay in ms> <GroupSum in ms> <SumMin in ms> <SumMax in ms>	szString (xxx)x.xx (range 0.00-1800.00) szString (xxx)x.xx (range 0.00-1800.00) szString (xxx)x.xx (range 0.00-1800.00) szString (xxx)x.xx (range 0.00-1800.00)

Type	Parameters	Range/resolution
=	<Module> <Delay in ms>	A-B szString (xxx)x.xx (range 0.00-1800.00)
Reply:	Ack	

Phase (Mod.In.Phase)		PLM 2.58, LM 0.32
Description		
Type	**Parameters**	**Range/resolution**
?	<Module>	A-B
Reply:	<Phase>	0=Negative, 1=Positive
Type	**Parameters**	**Range/resolution**
=	<Module> <Phase>	A-B 0=Negative, 1=Positive
Reply:	Ack	

Input Label (Mod.In.Label)		PLM 2.58, LM 0.32
Description		
Type	**Parameters**	**Range/resolution**
?	<Module>	A-B
Reply:	<Label>	szString[32]
Type	**Parameters**	**Range/resolution**
=	<Module> <Label>	A-B szString[32]
Reply:	Ack	

8.4.4 Module

Module Label (Mod.Mod.Label)		PLM 2.58, LM 0.32
Description		
Type	**Parameters**	**Range/resolution**
?	<Module>	A-B
Reply:	<Label>	szString[32]
Type	**Parameters**	**Range/resolution**
=	<Module> <Label>	A-B szString[32]
Reply:	Ack	

Module selected (Mod.Mod.Selected)		PLM 2.58, LM 0.32
Description		
Used to indicate to PC and on front that module is selected.		
Type	**Parameters**	**Range/resolution**
?	<Module>	A-B

Reply:	<Selected>	0=Not selected, 1=Selected
Type	**Parameters**	**Range/resolution**
=	<Module> <Selected>	A-B 0=Not selected, 1=Selected
Reply:	Ack	

9 Appendix B: Full meter data binary structure

9.1 PLM protocol Version 1 (FW Version 2.58 and above)

Field	Name	Size	Comment
sAMPInfo	**AMP Information**	**36 Bytes**	**See 9.1.2**
uPeakA	**Peak value module A**	**1 Bytes**	**Peak and RMS values for module inputs. 0xFF=ADO/AES fullscale (overshoot), 0xFE=-0.5dBFS, 0x01=-127.0dBFS, 0x00=<-127dBFS**
uRMSA	**RMS value module A**	**1 Bytes**	
uPeakB	**Peak value module B**	**1 Bytes**	
uRMSB	**RMS value module B**	**1 Bytes**	
uStatus	**Status information**	**4 Bytes**	**See 9.1.1**
uPeak1	**Peak value physical In 1**	**1 Bytes**	
uRMS1	**RMS value physical In 1**	**1 Bytes**	
uPeak2	**Peak value physical In 2**	**1 Bytes**	
uRMS2	**RMS value physical In 2**	**1 Bytes**	

9.1.1 uStatus field

uStatus	Bit	Comment
LINK_1	0	Ethernet 1 link status, 1=Network connected (Link), 0=No connection to network
LINK_2	1	Ethernet 2 link status, 1=Network connected (Link), 0=No connection to network
RESERVED	2-3	Unused bits should be set to 0
CTRL_PRESENT	4	1=Controller present on network, 0=No controller present on network
AD_DATA_FAIL	5	1=AD data encoding violation, 0=No violation.
RESERVED	6-7	Unused bits should be set to 0
ModA_1	8	Module A input 1, 1=Connected, 0=Not Connected
ModA_2	9	Module A input 2, 1=Connected, 0=Not Connected
ModB_1	10	Module B input 1, 1=Connected, 0=Not Connected
ModB_2	11	Module B input 2, 1=Connected, 0=Not Connected
RESERVED	12-15	Unused bits should be set to 0
OUTPUT_CLIP_A	16-21	6 bits, each bit indicates the clip state of Module A output channels

OUTPUT_CLIP_B	22-27	6 bits, each bit indicates the clip state of Module B output channels
RESERVED	28-31	Unused bits should be set to 0

9.1.2 Amp Info

All Amp data is peak and hold (worst value kept until read).

Amp Meter data (sAmpInfo)			
Bytes Bit 31-24	**Bit 23-16**	**Bit 15-8**	**Bit 7-0**
Reserved		Amp Status	
ChA Status			
ChA Power	ChA Voltage	ChA Current	ChA Gain Reduction
ChB Status			
ChB Power	ChB Voltage	ChB Current	ChB Gain Reduction
ChC Status			
ChC Power	ChC Voltage	ChC Current	ChC Gain Reduction
ChD Status			
ChD Power	ChD Voltage	ChD Current	ChD Gain Reduction

9.1.2.1 *Amp Status*

Amp Status			
Bit number	**Parameter**	**Severity**	**Description**
15	Reserved	-	-
14	DLM selected	Status	Indicates that the DLM is selected. (Selection is done by pressing the appropriate button on the front panel).
13	DLM dirty	Status	Indicates that data in the DLM has been changed and should be re-read by the external host.
12	Sense warning	Warning	There is no activity in voltage and current sense on any channel when power is ON.
11	Board Data Fault	Fault	Flash checksum error in host.
10	AD Data Fault	Fault	AD encoding error in host.
8-9	Slot temperature	Fault	Temp warning indicator for DSP area **0x00** No Warning **0x01** Not used **0x02** Fault
7	Protect	Fault	Active if any channel in protective mode.
6	Load monitor	Status	There is an ongoing load monitor measurement.
5-4	Power Supply Temperature	Fault	Temp warning indicator. **0x00** No Warning **0x01** Not used **0x02** Fault
3	Audio In Fault	Fault	Deactivated Audio OK/ or missing audio clocks.

2	PAL	Fault	Active limitation of mains input due to risk of braking mains fuse.
1	Power Supply Failure	Fault	Main power supply failing.
0	Power Status	Status	Actual power state ON/OFF (Delayed).

9.1.2.2 *Channel Status*

Channel Status			
Bit number	**Parameter**	**Severity**	**Description**
31	Service	Fault	Amp channel needs service fault
30	VHF	Fault	Very High Frequency Fault
29	Short Circuit	Fault	Short Circuit Fault.
28-27	Temp	Warning/Fault	Temp warning indicator **0x00** No Warning **0x01** Warning **0x02** Fault
26	Open Load	Warning	Load monitor detected no load.
25	U_{clip}	Status	Voltage Clip
24	I_{clip}	Status	Current Clip
23	Correct	Status	Load monitor correct speaker is connected.
22	Wrong	Fault	Load monitor wrong speaker is connected.
21	Not Verified	Warning	Load monitor has not verified the speakers.
20	No Ident Model	Warning	Load monitor has no identification model.
19	No Temp Model	Warning	Load monitor has no temperature model.
18	Voice Coil Temp Fault	Fault	Load monitor voice coil temperature fault.
17	Voice Coil Temp Warning	Warning	Load monitor voice coil temperature warning.
16	Mag Temp Fault	Fault	Load monitor speaker magnet temperature fault.
15	Mag Temp Warning	Warning	Load monitor speaker magnet temperature warning.
14	Less Speakers	Warning	Load monitor detected less speakers than expected.
13	More Speakers	Warning	Load monitor detected more speakers than expected.
12	Model Prec Low	Warning	Load monitor model precision is low.
11	Uncertain	Warning	Load monitor is uncertain if correct load is connected or not.
10	CAL	Warning	Current avarage limiter is active
9	Live not started	Warning	Live part of load monitor has not been started
8	Reserved	N/A	Reserved for future use.
7-0	VoltageRMS	Value	Voltage RMS **Value** / **Description** 0xFF: RMS of Max Out Voltage 0xFE: 0.5dB from RMS of Max Out Voltage ….: ….

			0x00	-Infinity from RMS of Max Out Voltage or below noise floor

9.1.2.3 ***Channel Power***

Channel Power	
Value	**Description**
0xFF	Max power into the attached nominal load
0xFE	-0,5 dB relative to Max power
….	….
0x01	-127 dB relative to Max Power
0x00	Below noise floor

9.1.2.4 ***Channel Voltage***

Channel Voltage	
Value	**Description**
0xFF	Max Out Voltage
0xFE	0.5dB from Max Out Voltage
….	….
0x00	-Infinity from Max Out Voltage or below noise floor

9.1.2.5 ***Channel Current***

Channel Current	
Value	**Description**
0xFF	Max Out Current
0xFE	0.5dB from Max Out Current
….	….
0x00	-Infinity from Max Out Current or below noise floor

9.1.2.6 ***Channel Gain Reduction***

Channel Gain Reduction	
Value	**Description**
0xFF	Reserved
0xFE	25.4 dB Gain reduction
….	….
0x01	0.1 dB Gain reduction
0x00	No gain reduction

9.2 PLM protocol Version 2 (FW Version 2.74 and above)

Field	Name	Size	Comment
sAMPInfo	**AMP Information**	**36 Bytes**	**See 9.2.2**
uPeakA	**Peak value module A**	**1 Bytes**	**Peak and RMS values for module inputs. 0xFF=ADO/AES fullscale (overshoot), 0xFE=-0.5dBFS, 0x01=-127.0dBFS, 0x00=<-127dBFS**
uRMSA	**RMS value module A**	**1 Bytes**	
uPeakB	**Peak value module B**	**1 Bytes**	
uRMSB	**RMS value module B**	**1 Bytes**	
uStatus	**Status information**	**4 Bytes**	**See 9.2.1**
uPeak1	**Peak value physical In 1**	**1 Bytes**	
uRMS1	**RMS value physical In 1**	**1 Bytes**	
uPeak2	**Peak value physical In 2**	**1 Bytes**	
uRMS2	**RMS value physical In 2**	**1 Bytes**	
uPeakGrModOut1	**Peak g.r.v. out ch. 1**	**1 Bytes**	**Peak gain reduction values for output channels. 0xFF = 25.5 dB gain reduction 0xFE = 25.0 dB, 0x01 = 0.1 dB 0x00 = 0 dB gain reduction**
uPeakGrModOut2	**Peak g.r.v. out ch. 2**	**1 Bytes**	
uPeakGrModOut3	**Peak g.r.v. out ch. 3**	**1 Bytes**	
uPeakGrModOut4	**Peak g.r.v. out ch. 4**	**1 Bytes**	
uRMSGrModOut1	**RMS g.r.v. out ch. 1**	**1 Bytes**	**RMS gain reduction values for output channels. 0xFF = 25.5 dB gain reduction 0xFE = 25.0 dB, 0x01 = 0.5 dB 0x00 = 0 dB gain reduction**
uRMSGrModOut2	**RMS g.r.v. out ch. 2**	**1 Bytes**	
uRMSGrModOut3	**RMS g.r.v. out ch. 3**	**1 Bytes**	
uRMSGrModOut4	**RMS g.r.v. out ch. 4**	**1 Bytes**	
uStatus2	**Status information 2**	**4 Bytes**	**Reserved**

9.2.1 uStatus field

uStatus	Bit	Comment
LINK_1	0	Ethernet 1 link status, 1=Network connected (Link), 0=No connection to network
LINK_2	1	Ethernet 2 link status, 1=Network connected (Link), 0=No connection to network
RESERVED	2-3	Unused bits should be set to 0
CTRL_PRESENT	4	1=Controller present on network, 0=No controller present on network
AD_DATA_FAIL	5	1=AD data encoding violation, 0=No violation.
CLOCK_SLIP_AES	6	1=AES clock slipping, 0=Clock ok
CLOCK_SLIP_DANTE	7	1=DANTE clock slipping, 0=Clock ok

ModA_1	8	Module A input 1, 1=Connected, 0=Not Connected
ModA_2	9	Module A input 2, 1=Connected, 0=Not Connected
ModB_1	10	Module B input 1, 1=Connected, 0=Not Connected
ModB_2	11	Module B input 2, 1=Connected, 0=Not Connected
DANTE_MASTER	12	1=Unit is DANTE clock master, 0=Slave
RESERVED	13-15	Unused bits should be set to 0
OUTPUT_CLIP_A	16-21	6 bits, each bit indicates the clip state of Module A output channels
OUTPUT_CLIP_B	22-27	6 bits, each bit indicates the clip state of Module B output channels
RESERVED	28-31	Unused bits should be set to 0

9.2.2 Amp Info

All Amp data is peak and hold (worst value kept until read).

Amp Meter data (sAmpInfo)			
Bytes Bit 31-24	**Bit 23-16**	**Bit 15-8**	**Bit 7-0**
PSU Mains Input		Amp Status	
ChA Status			
ChA Power	ChA Voltage	ChA Current	ChA Gain Reduction
ChB Status			
ChB Power	ChB Voltage	ChB Current	ChB Gain Reduction
ChC Status			
ChC Power	ChC Voltage	ChC Current	ChC Gain Reduction
ChD Status			
ChD Power	ChD Voltage	ChD Current	ChD Gain Reduction
PSU Model Limit	PSU BEL Limit	PSU UVL Limit	PSU Current Limit Activity
PSU Peak Current	PSU Average Current	PSU Peak Power	PSU Average Power
ChA Temp Limit	ChB Temp Limit	ChC Temp Limit	ChD Temp Limit
PSU Temp Limit	PSU Vcap Limit	PSU Status	
ChA Ext Status	ChB Ext Status	ChC Ext Status	ChD Ext Status
Amp Ext Status			

9.2.2.1 *PSU Mains Input*

This block is only defined for PLM20000Q and is otherwise not valid.

Bit number	**Parameter**	**Severity**	**Description**
15-8	PSU Mains current		Mains current RMS **Value** / **Description** 0x00 / 0 % of Max input current 0x01 / 0.5 % of Max input current 0x02 / 1.0% of Max input current …. / ….

<table>
<tr><td></td><td></td><td></td><td><table><tr><td>0xFF</td><td>127.5 % of Max input current</td></tr></table></td></tr>
<tr><td>7-0</td><td>PSU Mains Voltage</td><td></td><td>Mains voltage RMS (43-413V) according to the following formula:
$y = \max(0, \min(x, 275) + \max(0, \frac{x-275}{6}) - 43)$
<table><tr><th>Value (y)</th><th>Voltage (x)</th></tr><tr><td>0x00</td><td>≤ 43 VRMS</td></tr><tr><td>0x01</td><td>44 VRMS</td></tr><tr><td>.</td><td>….</td></tr><tr><td>0xE7</td><td>274 VRMS</td></tr><tr><td>0xE8</td><td>275 VRMS</td></tr><tr><td>0xEE</td><td>281 VRMS</td></tr><tr><td>…</td><td>…</td></tr><tr><td>0xFE</td><td>407 VRMS</td></tr><tr><td>0xFF</td><td>≥ 413 VRMS</td></tr></table></td></tr>
</table>

9.2.2.2 ***Amp Status***

Amp Status			
Bit number	**Parameter**	**Severity**	**Description**
15	Reserved		Value is zero
14	Reserved for LP2D as DLM selected	Status	Indicates that the DLM is selected. (Selection is done by pressing the appropriate button on the front panel)
13	Reserved for LP2D as DLM dirty	Status	Indicates that data in the DLM has been changed and should be reread by the external host.
12	Sense warning	Warning	There is no activity in voltage and current sense on any channel when power is ON
11	Reserved for LP2D as Board Data Fault	Fault	Flash checksum error in host.
10	Reserved for LP2D as AD Data Fault	Fault	AD encoding error in host.
8-9	DSP area temperature	Fault	Temp warning indicator for DSP area **0x00** **No Warning** **0x01** Warning **0x02** Not used **0x03** Fault
7	Protect	Fault	Active if any channel in protective mode.
6	Load monitor	Status	There is an ongoing load monitor measurement.

5-4	Power Supply Temperature	Fault	Temp warning indicator. **0x00** **No Warning** **0x01** Warning ISVPL limit threshold reduced **0x02** Not used **0x03** Fault
3	Audio In Fault	Fault	Deactivated Audio OK/ or missing audio clocks.
2	PAL	PAL	Active limitation of mains input due to risk of braking mains fuse. This flag is valid only for PLM1000Q and PLM14000. For PLM20000Q see PSU status block below.
1	Power Supply Failure	Fault	Mains power supply failing.
0	Power Status	Status	Actual power state ON/OFF (Delayed).

9.2.2.3 ***Channel Status***

Channel Status			
Bit number	**Parameter**	**Severity**	**Description**
31	Service	Fault	Amp channel needs service fault
30	VHF	Fault	Very High Frequency Fault
29	Short Circuit	Fault	Short Circuit Fault.
28-27	Temp	Warning/Fault	Temp warning indicator **0x00** No Warning **0x01** Warning **0x02** Fault
26	Open Load	Warning	Load monitor detected no load.
25	U_{clip}	Status	Voltage Clip
24	I_{clip}	Status	Current Clip
23	Correct	Status	Load monitor correct speaker is connected.
22	Wrong	Fault	Load monitor wrong speaker is connected.
21	Not Verified	Warning	Load monitor has not verified the speakers.
20	No Ident Model	Warning	Load monitor has no identification model.
19	No Temp Model	Warning	Load monitor has no temperature model.
18	Voice Coil Temp Fault	Fault	Load monitor voice coil temperature fault.
17	Voice Coil Temp Warning	Warning	Load monitor voice coil temperature warning.
16	Mag Temp Fault	Fault	Load monitor speaker magnet temperature fault.
15	Mag Temp Warning	Warning	Load monitor speaker magnet temperature warning.
14	Less Speakers	Warning	Load monitor detected less speakers than expected.
13	More Speakers	Warning	Load monitor detected more speakers than expected.
12	Model Prec Low	Warning	Load monitor model precision is low.
11	Uncertain	Warning	Load monitor is uncertain if correct load is connected or not.
10	CAL	Warning	Current avarage limiter is active

<table>
<tr><td>9</td><td>Live not started</td><td>Warning</td><td>Live part of load monitor has not been started</td></tr>
<tr><td>8</td><td>Reserved</td><td>N/A</td><td>Reserved for future use.</td></tr>
<tr><td>7-0</td><td>VoltageRMS</td><td>Value</td><td>Voltage RMS
<table>
<tr><th>Value</th><th>Description</th></tr>
<tr><td>0xFF</td><td>RMS of Max Out Voltage</td></tr>
<tr><td>0xFE</td><td>0.5dB from RMS of Max Out Voltage</td></tr>
<tr><td>….</td><td>….</td></tr>
<tr><td>0x00</td><td>-Infinity from RMS of Max Out Voltage or below noise floor</td></tr>
</table></td></tr>
</table>

9.2.2.4 *Channel Power*

Channel Power	
Value	**Description**
0xFF	Max power into the attached nominal load
0xFE	-0,5 dB relative to Max power
….	….
0x01	-127 dB relative to Max Power
0x00	Below noise floor

9.2.2.5 *Channel Voltage*

Channel Voltage	
Value	**Description**
0xFF	Max Out Voltage
0xFE	0.5dB from Max Out Voltage
….	….
0x00	-Infinity from Max Out Voltage or below noise floor

9.2.2.6 *Channel Current*

Channel Current	
Value	**Description**
0xFF	Max Out Current
0xFE	0.5dB from Max Out Current
….	….
0x00	-Infinity from Max Out Current or below noise floor

9.2.2.7 *Channel Gain Reduction*

Channel Gain Reduction	
Value	**Description**
0xFF	Reserved

0xFE	25.4 dB Gain reduction
….	….
0x01	0.1 dB Gain reduction
0x00	No gain reduction

9.2.2.8 *PSU Model Limit*

A one byte value indicating the minimum value of the current limit enforced from the model power or current limit. This block is only defined for PLM20000Q and is filled with zeros otherwise. The value is intended as a roof in PSU power meters.

Value	Description
0xFF	63.75 A
0xFE	63.50 A
….	….
0x01	0.25 A
0x00	0 A

9.2.2.9 *PSU BEL Limit*

A one byte value indicating the minimum value of the current limit enforced from the external fuse protection. This block is only defined for PLM20000Q and is filled with zeros otherwise. The value is intended as a roof in PSU power meters

Value	Description
0xFF	63.75 A
0xFE	63.50 A
….	….
0x01	0.25 A
0x00	0 A

9.2.2.10 *PSU UVL Limit*

A one byte value indicating the minimum value of the current limit enforced during a mains under voltage condition. This block is only defined for PLM20000Q and is filled with zeros otherwise. The value is intended as a roof in PSU power meters

Value	Description
0xFF	63.75 A
0xFE	63.50 A
….	….
0x01	0.25 A
0x00	0 A

9.2.2.11 *PSU Current Activity*

A one byte value indicating the percentage of time during the last xx mains voltage cycles that one of the current limits above actually limited the input current. This block is only defined for PLM20000Q and is filled with zeros otherwise. The value can be used as a scale factor for PSU power meter roofs.

Value	Description
0x65 – 0xFF	Reserved
0x64	100 %
0x63	99 %
….	….
0x01	1 %
0x00	0 %

9.2.2.12 *PSU Peak Current*

A one byte value indicating the maximum value of the mains input peak current. This block is only defined for PLM20000Q and is filled with zeros otherwise. The value is intended for PSU power meters.

Value	Description
0xFF	63.75 A
0xFE	63.50 A
….	….
0x01	0.25 A
0x00	0 A

9.2.2.13 *PSU Average Current*

A one byte value indicating the maximum value of the mains input average current. This block is only defined for PLM20000Q and is filled with zeros otherwise. The value is intended for PSU power meters.

Value	Description
0xFF	63.75 A
0xFE	63.50 A
….	….
0x01	0.25 A
0x00	0 A

9.2.2.14 *PSU Peak Power*

A one byte value indicating the maximum value of the mains input peak power. This block is only defined for PLM20000Q and is filled with zeros otherwise. The value is intended for PSU power meters.

Value	Description
0xFF	8160 W
0xFE	8128 W
….	….
0x01	32 W
0x00	0 W

9.2.2.15 *PSU Average Power*

A one byte value indicating the maximum value of the mains input average power. This block is only defined for PLM20000Q and is filled with zeros otherwise. The value is intended for PSU power meters.

Value	Description
0xFF	8160 W
0xFE	8128 W
….	….
0x01	32 W
0x00	0 W

9.2.2.16 *Channel Temp Limit*

A one byte value indicating the ISVPL threshold decrease factor for module temp SMGO in dB relative the ISVPL threshold for each channel. The value is intended as a roof in output meters.

Value	Description
0xFF	Reserved
0xFE	-25.4 dB ISVPL threshold decrease
….	….
0x01	-0.1 dB ISVPL threshold decrease
0x00	No ISVPL threshold decrease

9.2.2.17 *PSU Temp Limit*

A one byte value indicating the ISVPL threshold decrease factor for PSU temp SMGO in dB relative the ISVPL threshold for each channel. This block is only defined for PLM20000Q and is filled with zeros otherwise. The value is intended as a roof in output meters.

Value	Description
0xFF	Reserved
0xFE	-25.4 dB ISVPL threshold decrease
….	….
0x01	-0.1 dB ISVPL threshold decrease
0x00	No ISVPL threshold decrease

9.2.2.18 *PSU Vcap Limit*

A one byte value indicating the ISVPL threshold decrease factor for Vcap SMGO in dB relative the ISVPL threshold for each channel. This block is only defined for PLM20000Q and is filled with zeros otherwise. The value is intended as a roof in output meters.

Value	Description
0xFF	Reserved
0xFE	-25.4 dB ISVPL threshold decrease
….	….
0x01	-0.1 dB ISVPL threshold decrease
0x00	No ISVPL threshold decrease

9.2.2.19 *PSU Status Block*

PSU fault and warning flags in a 16-bit data block. This block is only defined for PLM20000Q and is filled with zeros otherwise.

<table>
<tr><th>Bit number</th><th>Parameter</th><th>Severity</th><th>Description</th></tr>
<tr><td>15-12</td><td>PSU Needs Service reason</td><td>Fault</td><td>The PSU shutdown reason when non-zero. The mains power needs to be cycled in order to recover.
<table>
<tr><th>Value</th><th>INFO TEXT</th></tr>
<tr><td>0</td><td>-</td></tr>
<tr><td></td><td>No error, normal mode</td></tr>
<tr><td>1</td><td>NEEDS SERVICE:1</td></tr>
<tr><td>2</td><td>NEEDS SERVICE:2</td></tr>
<tr><td>3</td><td>NEEDS SERVICE:3</td></tr>
<tr><td>4</td><td>NEEDS SERVICE:4</td></tr>
<tr><td>5</td><td>NEEDS SERVICE:5</td></tr>
<tr><td>6</td><td>NEEDS SERVICE:6</td></tr>
<tr><td>7</td><td>NEEDS SERVICE:7</td></tr>
<tr><td>8</td><td>NEEDS SERVICE:8</td></tr>
</table></td></tr>
<tr><td rowspan="2">11</td><td rowspan="2">PSU Mains Glitch</td><td rowspan="2">Warning</td><td>MAINS GLITCH</td></tr>
<tr><td>A glitch on the mains voltage was detected. PSU was not turned off.</td></tr>
<tr><td rowspan="2">10</td><td rowspan="2">PSU Safe mode</td><td rowspan="2">Fault</td><td>PSU SAFE MODE</td></tr>
<tr><td>N/A</td></tr>
<tr><td rowspan="2">9</td><td rowspan="2">Check AC Mains</td><td rowspan="2">Fault</td><td>CHECK AC MAINS</td></tr>
<tr><td>Retry in 10 seconds</td></tr>
<tr><td rowspan="2">8</td><td rowspan="2">PSU Power Limit</td><td rowspan="2">Warning</td><td>PAL</td></tr>
<tr><td>The output power is limited due to a mains input current limitation imposed by the maximum allowed input power</td></tr>
<tr><td rowspan="2">7</td><td rowspan="2">PSU Temp Limit</td><td rowspan="2">Warning</td><td>PTL</td></tr>
<tr><td>The output power is limited due to an overheat condition in the PSU</td></tr>
<tr><td rowspan="2">6</td><td rowspan="2">Amp Temp Limit</td><td rowspan="2">Warning</td><td>ATL</td></tr>
<tr><td>The output power is limited due to an overheat condition in one or more output amplifier modules.</td></tr>
<tr><td rowspan="2">5</td><td rowspan="2">Fuse Current Limit</td><td rowspan="2">Warning</td><td>BEL</td></tr>
<tr><td>The output power is limited due to a mains input current limitation induced by the fuse model</td></tr>
<tr><td rowspan="2">4</td><td rowspan="2">Under Voltage Limit</td><td rowspan="2">Warning</td><td>UVL</td></tr>
<tr><td>The output power is limited due to a mains input current limitation that is caused by a mains under voltage condition</td></tr>
<tr><td rowspan="2">3</td><td rowspan="2">PSU Rail Protect</td><td rowspan="2">Fault</td><td>PSU POWER PROT</td></tr>
<tr><td>Retry in 10 seconds</td></tr>
</table>

2	Mains over voltage peak	Fault	**MAINS >400Vpk**
			Mains peak voltage is too high for continued operation
1	Mains over voltage RMS	Fault	**MAINS >270V**
			Mains RMS voltage is too high for continued operation
0	Mains Under voltage	Fault	**MAINS <65V**
			Mains RMS voltage is too low for continued operation

9.2.2.20 *Channel Ext Status*

These 8 bit flags per amplifier channel extend the existing channel status with reserved bits for future use.

Bit number	Parameter	Severity	Description
0-7	Reserved		Reserved for future use

9.2.2.21 *Amp Ext Status*

These 32 bit flags per amplifier channel extend the existing channel status with reserved bits for future use.

Bit number	Parameter	Severity	Description
0-31	Reserved		Reserved for future use

9.3 LM protocol Version

Field	Name	Size	Comment
uStatus	Status information	4 Bytes	Status of host. See 9.3.1
(Spare)	(Reserved field)	4 Bytes	Reserved for future use.
uPeakRtrOut1	Peak value router out 1	1 Bytes	Peak values for input routers. 0xFF=ADO/AES fullscale (overshoot), 0xFE=-0.5dBFS, 0x01=-127.0dBFS, 0x00=<-127dBFS
uPeakRtrOut2	Peak value router out 2	1 Bytes	
uPeakRtrOut3	Peak value router out 3	1 Bytes	
uPeakRtrOut4	Peak value router out 4	1 Bytes	
uPeakModInA	Peak value module in A	1 Bytes	Peak values for module inputs. 0xFF=ADO/AES fullscale (overshoot), 0xFE=-0.5dBFS, 0x01=-127.0dBFS, 0x00=<-127dBFS
uPeakModInB	Peak value module in B	1 Bytes	
uPeakModOut1	Peak value module out 1	1 Bytes	Peak values for module outputs. 0xFF=ADO/AES fullscale (overshoot), 0xFE=-0.5dBFS, 0x01=-127.0dBFS, 0x00=<-127dBFS
uPeakModOut2	Peak value module out 2	1 Bytes	
uPeakModOut3	Peak value module out 3	1 Bytes	
uPeakModOut4	Peak value module out 4	1 Bytes	
uPeakModOut5	Peak value module out 5	1 Bytes	
uPeakModOut6	Peak value module out 6	1 Bytes	
uPeakGrModOut1	Peak g.r.v. module out 1	1 Bytes	Peak gain reduction values for module outputs. 0xFF = 25.5 dB gain reduction 0xFE = 25.0 dB, 0x01 = 0.1 dB
uPeakGrModOut2	Peak g.r.v. module out 2	1 Bytes	
uPeakGrModOut3	Peak g.r.v. module out 3	1 Bytes	
uPeakGrModOut4	Peak g.r.v. module out 4	1 Bytes	

uPeakGrModOut5	Peak g.r.v. module out 5	1 Bytes	0x00 = 0 dB gain reduction
uPeakGrModOut6	Peak g.r.v. module out 6	1 Bytes	
uRMSRtrOut1	RMS value router out 1	1 Bytes	RMS values for input routers. 0xFF=ADO/AES fullscale (overshoot), 0xFE=-0.5dBFS, 0x01=-127.0dBFS, 0x00=<-127dBFS
uRMSRtrOut2	RMS value router out 2	1 Bytes	
uRMSRtrOut3	RMS value router out 3	1 Bytes	
uRMSRtrOut4	RMS value router out 4	1 Bytes	
uRMSModInA	RMS value module in A	1 Bytes	RMS values for module inputs. 0xFF=ADO/AES fullscale (overshoot), 0xFE=-0.5dBFS, 0x01=-127.0dBFS, 0x00=<-127dBFS
uRMSModInB	RMS value module in B	1 Bytes	
uRMSModOut1	RMS value module out 1	1 Bytes	RMS values for module outputs. 0xFF=ADO/AES fullscale (overshoot), 0xFE=-0.5dBFS, 0x01=-127.0dBFS, 0x00=<-127dBFS
uRMSModOut2	RMS value module out 2	1 Bytes	
uRMSModOut3	RMS value module out 3	1 Bytes	
uRMSModOut4	RMS value module out 4	1 Bytes	
uRMSModOut5	RMS value module out 5	1 Bytes	
uRMSModOut6	RMS value module out 6	1 Bytes	
(Spare)	(Reserved field)	6 Bytes	Reserved for future use.

ModOut 1-6 are the more logical order of dsp processing, in a 3+3 1-3 is for modA and 4-6 is for modB, in a 4+2 1-4 is for modA and 5-6 is for modB, in 2+2 1-2 is for modA and 4-5 is for modB.

9.3.1 uStatus field

uStatus	Bit	Comment
LINK_1	0	Ethernet 1 link status, 1=Network connected (Link), 0=No connection to network
LINK_2	1	Ethernet 2 link status, 1=Network connected (Link), 0=No connection to network
RESERVED	2-3	Unused bits should be set to 0
CTRL_PRESENT	4	1=Controller present on network, 0=No controller present on network
AD_DATA_FAIL	5	1=AD data encoding violation, 0=No violation.
NO_INPUT_SOURCE	6	1=No input source available in any of the input routers, 0=Both have input
RESERVED	7	Unused bits should be set to 0
CLOCK_SLIP_AES_1	8	1=AES1 clock slipping, 0=Clock ok
CLOCK_SLIP_AES_2	9	1=AES2 clock slipping, 0=Clock ok
CLOCK_SLIP_DANTE	10	1=DANTE clock slipping, 0=Clock ok
PROT_MUTE_STATE	11	1=Muted by GPI protection mute state, 0=Normal operation
DANTE_MASTER	12	1=Unit is DANTE clock master, 0=Slave
FAN_ALARM	13	1=Fan is broken, 0=Normal operation
TEMP_WARNING	14	1=Temperature warning, 0=Temperature is below threshold
OVERTEMP	15	1=Temperature is too high, 0=Temperature is below threshold
RESERVED	16-31	Unused bits should be set to 0

10 FAQ

1. **Question**:
A Source ID for the sender is mentioned. Is this something we generate ourselves? I.e. we'll have our control system talking to the units, so does this mean we generate a unique 64 bit ID for it ourselves? It also mentions the ID being incremented each time, does this mean we have to start every message we send with a new ID number?

Answer:
You have to generate the source ID yourself and it can be any random number. Once generated it can remain the same for the whole session. Just remember to increase the value of the message ID (the last field in the packet header) by one for each message you send so that each message gets a unique ID.

2. **Question**:
Source and Destination Class, use 6 and 5 or 0 respectively. Is the system looking for literally (06) and (05)/(00), or does it need the Hex representation of the Ascii code for these numbers? (36) and (35)/(30) respectively.

Answer:
You must enter them literally, i.e. use (06) and (05)/(00).

3. **Question**:
The Documentation mentions the Packet Footer being a reserved 4byte value, but doesn't seem to mention what that reserved value is.

Answer:
The packet footer is unused for the DLM message type (packet type 701) so it doesn't matter what you enter into this field. You could set it to zero if you like.

4. **Question**:
Are we working with 2 separate UDP ports, 6004 for receiving, and 6015 for transmitting?

Answer:
The PLM and LM26 will send messages to port 6004 so you must use this port for the receiving socket.
The transmitting socket can use any port but you must send the messages to port 6015 since this is the port that the PLM listens to.
Also, please have a look at appendix C in the DLM control protocol documentation for an example application written in C code.

5. **Question**:
Is there any reason for the value of 120 instead of 528 for the payload size? Presumably this just reflects the maximum payload size of actual messages used within this particular application?

Answer:
Correct, 120 is the maximum payload for current messages. 560 is the maximum size for the whole packet.

6. **Question**:
It is not stated in the protocol doc but it looks from the test code that the payload size should always be a multiple of 4 bytes (see macro (DLM_ALIGN_SIZE)), is this correct?

Answer:
No, you don't need to align the size. The unit will align the response e.g. "Dev.Power?" will have sLength set to 36 instead of 33 (footer + header = 32).

7. **Question**:
Msg_Ack result code values, the test code has some different result codes defined (-7 to -21) and doesn't have the ones listed in the document (-3 to -6). What is the definitive result code list?

Answer:
Suggest that you only check for ACK_SUCCESS (-2L), all the others are more or less internal error codes i.e. can be good for us to know if you have a specific problem. If something fails you send the command again.

8. **Question**:
Is there a document I can read that explains the gain structure of a PLM?
Answer:
Yes, please see section “Signal Flow and Lake Processing” in;
For PLM: PLM Series Operation Manual
For LM26: LM26 Operation Manual

9. **Question**:
When using the Dev.Preset.Recall, is there any way to interrogate the PLM to determine what preset is currently active?
Answer:
No, the PLM and LM26 don’t know which frame preset that is active. But, if the frame Label is used i.e. with different frame labels for different presets, then this can be used to determine which preset that currently is active.

10. **Question**:
Is the current preset remembered after a PLM power cycle?
Answer:
Yes, all parameters are remembered. The PLM does however not remember what preset number currently is active.

11. **Question**:
Is there any problem in repeatedly sending the Dev.Preset.Recall command with the same preset value, i.e. would this result in any audible artifacts?
Answer:
Yes, the audio is muted during a preset recall since all the DSP parameters is set. This will have audio muted during about 1 second. And since the PLM doesn’t know which preset that is active, it will do this for every time the command is received.

12. **Question**:
When sending to a specific PLM unit I presume that the Destination Class should be set to 5 for both Unicast and Broadcast cases?
Answer:
Yes.

13. **Question**:
When sending to all PLM units I presume that the Destination ID is 'don't care' as implied by section 4.1.2 ("the application doesn't need to set this").
Answer:
Yes.

14. **Question**:
I have done a DLM implementation using the 3rd party protocol for PLM Series version 1 with PLM Series firmware 2.69. Can I still use my 3rd party control module if I update to PLM Series firmware 2.75?
Answer:
Yes, all commands for version 1 are still valid with firmware 2.75, and it will answer in the same way as before. If version 2 is wanted, the command needs to be extended to also include a “2” to actively choose that it is the extended version 2 that is wanted.

15. **Question**:
Can I monitor PSU Mains input with my PLM 10000Q or 14000 via DLM?
Answer:
No, PSU Mains input is only available on the PLM 20000Q with firmware above 2.74.

16. **Question**:
I want to send a single broadcast UDP command to all online units, and I don't care about getting an acc.

Answer:

There is an example of this in the example code section, “Transmit heartbeat”, where the header can be re-used except for packet length and type.

17. **Question**:

I can’t get the communication to work, and I have double checked my commands and everything is in order.

Answer:

Confirm that you are on the same subnet with your Controller as the devices you are trying to control.

11 Appendix C: Example application source code

This application allows you to send any of the command strings (from 8.2) to a single PLM using unicast (you can also use broadcast by replacing SENDUDP_UNICAST with SENDUDP_BROADCAST). The application takes three arguments; adapter IP (on the host machine), unique 64bits ID and the message. For example, to set the frame label you need to type this on the command line:
>dlmTest 10.10.10.116 3d000011:d6ed9201 "Dev.FrameLabel=testName"

We can also fetch the frame label by typing:
>dlmTest 10.10.10.116 3d000011:d6ed9201 "Dev.FrameLabel?"

```
/*============================================================================
  Console application for sending DLM protocol messages to a single unit.
============================================================================*/

#include <windows.h>
#include <stdio.h>

/*============================================================================
Various defs and structures
============================================================================*/
#define SENDUDP_BROADCAST   true  // Use this in order to send broadcast
#define SENDUDP_UNICAST     false // Use this in order to send unicast
#define RESPTYPE_NORESP     -1          // Ignore response

typedef struct in_addr   MYINADDR;
typedef struct in_addr *PMYINADDR;

#define MAX_NETWORKBINDING_DESCR   128
struct Network_Binding
{
    // A user-friendly description of the connection
    char szDescription[MAX_NETWORKBINDING_DESCR];

    // IP address of the connection. If 0, then this connection is
    // currently not connected (eg: network cable unplugged).
    unsigned long ulIpAddr;
};
typedef struct Network_Binding   NETWORKBINDING;
typedef struct Network_Binding *PNETWORKBINDING;

#define MAX_NETWORKBINDING 10
struct Network_Info
{
    unsigned int nBindingsFound;
    NETWORKBINDING Network_Bindings[MAX_NETWORKBINDING];
};
typedef struct Network_Info   NETWORKINFO;
typedef struct Network_Info *PNETWORKINFO;

/*============================================================================
Structures and definitions describing PLM messages.
A PLM message comprises, in order: header + payload + footer
============================================================================*/
#define BROADCAST_CLASSID    0
#define PLM_CLASSID          5
#define HOST_CLASSID        6

#define TYPE1_CLASS_MASK     0x8000
#define TYPE2_CLASS_MASK     0x4000
#define TYPE3_CLASS_MASK     0x2000
#define ALL_CLASS_MASK (TYPE1_CLASS_MASK | TYPE2_CLASS_MASK | TYPE3_CLASS_MASK)

#define BRDCAST_PRODUCT_PLM10000Q  0x4
#define BRDCAST_PRODUCT_PLM14000   0x6
#define BRDCAST_PRODUCT_PLM20000Q  0x9
#define BRDCAST_PRODUCT_LM26       0xa

#define MAX_MESSAGE_SIZE_CHAR (140 * 4)
```

```
#define BROADCAST_IDHI          -2
#define BROADCAST_IDLO          -3

#define HOST_PORT_RCV 6004 // DLM Host listens on this port
#define HOST_PORT_SND 6015 // DLM Host sends on this port. Other ports can be used
#define DLM_PORT_RCV  6015 // DLM Unit listens on this port

/* Message Header */
struct MSG_CmdHdr
{
      long lSrcIDHi;        // Hi 32 bits of the unique 64-bit source ID
      long lSrcIDLo;        // Lo 32 bits of the unique 64-bit source ID
      long lDestIDHi;       // Hi 32 bits of the unique 64-bit dest ID
      long lDestIDLo;       // Lo 32 bits of the unique 64-bit dest ID
      short sSrcClass;      // Source class ID
      short sDestClass;     // Dest class ID
      short sLength;        // Total msg length, in bytes, incl. header, payload and footer
      short sType;          // Msg type
      long lMsgID;          // Message ID
};
typedef struct MSG_CmdHdr MSGCMDHDR;
typedef struct MSG_CmdHdr *PMSGCMDHDR;

/* Message Footer */
struct MSG_CmdFtr
{
      long    lChecksum;
};
typedef struct MSG_CmdFtr MSGCMDFTR;
typedef struct MSG_CmdFtr *PMSGCMDFTR;

/* Msg types */
#define Msg_Ack                 2
#define Msg_BroadcastID4
#define Msg_DLMMsg              701

/* Ack response codes */
#define ACK_SUCCESS         -2    /* Success */

/* DLM msg payload size (bytes) */
#define MAX_DLMPROTO_PAYLOAD_LEN  120

/* DLM pkt alignment */
#define DLM_PKT_ALIGNMENT     4
#define DLM_ALIGN_SIZE(sz) ((sz + (DLM_PKT_ALIGNMENT-1)) & (~(DLM_PKT_ALIGNMENT-1)))

/* Ack response */
struct MSG_Ack
{
   MSGCMDHDR hdr;
   long lResult;
   MSGCMDFTR ftr;
};
typedef struct MSG_Ack MSGACK;
typedef struct MSG_Ack *PMSGACK;

/* DLM unit BroadcastID msg */
struct MSG_BroadcastID {
   MSGCMDHDR hdr;
      long lPad1[9];
      long lFlags[4];
      long lPad2[4];
   MSGCMDFTR ftr;
};
typedef struct MSG_BroadcastID MSGBROADCASTID;
typedef struct MSG_BroadcastID *PMSGBROADCASTID;

/* DLM message */
struct MSG_DLMMsg
{
   MSGCMDHDR     hdr;
      char       szMsg[DLM_ALIGN_SIZE(MAX_DLMPROTO_PAYLOAD_LEN)];
   MSGCMDFTR     ftr;
};
typedef struct MSG_DLMMsg MSGDLMMSG;
typedef struct MSG_DLMMsg *PMSGDLMMSG;
```

```
/*============================================================================
Structure describing a networked DLM unit
============================================================================*/
#define MAXSZFRAMENAME 16
struct Dlm_Unit
{
	short		ClassId;
	long		IdHi;
	long		IdLo;
	char		Name[MAXSZFRAMENAME];
	unsigned long	IP;
};
typedef struct Dlm_Unit   DLMUNIT;
typedef struct Dlm_Unit *PDLMUNIT;

/*============================================================================
Function prototypes
============================================================================*/
void fillHeader(PMSGCMDHDR pHdr, short sLength, short sType, bool bBroadcast);
long lGenerateChecksum(PMSGCMDHDR pMsg);

/*============================================================================
Global data
============================================================================*/
SOCKET TranSocket;		// Transmit DLM msgs on this socket
SOCKET RecvSocket;		// Receive  DLM msgs on this socket
DLMUNIT gDlmUnit;	// The unit we are trying to address
unsigned long HostIP;	// IP of the DLM host
long HostIdHi;		// High 32-bit of 64-bit host ID
long HostIdLo;		// Lo   32-bit of 64-bit host ID
long HostMsgId;			// Unique ID for every transmitted msg

// Info about the various network connections available on the host machine.
NETWORKINFO MyNetworkInfo;

// Buffer for received UDP datagrams
char rxUdp[MAX_MESSAGE_SIZE_CHAR];

/*============================================================================
Initialise the 64-bit host ID using the system time (to make it unique). This could
actually be any number but since this application exits after one command 'HostMsgId'
won't be increased correctly (if we run it more than once).
============================================================================*/
void GetHostID(void)
{
	if ((HostIdHi != 0) && (HostIdLo != 0))	{return;}

	SYSTEMTIME CurrentTime;
	GetLocalTime(&CurrentTime);
	FILETIME CurrentFileTime;
	SystemTimeToFileTime(&CurrentTime, &CurrentFileTime);
	HostIdHi = CurrentFileTime.dwHighDateTime;
	HostIdLo = CurrentFileTime.dwLowDateTime;

	return;
}

/*============================================================================
Session initialisation.
============================================================================*/
void DlmTestInit(void)
{
	TranSocket = INVALID_SOCKET;
	RecvSocket = INVALID_SOCKET;

	memset(&gDlmUnit, 0, sizeof(gDlmUnit));
	memset(&MyNetworkInfo, 0, sizeof(NETWORKINFO));

	HostIP    = inet_addr("0.0.0.0");
	HostIdHi  = 0;
	HostIdLo  = 0;
	HostMsgId = 1;

	GetHostID();
}
```

```
/*============================================================================
Generate DLM msg checksum
============================================================================*/
long lGenerateChecksum(PMSGCMDHDR pMsg)
{
        int i;
    unsigned long lCheckSum = 0L;
    for(i=0;i< (int)((pMsg->sLength >> 1)-sizeof(unsigned short)); i++) {
            lCheckSum = (lCheckSum<<1) | ( ((lCheckSum>>31)&1) ^ ((lCheckSum>>18)&1) );
        lCheckSum ^= (unsigned long)(((unsigned short *)(pMsg))[i]);
    }

    return((long)lCheckSum);
}

/*============================================================================
Read a DLM msg from the UDP port. Returns TRUE if it's a properly-formed DLM msg
whose 64-bit ID either matches the host ID or is the special 'broadcast' ID.
============================================================================*/
bool bReadUDP(char *RxMsg, unsigned long nBytesMax, unsigned long *nBytes, PMYINADDR pIpRx)
{
        SOCKADDR_IN sa;
        int nSize = sizeof(SOCKADDR_IN);
        PMSGCMDHDR pHdr;

        int nBytesRcvd = recvfrom(      RecvSocket,
                                        RxMsg,
                                        nBytesMax,
                                        0,
                                        (SOCKADDR FAR *)&sa,
                                        &nSize);

    if (   nBytesRcvd == SOCKET_ERROR
            || nBytesRcvd < sizeof(MSGCMDHDR)
            || sa.sin_addr.s_addr == HostIP )     // Ignore anything from the host address
        {
            goto bReadUDP_Error;
        }

        pHdr = (PMSGCMDHDR) RxMsg;


        if (   pHdr->sSrcClass & ALL_CLASS_MASK    // PLMs don't have a class mask bit
            || (pHdr->lSrcIDHi == HostIdHi  &&  pHdr->lSrcIDLo == HostIdLo)    // Ignore anything from host
            || pHdr->sLength <= sizeof(MSGCMDHDR)
            || pHdr->sLength > MAX_MESSAGE_SIZE_CHAR
            || pHdr->sLength != nBytesRcvd )
        {
            goto bReadUDP_Error;
        }

        if ( !(   (pHdr->lDestIDHi == BROADCAST_IDHI  &&  pHdr->lDestIDLo == BROADCAST_IDLO)
                || (pHdr->lDestIDHi == HostIdHi       &&  pHdr->lDestIDLo == HostIdLo      ))
          )
        {
            goto bReadUDP_Error;
        }

        *nBytes = (unsigned long)nBytesRcvd;
        if(pIpRx)   {*pIpRx = sa.sin_addr;}
        return true;

bReadUDP_Error:

    *nBytes = 0;
    return false;
}

/*============================================================================
Write a DLM msg to the UDP port. Returns true on success otherwise false.
============================================================================*/
bool bWriteUDP(char *TxMsg, unsigned long nBytes, bool bBroadcast)
{
    if (nBytes > MAX_MESSAGE_SIZE_CHAR)
        {
```

```
            printf("Length of packet too long\n");
            return false;
    }

        SOCKADDR_IN sa;
        sa.sin_family = AF_INET;
        sa.sin_port   = htons(DLM_PORT_RCV);

        if (bBroadcast)
            sa.sin_addr.s_addr = htonl(INADDR_BROADCAST);
        else
            sa.sin_addr.s_addr = gDlmUnit.IP;

        int nBytesSent = sendto(TranSocket,
                                (const char *)TxMsg,
                                nBytes,
                                0,
                                (SOCKADDR *)&sa,
                                sizeof(SOCKADDR_IN));

        if (nBytesSent == SOCKET_ERROR  ||  nBytesSent < (int)nBytes)
        {
            printf("Error from sendto\n");
            return false;
    }

    return true;
}


/*=============================================================================
Close sockets
=============================================================================*/
void CloseUDP(void)
{
        if(TranSocket != INVALID_SOCKET)        {closesocket(TranSocket);}
        TranSocket = INVALID_SOCKET;
        if(RecvSocket != INVALID_SOCKET)        {closesocket(RecvSocket);}
        RecvSocket = INVALID_SOCKET;
}

/*=============================================================================
Flush receive socket
=============================================================================*/
void FlushUDP(void)
{
        unsigned long DataSize=1;
    SOCKADDR_IN sa;
        int nSize = sizeof(SOCKADDR_IN);

        while (DataSize)
        {
            ioctlsocket(RecvSocket, FIONREAD, &DataSize);
            if (DataSize)
            {
                recvfrom(   RecvSocket
                            ,rxUdp
                            ,MAX_MESSAGE_SIZE_CHAR
                            ,0
                            ,(SOCKADDR FAR *)&sa
                            ,&nSize);
            }
        }
}

/*=============================================================================
Check if data available on receive socket
=============================================================================*/
unsigned long IsRcvRdy(void)
{
        unsigned long DataSize=1;
        ioctlsocket(RecvSocket,FIONREAD,&DataSize);
        return DataSize;
}

/*=============================================================================
```

```
Fill DLM msg header
==========================================================================*/
void fillHeader(PMSGCMDHDR pHdr, short sLength, short sType, bool bBroadcast)
{
	pHdr->lDestIDHi  = bBroadcast ? 0x12345678 : gDlmUnit.IdHi; // Not used for broadcast
	pHdr->lDestIDLo  = bBroadcast ? 0x9abcdef0 : gDlmUnit.IdLo;	// Not used for broadcast
	pHdr->lSrcIDHi   = HostIdHi;
	pHdr->lSrcIDLo   = HostIdLo;
	pHdr->sDestClass = bBroadcast ? BROADCAST_CLASSID : gDlmUnit.ClassId;
	pHdr->sSrcClass  = HOST_CLASSID;
	pHdr->sLength    = sLength;
	pHdr->sType      = sType;
	// Hosts must increment msg ID with each new msg sent to a particular unit.
	// It's OK to use the same msg ID when sending the same payload to multiple units.
	// For this application we actually don't need to do this since we have a unique source id.
	pHdr->lMsgID     = HostMsgId++;
}

/*==========================================================================
Send a DLM msg and look for a response of the specified type. If the response is
found it returns a pointer to the response otherwise NULL.
==========================================================================*/
PMSGCMDHDR sendDlmMsg(PMSGCMDHDR TxMsg, short respType, bool bBroadcast, bool bAllowAck)
{
	const int nSendRetries = 3;
	const int nReceiveRetries = 10;

	if (respType < 0)
	{
		FlushUDP();
		bWriteUDP((char *)TxMsg, TxMsg->sLength, bBroadcast);
		return NULL;
	}

	for (int s = 0; s < nSendRetries; s++)
	{
		FlushUDP();
		if (!bWriteUDP((char *)TxMsg, TxMsg->sLength, bBroadcast))
		{
			Sleep(100);
			continue;
		}

		for (int r = 0; r < nReceiveRetries; r++)
		{
			unsigned long nBytesRx = 0;
			if ( IsRcvRdy()  &&  bReadUDP(rxUdp, MAX_MESSAGE_SIZE_CHAR, &nBytesRx, NULL) )
			{
				PMSGCMDHDR RxMsg = (PMSGCMDHDR) rxUdp;
				if (   RxMsg->lMsgID == TxMsg->lMsgID
					&& (RxMsg->sType  == respType  ||  (RxMsg->sType == Msg_Ack && bAllowAck))
					/* Ignore heartbeat and meter broadcasts from the units. */
					&& !( (RxMsg->lDestIDHi == BROADCAST_IDHI)  &&  (RxMsg->lDestIDLo == BROADCAST_IDLO) ) )
				{
					return RxMsg;
				}
			}

			Sleep(100);
		}
	}

	return NULL;
}

/*==========================================================================
Broadcasts a heartbeat message on the network to signal the presence of a host.
==========================================================================*/
void TransmitHeartbeat(void)
{
	// Setup the header of the packet
	MSGBROADCASTID message;
	memset(&message, 0, (sizeof(message)));

	message.hdr.lDestIDHi = BROADCAST_IDHI;		// Message destination is all devices on the network
	message.hdr.lDestIDLo = BROADCAST_IDLO;
```

```
	message.hdr.lSrcIDHi = HostIdHi;			// Dummy ID of the sender
	message.hdr.lSrcIDLo = HostIdLo;
	message.hdr.sSrcClass = HOST_CLASSID;		// Packet sent from a Host
	message.hdr.sDestClass = BROADCAST_CLASSID;	// Packet sent to all device types
	message.hdr.sType = Msg_BroadcastID;		// Broadcast ID Msg packet
	message.hdr.sLength = sizeof(message);		// Size of packet in bytes
	message.hdr.lMsgID = HostMsgId++;
	message.ftr.lChecksum = lGenerateChecksum(&message.hdr);	// Calculate checksum

	// Send message
	sendDlmMsg( (PMSGCMDHDR)&message, RESPTYPE_NORESP, SENDUDP_BROADCAST, FALSE);
}

/*============================================================================
Listen on the network for DLM units (heartbeat) and look for the unique 64-bit ID.
Returns true if the unit is found, false otherwise.
============================================================================*/
bool bFindDlmUnit(const char* szFrameId)
{
	if (HostIP == inet_addr("0.0.0.0")  ||  TranSocket == INVALID_SOCKET  ||  RecvSocket == INVALID_SOCKET)
		return false;

	unsigned long nBytes=0;
	PMSGBROADCASTID pBcastId = (PMSGBROADCASTID) rxUdp;
	MYINADDR unitaddr;

	FlushUDP(); //Flush all buffered data since we last scanned the receive socket

	/* Transmit a heartbeat. All units that receive a heartbeat will start transmitting heartbeats for 10s. */
	TransmitHeartbeat();

	for(int i = 0; i < 300; i++)
	{
		if (   IsRcvRdy()
			&& bReadUDP(rxUdp, MAX_MESSAGE_SIZE_CHAR, &nBytes, &unitaddr)
			&& pBcastId->hdr.sType == Msg_BroadcastID
			&& (   pBcastId->lFlags[2] == BRDCAST_PRODUCT_PLM10000Q
			    || pBcastId->lFlags[2] == BRDCAST_PRODUCT_PLM14000
				|| pBcastId->lFlags[2] == BRDCAST_PRODUCT_PLM20000Q
				|| pBcastId->lFlags[2] == BRDCAST_PRODUCT_LM26)
				&& ( (pBcastId->hdr.sSrcClass & ~ALL_CLASS_MASK) == PLM_CLASSID)
		  )
		{
			char buf[32];
			sprintf(buf, "%x:%x", pBcastId->hdr.lSrcIDHi, pBcastId->hdr.lSrcIDLo);
			if (strcmp(buf, szFrameId) == 0 )
			{
				// Correct unit found
				gDlmUnit.ClassId = pBcastId->hdr.sSrcClass & ~ALL_CLASS_MASK;
				gDlmUnit.IdHi    = pBcastId->hdr.lSrcIDHi;
				gDlmUnit.IdLo    = pBcastId->hdr.lSrcIDLo;
				gDlmUnit.IP      = unitaddr.s_addr;

				struct in_addr UnitAddr;
				UnitAddr.s_addr = gDlmUnit.IP;
				printf("Found %08x:%08x (IP = %s)\n", pBcastId->hdr.lSrcIDHi, pBcastId->hdr.lSrcIDLo, inet_ntoa(UnitAddr));
				return true;
			}
		}

		Sleep(5);
	}

	printf("%s was not found on the network!\n", szFrameId);
	return false;
}

/*============================================================================
Initialise UDP transport
============================================================================*/
bool InitialiseSockets(void)
{
	CloseUDP();

	/* Check for WinSock 2.2. Later versions may also work. */
	WORD wVersionRequested = 0x0202;
```

```
    WSADATA wsaData;
    int err;

    err = WSAStartup( wVersionRequested, &wsaData );
    if ( err != 0 )
    {
        printf("Error from WSAStarup - Check TCP/IP Installation\n");
        return false;
    }

    if (   LOBYTE( wsaData.wVersion ) != 2
        || HIBYTE( wsaData.wVersion ) != 2 )
    {
        printf("Windows Sockets DLL does not support 2.2\n");
        return false;
    }

TranSocket = socket(AF_INET, SOCK_DGRAM, IPPROTO_UDP);
RecvSocket = socket(AF_INET, SOCK_DGRAM, IPPROTO_UDP);
if (INVALID_SOCKET == TranSocket  ||  INVALID_SOCKET == RecvSocket)
    {
    CloseUDP();
        printf("Unable to create socket\n");
        return false;
}

    SOCKET MySocks[2];
    MySocks[0] = TranSocket;
    MySocks[1] = RecvSocket;
    BOOL bReuseAddr = TRUE;
    BOOL bBroadcast = TRUE;
    SOCKADDR_IN sa;

    for (int i=0; i<2; i++)
    {
        if (setsockopt
                    (MySocks[i]
                    ,SOL_SOCKET
                    ,SO_REUSEADDR
                    ,(const char*) &bReuseAddr
                    ,sizeof(bReuseAddr)
                    )
          )
        {
            CloseUDP();
            printf("setsockopt error\n");
            return false;
        }

        if (setsockopt
                    (MySocks[i]
                    ,SOL_SOCKET
                    ,SO_BROADCAST
                    ,(const char*) &bBroadcast
                    ,sizeof(bBroadcast)
                    )
          )
        {
            CloseUDP();
            printf("setsockopt error\n");
            return false;
        }

        sa.sin_family      = AF_INET;
        sa.sin_addr.s_addr = HostIP;
        if(MySocks[i] == TranSocket)   {sa.sin_port = htons(HOST_PORT_SND);}
        else                                    {sa.sin_port = htons(HOST_PORT_RCV);}

        if ( bind(MySocks[i], (struct sockaddr*) &sa, sizeof(sa)) )
        {
            CloseUDP();
            printf("Bind error\n");
            return false;
        }
    }
  return true;
```

```
}

/*============================================================================
Select connection on host machine
============================================================================*/
bool bSetNetworkBinding(unsigned int nBinding, char *szIP)
{
	if ( !szIP && nBinding >= MyNetworkInfo.nBindingsFound )
	{
		return false;
	}

	if (!szIP)	{ HostIP = MyNetworkInfo.Network_Bindings[nBinding].ulIpAddr; }
	else		{ HostIP = inet_addr(szIP); }

	if (HostIP == INADDR_NONE )
	{
		HostIP = inet_addr("0.0.0.0");
		return false;
	}

	/* Initialise the UDP transport */
	CloseUDP();
	int nRetry = 5;
	while (--nRetry > 0)
	{
		if (InitialiseSockets())	{break;}
		Sleep(250);
	}
	if (nRetry == 0)
	{
		CloseUDP();
		HostIP = inet_addr("0.0.0.0");
		return false;
	}
	FlushUDP();
	return true;
}

/*============================================================================
Send a DLM msg to the specified unit. Returns true if a successful response
otherwise false.
============================================================================*/
bool bHandleDlmMsg(char *szMsg)
{
	MSGDLMMSG DlmCmd;
	PMSGDLMMSG pDlmCmdRx;

	/* Construct the payload */
	memset(DlmCmd.szMsg, 0, DLM_ALIGN_SIZE(MAX_DLMPROTO_PAYLOAD_LEN));
	strncpy(DlmCmd.szMsg, szMsg, DLM_ALIGN_SIZE(MAX_DLMPROTO_PAYLOAD_LEN) - 1);

	printf("DLM msg is \"%s\"\n\n", DlmCmd.szMsg);

	/* Must fill header each time a new unit is selected in
	   order to pick up the correct 64-bit destination ID */
	fillHeader(&(DlmCmd.hdr), sizeof(MSGDLMMSG), Msg_DLMMsg, SENDUDP_UNICAST);

	/* Set checksum to zero for DLMMsg commands. */
	DlmCmd.ftr.lChecksum = 0;

	/* Send the msg */
	pDlmCmdRx = (PMSGDLMMSG) sendDlmMsg( (PMSGCMDHDR)&DlmCmd, Msg_DLMMsg, SENDUDP_UNICAST, true);

	/* Print status */
	if (pDlmCmdRx)
	{
		if (pDlmCmdRx->hdr.sType == Msg_DLMMsg)
			printf("%08x:%08x returned \"%s\"\n", gDlmUnit.IdHi, gDlmUnit.IdLo, pDlmCmdRx->szMsg);
		else if (pDlmCmdRx->hdr.sType == Msg_Ack)
		{
			long lResult = ((PMSGACK)pDlmCmdRx)->lResult;
			if (lResult == ACK_SUCCESS)
				printf("%08x:%08x command succeeded\n", gDlmUnit.IdHi, gDlmUnit.IdLo);
			else
			{
```

```
                    printf("%08x:%08x returned unsuccessful Ack (%i)\n", gDlmUnit.IdHi, gDlmUnit.IdLo, lResult);
                    return false;
                }
            }

            return true;
        }
        else
        {
            printf("%08x:%08x did not respond\n", gDlmUnit.IdHi, gDlmUnit.IdLo);
            return false;
        }
}

/*============================================================================
The application takes 3 input parameters: adapter IP, unique 64bits ID and the message
Here is an example (dlmTest is the name of the compiled .exe):
>dlmTest 10.10.10.116 3d000011:d6ed9201 "Dev.Power?"
============================================================================*/
int main(int argc, char* argv[])
{
    DlmTestInit();

    int nResult = 0;
    if (argc != 4 || !bSetNetworkBinding(0, argv[1]) || !bFindDlmUnit(argv[2]) || !bHandleDlmMsg(argv[3]))
        nResult = -1;

    CloseUDP();
    return nResult;
}
```