LivePremier™

AWJ Protocol Programmer's Guide

For firmware version v2.2 or higher

ANALOG WAY®

Pioneer in Analog, Leader in Digital

Table of contents

1. Presentation

1.1. Description

The AWJ protocol for LivePremier™ is a powerful way for you to automate your interaction with the LivePremier™ seamless switchers. The AWJ protocol for LivePremier™ is based on TCP/IP communication (port 10606) and uses JSON Patch commands to interact with the device. Up to 5 concurrent TCP clients can be connected to the same device. Before connecting to this port, please check that it has not been disabled by security on the Web RCS and that a firewall is not blocking it.

A LivePremier™ device should be considered as a state machine whose values are stored and organized inside a large JSON object. Changing a value in this JSON object immediately changes the state of the machine. The current state of the machine is always available by reading the JSON object properties. It is possible to use an AWJ command to read or modify one or more properties of the device.

The objective of this document is not to describe the entire LivePremier™ device JSON object model nor to list all the possible commands allowing to read or modify the corresponding values. The objective is however to list the most frequently used commands such preset recall, transition, layer source change, etc. This document refers to LivePremier™ firmware v2.2 or higher.

1.2. Syntax

JavaScript Object Notation (JSON) is a common format for the exchange and storage of structured data. JSON Patch is a format for expressing a sequence of operations to apply to a target JSON document.

AWJ protocol read or write commands must be surrounded by { } and must be terminated by the ASCII 0x04 character.

The commands MUST have exactly one "op" member, whose value indicates the operation to perform. Its value MUST be one of "get" (read command) or "replace" (write command).

Additionally, the commands MUST have exactly one "path" member, whose value MUST be a string containing a JSON path value that references the location within the device LivePremier™ JSON object to perform the operation.

The AWJ write commands MUST also have exactly one “value” member, whose value corresponds to the new value to be applied to the property or object defined by the "path'" member. For example:

```
{"op": "replace", "path": "/a/b/c", "value": "foo"}\0x04
```

Once an AWJ read command has been received and processed by the device, it will return a JSON string containing the value of the requested property or object. This string is surrounded by { } and is terminated by the ASCII 0x04 character as for the read or write commands.

This answer has exactly one "path" member, whose value is a string containing a JSON path value that references the location within the device JSON object for which the value was requested.

This answer has also exactly one "value" member, whose value corresponds to the value defined by the "path'" member. For example:

```
{"path": "/a/b/c", "value": "foo"}\0x04
```

1.3. Error messages

If the AWJ command you have sent cannot be processed, the device will return a message describing the reason, for example

```
{"error":{"code":"E12","message":"Unexpected path \\"DeviceObject/system/@props
/div\\""}}\0x04
```

The message contains an error code as well as message describing the error. The most common error codes are:

Code	Description
E09	Unexpected command JSON token
E10	Unexpected keywords. Keywords supported are “op”, “path” and “value”
E11	Unexpected operator. Operators supported are “get” and “replace”
E12	Unexpected path
E13	Unexpected value

1.4. Subscribing to machine state change notifications

By default, when the current state of the machine changes, the corresponding values are not forwarded to the connected TCP clients. But it is possible to subscribe to some of the machine state change notifications to be automatically notified when the value of one or more machine properties changes (new preset label, new layer source, etc..).

The TCP client's subscription list is empty by default, meaning that this TCP connection won’t receive any value/changes from the device. If the TCP client needs to receive some notifications/values from the device, the client must subscribe to the corresponding JSON path.

Reading subscription filters

```
{"op":"get","path":"Subscriptions"}\0x04
```

The machine returns:

```
{"path":"Subscriptions","value":[]}\0x04
```

Subscription filters modification

```
{"op":"replace","path":"Subscriptions",
"value":["DeviceObject/$screenGroup/@items/S1/control/@props",
"DeviceObject/$screenGroup/@items/S2",
"DeviceObject/$screenGroup/@items/S1/status/@props/presetStatus"]}\0x04
```

The machine returns:

```
{"path": "Subscriptions", "value": [ "DeviceObject/$screenGroup/@items/S2",
"DeviceObject/$screenGroup/@items/S1/control/@props",
"DeviceObject/$screenGroup/@items/S1/status/@props/presetStatus"]}\0x04
```

As soon as a PATH starts with one of the subscriptions, it will be sent to the client. If the PATH does not check any subscriptions, it will be filtered.

Example:

As soon as transition is requested for Screen 1 with the Web RCS, the "DeviceObject/$screenGroup/@items/S1/control/@props/xTake" will be transmitted.

Important: A GET made directly on a property is never filtered.

2. System commands

2.1. Reading the device type

Poll the type of the device

```
{"op":"get","path":"DeviceObject/system/@props/dev"}\0x04
```

The machine returns:

```
{"path":"DeviceObject/system/@props/dev","value":"NLC_RS4"}\0x04
```

Possible returned values are:

NLC_RSALPHA for the Aquilon RS Alpha
NLC_RS1 for the Aquilon RS1
NLC_RS2 for the Aquilon RS2
NLC_RS3 for the Aquilon RS3
NLC_RS4 for the Aquilon RS4
NLC_C for the Aquilon C
NLC_CPLUS for the Aquilon C+

2.2. Reading the device serial number

Poll the serial number of the device (XX9999 for ex)

```
{"op":"get","path":"DeviceObject/system/serial/@props/serialNumber"}\0x04
```

The machine returns:

```
{"path":"DeviceObject/system/serial/@props/serialNumber","value":"XX9999"}\0x04
```

2.3. Reading the device firmware version

Poll the firmware version of the device (2.2.80 for ex)

```
{"op":"get","path":"DeviceObject/system/version/@props/updater"}\0x04
```

The machine returns:

```
{"path":"DeviceObject/system/version/@props/updater","value":"2.2.80"}\0x04
```

2.4. Restarting the device

Perform a soft reboot of the unit

```
{"op":"replace","path":"DeviceObject/system/shutdown /cmd/@props/xRequest",
"value":"REBOOT"}\0x04
```

The device will not return a string

2.5. Shutting down the device (switch off)

Power the unit down (must be restarted manually)

```
{"op":"replace","path":"DeviceObject/system/shutdown /cmd/@props/xRequest",
"value":"SHUTDOWN"}\0x04
```

The device will not return a string

3. Screen/Auxiliary Screen commands

3.1. TAKE: Transitioning a Preview content to the Program

Take Screen 1

```
{"op":"replace","path":"DeviceObject/$screenGroup/@items/S1/control/@props/xTake",
"value":true}\0x04
```

The device will not return a string

3.2. Recalling a Screen Preset

Recall preset 33 on the preview of screen 1

```
{"op":"replace","path":"DeviceObject/presetBank/control/load/$slot/@items/33/$screen/
@items/S1/$preset/@items/PREVIEW/@props/xRequest","value":true}\0x04
```

The device will not return a string

Recall preset 13 on the program of screen 2

```
{"op":"replace","path":"DeviceObject/presetBank/control/load/$slot/@items/13/$screen/
@items/S2/$preset/@items/PROGRAM/@props/xRequest","value":true}\0x04
```

The device will not return a string

Recall preset 8 on the program of Auxiliary screen 1

```
{"op":"replace","path":"DeviceObject/presetBank/control/load/$slot/@items/8/$screen/
@items/A1/$preset/@items/PROGRAM/@props/xRequest","value":true}\0x04
```

The device will not return a string

3.3. Recalling a Master Preset

Recall master preset 15 to preview

```
{"op":"replace","path":"DeviceObject/masterPresetBank/control/load/$slot/@items/15/$preset/
@items/PREVIEW/@props/xRequest","value":true}\0x04
```

The device will not return a string

Recall master preset 3 to program

```
{"op":"replace","path":"DeviceObject/ masterPresetBank/control/load/$slot/@items/3/$preset/
@items/PROGRAM/@props/xRequest","value":true}\0x04
```

The device will not return a string

3.4. Reading Preset information

Poll for the name of Master Preset 3, which is labeled “Preset3”

```
{"op":"get","path":"DeviceObject/masterPresetBank/$bank/@items/3/control/@props/label”}\0x04
```

The device returns:

```
{"path":"DeviceObject/masterPresetBank/$bank/@items/3/control/@props/label","value":
"Preset3"}\0x04
```

Poll for the name of Screen Preset 12, which is labeled “ScreenPre12”

```
{"op":"get","path":"DeviceObject/presetBank/$bank/@items/12/control/@props/label”}\0x04
```

The device returns:

```
{"path":"DeviceObject/presetBank/$bank/@items/12/control/@props/label","value":
"ScreenPre12"}\0x04
```

3.5. Changing the source in a layer

The change of layer parameters is a bit more complex as the corresponding command set is indexed by preset **A/B**.

Preset A corresponds to the parameter set when the virtual TBAR is at the bottom (Down) and preset B corresponds to the parameter set when the virtual TBAR is at the top (Up).

It is therefore necessary to determine where the TBAR (Up or Down) is located before changing any layer parameter(s). For Screen 1, the following command must be sent to the device:

```
{"op":"get","path":"DeviceObject/$screenGroup/@items/S1/status/@props/transition"}\0x04
```

If the device returns:

```
{"path":"DeviceObject/$screenGroup/@items/S1/status/@props/transition","value":"AT_DOWN"}\0x04
```

This means that the TBAR of screen 1 is at the bottom. If you want to modify any layer parameters on the Program, you must therefore change DOWN parameters (or UP to change layer parameters on the Preview)

If the device returns:

```
{"path":" DeviceObject/$screenGroup/@items/S1/status/@props/transition","value":
"AT_UP"}\0x04
```

This means that the TBAR of screen 1 is at the top.

If you want to modify any layer parameters on the Program, you must therefore change UP parameters (or DOWN to change layer parameters on the Preview):

Transition status	**To work on PGM**	**To work on PRW**
AT DOWN	A	B
AT UP	B	A

Load Live 3 source on layer 2 of the screen 1 program (with TBAR at DOWN)

```
{"op":"replace","path":"DeviceObject/$screen/@items/S1/$preset/@items/A/
$layer/@items/2/source/@props/inputNum","value":"LIVE_3"}\0x04
```

The device will not return a string

Load Live 5 source on layer 1 of the screen 2 preview (with TBAR at DOWN)

```
{"op":"replace","path":"DeviceObject/$screen/@items/S2/$preset/@items/B/
$layer/@items/1/source/@props/inputNum","value":"LIVE_5"}\0x04
```

The device will not return a string

Load Live 8 source on layer 2 of the Auxiliary Screen 1 program (with TBAR at DOWN)

```
{"op":"replace","path":"DeviceObject/$screen/@items/A1/$preset/@items/A/
$layer/@items/2/source/@props/inputNum","value":"LIVE_8"}\0x04
```

The device will not return a string

Important: A global update is required to consider all the changes on the layers, mainly on Foreground Layer

```
{"op":"replace","path":"DeviceObject/$screenGroup/control/@props/xUpdate","value":true}\0x04
```

The device will not return a string

3.6. Changing screen background source

Important: Preview and Program are indexed to the TBAR, such that the current position of the TBAR will need to be known to correctly route to Preview or Program. See “Changing the source in a layer” for more details.

Load Background Set 2 to screen 1 program (with TBAR at DOWN)

```
{"op":"replace","path":"DeviceObject/$screen/@items/S1/$preset/@items/A/
$layer/@items/NATIVE/source/@props/inputNum","value":"NATIVE_2"}\0x04
```

The device will not return a string

Load Background Set 3 to screen 2 preview (with TBAR at DOWN)

```
{"op":"replace","path":"DeviceObject/$screen/@items/S2/$preset/@items/B/
$layer/@items/NATIVE/source/@props/inputNum","value":"NATIVE_3"}\0x04
```

The device will not return a string

3.7. Reading the last loaded preset

Important: Preview and Program are indexed to the TBAR, such that the current position of the TBAR will need to be known to correctly read the Preview or Program status. See "Changing the source in a layer" for more details.

Poll for the last recalled preset to program on screen 1 (last recalled was preset 3, with TBAR at DOWN)

```
{"op":"get","path":"DeviceObject/$screen/@items/S1/$preset/@items/A/presetId/status/
@props/id" }\0x04
```

The device returns:

```
{"path":"DeviceObject/$screen/@items/S1/$preset/@items/A/presetId/status/@props/id",
"value":3}\0x04
```

Poll for the last recalled preset to preview on Aux 1 (last recalled was preset 2, with TBAR at DOWN)

```
{"op":"get","path":"DeviceObject/$screen/@items/A1/$preset/@items/B/presetId/status/
@props/id"}\0x04
```

The device returns:

```
{"path":"DeviceObject/$screen/@items/A1/$preset/@items/B/presetId/status/@props/id","value":2}
```

3.8. Reading the last loaded master preset

Poll for the last recalled master preset to program (last recalled was master preset 3)

```
{"op":"get","path":"DeviceObject/masterPresetBank/status/lastUsed/$presetMode/@items/
PROGRAM/@props/memoryId"}\0x04
```

The device returns:

```
{"path":"DeviceObject/masterPresetBank/status/lastUsed/$presetMode/@items/
PROGRAM/@props/memoryId","value":3} \0x04
```

Poll for the last recalled master preset to preview (last recalled was master preset 2)

```
{"op":"get","path":"DeviceObject/masterPresetBank/status/lastUsed/$presetMode/@items/
PREVIEW/@props/memoryId"}\0x04
```

The device returns:

```
{"path":"DeviceObject/masterPresetBank/status/lastUsed/$presetMode/@items/
PREVIEW/@props/memoryId","value":2} \0x04
```

3.9. Changing routing audio source

The audio routing is done at channel level and is static. It means that it is required to configure each audio channel individually for each output or Dante Output.

The possible sources values are

Sources	Channel Names
No source	NONE
Live input channels	INPUT_xx_CHANNEL_x
Dante input channels	DANTE_xx_CHANNEL_x

The possible destination values are

Destinations	Channel Names
Screen/Aux Output	OUTPUT_x
Dante output group	DANTE_x (from 1 to 8)
Multiviewer	MVW_x

Route audio channel from Dante Group 4 Channel 2 to Output 8 Channel 1

```
{"op":"replace","path":"DeviceObject/audio/control/$tx/@items/OUTPUT_8/$channel/@items/1/control/
@props/source", "value": "DANTE_4_CHANNEL_2"}\0x04
```

The device will not return a string

4. Multiviewer commands

4.1. Recalling a Multiviewer Preset

Recall multiviewer preset 15 on Multiviewer 1

```
{"op":"replace","path":"DeviceObject/monitoringBank/control/load/$slot/@items/15/$output/
@items/1/@props/xRequest","value":true}\0x04
```

The device will not return a string

4.2. Changing multiviewer widget source

The possible sources of a widget are

Sources	Names
No Source	NONE
Program Screen ; Preview Screen	PROGRAM_Sx ; PREVIEW_Sx
Auxiliary Screen	A_x
Live Input	IN_x
Timer	TIMER_x
Still Image	STILL_x

The Widget value is indexed from 0 to 63.
So, if you want to target the widget 13, you shall indicate 12 as value.

Load live source 3 to multiviewer widget 13 on Multiviewer 1

```
{"op":"replace","path":"DeviceObject/$monitoring/@items/1/layout/$widget/@items/12/control/
@props/source”,”value”:”IN_3”}\0x04
```

The device will not return a string

5. Using thumbnails

5.1. Introduction

Thumbnails of live inputs, still images, outputs and multiviewer outputs are available. These thumbnails are regularly refreshed (except still images thumbnails which are refreshed only on change). Snapshot request rate must not be more than 1 per second.
Picture size is 256 pixels (width) by up to 256 pixels (height).
Black borders are automatically added, depending on aspect ratio. Picture type is PNG.

5.2. Live inputs thumbnails URL

http://<ipadress>/api/device/snapshots/inputs/1
up to
http://<ipadress>/api/device/snapshots/inputs/24

5.3. Still Images thumbnails URL (does not work for timers thumbnails)

http://<ipadress>/api/device/snapshots/images/1
up to
http://<ipadress>/api/device/snapshots/images/48
The number is depending on machine type and configuration

5.4. Outputs thumbnails URL

http://< ipadress>/api/device/snapshots/outputs/1
up to
http://<ipadress>/api/device/snapshots/outputs/20
The number is depending on machine type and configuration

5.5. Multiviewer thumbnails URL

http://< ipadress>/api/device/snapshots/multiviewers/1
up to
http://<ipadress>/api/device/snapshots/multiviewers/2

5.6. Timers thumbnails URL

http://<ipadress>/api/device/snapshots/timers/1
up to
http://<ipadress>/api/device/snapshots/timers/4

6. Waking the LivePremier™ device (over LAN)

6.1. Description

When the device has been shut down with the Wake on LAN feature enabled (‘sleep’ state), the only way to wake this device is to send a broadcast message over the network (‘magic packet’).

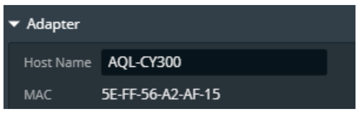
6.2. Wake on LAN and Magic Packet

Wake on LAN (or WOL) is the ability to send a signal over a local area network (LAN) to wake up a device. When the device is powered off with the Wake-On-LAN feature enabled, no operating system is running and there is no IP address assigned. Luckily, the MAC (Media Access Control) address being hardcoded in the network adaptor remains usable to identify a device in all states.

Using this MAC address, another system on the network may send to the sleeping device a wake-up signal. The wake up signal is a specific data frame, called ‘magic packet’, containing the MAC address of the remote network card. The magic packet is sent to all devices on the network (UDP broadcast) but is caught only by the device owning the matching MAC Address.

6.3. LivePremier™ device MAC address

You can get the LivePremier™ device MAC address using the Web RCS: Click the Information button located in the top right corner then select the Network option:


Adapter

Host Name	AQL-CY300
MAC	5E-FF-56-A2-AF-15

You can also retrieve the MAC address from the front panel menu: CONTROL -> Network.

6.4. Programming example

This .NET C# example sends a ‘magic packet’ for MAC address 00:25:90:3D:11:4A.

```
void SendWOLPacket(byte[] macAddress)
{
   // WOL 'magic' packet is sent over UDP.
   using (UdpClient client = new UdpClient())
  {
      // Send to: 255.255.255.0:9 over UDP (port number 9: Discard)
     client.Connect(IPAddress.Broadcast, 9);

     // Two parts to a 'magic' packet:
     // First is 0xFFFFFFFFFFFF,
     // Second is 16 * MACAddress.
     byte[] packet = new byte[17 * 6];

     // Set to: 0xFFFFFFFFFFFF.
     for (int i = 0; i < 6; i++)
       packet[i] = 0xFF;

     // Set to: 16 * MACAddress
     for (int i = 1; i <= 16; i++)
     {
         for (int j = 0; j < 6; j++)
             packet[i * 6 + j] = macAddress[j];
     }
     // Send WOL 'magic' packet.
     client.Send(packet, packet.Length);
   }
}

byte[] macaddress = new byte[] {0x00, 0x25, 0x90, 0x3D, 0x11, 0x4A};
WakeOnLan(macaddress);
```

July 2021
Document version 2.2.B

Connect with us on