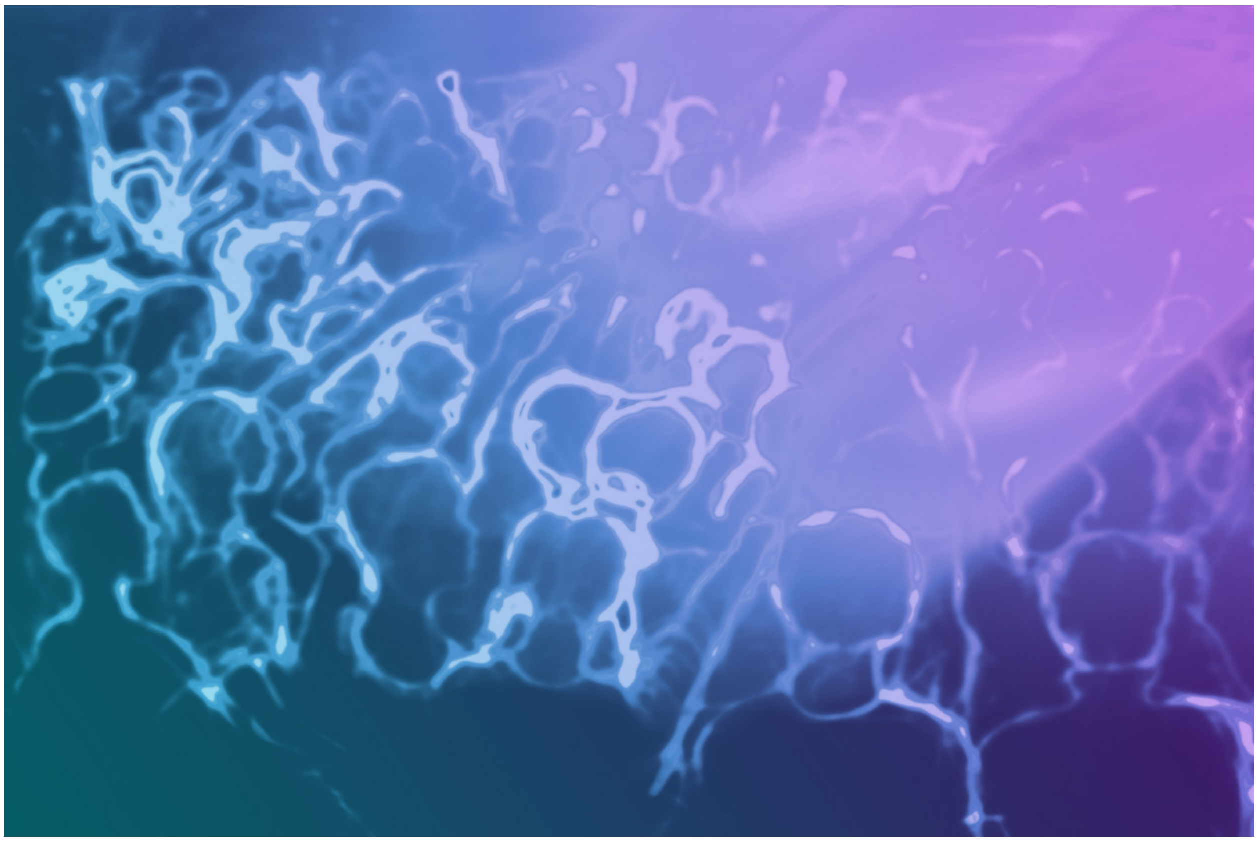

Medialon OpenCapXML

Protocol Specification

medialon
BY 7THSENSE

Document Revision

Date	Document edition	Software version	Revision details	Author/Editor
December 2017	1	1.0.0	Original release	David Rouchet and Kim Bui
June 2021	2	1.0.0	Rebranded, edited	Andie Davidson

M674-2

Contents

1 Generalities

1.1 Presentation

OpenCapXML is a communication protocol used to control Medialon products. It inherits the semantic of MON protocol but it based upon pure ASCII messages using XML format.

Coding format is UTF-8 for foreign characters.

OpenCapXML makes no assumption upon the underlying transport protocol although it is intended to be used over TCP or UDP sockets.

OpenCapXML is based upon 3 types of messages:

- Commands
- Replies/Errors
- Events

Commands are sent from client to server. Replies are sent from the server to client as answers to commands. They include session and sequence if they were given in the command.

Events can be sent at any time from the server (typically from when a change is detected).

Node value and node attributes name are not case sensitive (attribute `ProtocolVersion` is the same as `protocolversion`).

The products which currently support this protocol are:

- Manager 6.0.0 and above version (Server)
- Showmaster 2.0.0 and above version (Server)
- Scheduler 1.2.0 and above version (Client)

However, not all commands are supported by all products. This is mentioned individually for each command.

1.2 Naming Conventions

Some naming conventions will be used throughout this document, especially in XML representation formats:

```
<nnn> indicates an XML node where nnn is the name of the node.
"nnn" indicates a text or a value where nnn is the text or the value.
["nnn"] indicates an optional text or value.
_nnn indicates a generic text or a value where nnn is the name of this generic text or value. For example,
     _variablename indicates that the actual value is the name of the command, which will depend on the
     command itself. _variablename will be replaced by 'Status' or 'Error' variable name for instance.
// ... nnn ... // indicates a comment or a sum-up of the XML content.
_BOOL indicates a value that can be 0=false/off/closed or 1=true/on/opened
_INT indicates an integer value not limited unless indicated
_REAL indicates a float value not limited unless indicated. Real are noted like this:
     1.7000000000000002560000000000000000000000e+308 or 2.2800
_COLOR indicates a color value, formatted as follow: "RRR,GGG,BBB',
     where RRR is the Red value from '000' to '255' (decimal), GGG is the Green value from '000' to '255'
     (decimal) and BBB the Blue value from '000' to '255' (decimal).
_FONT indicates a font value, formatted as follow: "_fontname, _fontsize[,
_fontattr[|#fontattr]]". _fontattr can be: 'Bold', 'Italic', 'Underline', 'Strike'. Example: "Roman, 10,
     Bold|Italic".
```

2 Elementary Data Structures

2.1 Presentation

The section describes what could be called the ‘backbone’ of Medialon data, to represent data types such as String, Integer etc. Most of all other sections make a string reference to these data.

Medialon products use also what are called variables. A variable is merely an elementary data (String, Integer, etc.) which can be changed, modified over the time.

The elementary data structures are used for two different purposes:

- To represent variables of the system like Device Variables or User Variables.
- To represent parameters of Cue Commands.

2.2 Variable and Data Format

2.2.1 Common Structure

All of the types of variables (Integers, Strings, Reals, etc.) share a common set of attributes in the parent node, as shown below:

```
<_typeofdata
   Name="_STRING"
   ID="_INT"
   Tag="0"
   Description="_STRING"
   VarType="_INT"
   Persistent="_BOOL"
   Networked="_BOOL"
   VarRef="0"
/>
```

Where the `_typeofdata` shortcut stands for any of the following values:

- `IntegerData`
- `StringData`
- `TimeData`
- `EnumData`
- `DateData`
- `RealData`
- `PrivateData`

Node Attributes:

- `Name` is the name of the variable/data.
- `ID` unused attribute, maintained for backward compatibility, can be omitted.
- `Tag` unused attribute, maintained for backward compatibility, can be omitted.
- `Description` gives details on the variable (optional).
- `VarType` Coded type of the variable. Although this attribute is redundant with the node name, it is maintained for backward compatibility and each of XML reading.

  1=Integer

  2=String

  3=Time

  4=Enum

  5=Date

  6=Reserved

  7=Real
- `Persistent` indicates if the variable has a persistent value, can be omitted.
- `Networked` indicates if the variable is shared on the network. Should be always 1 with OpenCapXML.
- `VarRef` is used internally, the value must be kept unchanged if given or set to 0 otherwise.

2.2.2 Integer

```
<IntegerData
   Name="_STRING"
   ID="_INT"
   Tag="0"
   Description="_STRING"
   VarType="1"
   UseDataDescriptor="0"
   Persistent="_BOOL"
   Networked="_BOOL"
   VarRef="0"
   MinValue="_INT"
   MaxValue="_INT"
   Value="_INT"
/>
```

Integer Data specific attributes:

- `MinValue` indicates the minimum value accepted by the integer variable.
- `MinValue` indicates the minimum value accepted by the integer variable.
- `MaxValue` indicates the maximum value accepted by the integer variable.
- `Value` indicates the current value.

2.2.3 Real

```
<RealData
   Name="_STRING"
   ID="_INT"
   Tag="0"
   Description="_STRING"
   VarType="7"
   UseDataDescriptor="0"
   Persistent="_BOOL"
   Networked="_BOOL"
   VarRef="0"
   MinValue="_REAL"
   MaxValue="_REAL"
   Value="_REAL"
   Precision="_INT"
/>
```

Real Data specific attributes:

- `MinValue` indicates the minimum value accepted by the real variable.
- `MaxValue` indicates the maximum value accepted by the real variable.
- `Value` indicates the current value.

2.2.4 String

```
<StringData
   Name="_STRING"
   ID="_INT"
   Tag="0"
   Description="_STRING"
   VarType="2"
   UseDataDescriptor="0"
   Persistent="_BOOL"
   Networked="_BOOL"
   VarRef="0"
   MaxLength="_INT"
   String="_STRING"
/>
```

String Data specific attributes:

- `MaxLength` indicates the maximum length of the string. -1 means not limited.
- `String` indicates the current value.

2.2.5 Time

```
<TimeData
   Name="_STRING"
   ID="_INT"
   Tag="0"
   Description="_STRING"
   VarType="3"
   UseDataDescriptor="0"
   Persistent="_BOOL"
   Networked="_BOOL"
   VarRef="0"
   Time="_INT"
   TimeCodeType="_INT"
   ToTimeCodeType="_INT"
/>
```

Time Data specific attributes:

- `TimeCodeType` indicates the type of time.

  0=24fps

  1=25fps

  2=30fps

  3=30DP

  4=100fps (1/100s)

  5=1000fps (1/1000s)
- `ToTimeCodeType` indicates the type of time for conversion purpose. This value must be the same as `TimeCodeType`.
- `Time` indicates the current value in frames (or fraction of second).

2.2.6 Date

```
<DateData
   Name="_STRING"
   ID="_INT"
   Tag="0"
   Description="_STRING"
   VarType="5"
   UseDataDescriptor="0"
   Persistent="_BOOL"
   Networked="_BOOL"
   VarRef="0"
   Year="_INT"
   Month="_INT"
   Day="_INT"
   DayOfWeek="_INT"
/>
```

Date Data specific attributes:

- `Year` indicates the year value part of the date as a 4 digit integer (1900 -> xxxx).
- `Month` indicates the month value part of the date, allowed values from 1 to 12.
- `Day` indicates the day value part of the date, allowed values from 1 to 31.
- `DayOfWeek` indicates the corresponding weekday value, allowed values from 1 (Monday) to 7 (Sunday).

2.2.7 Enum

```
<EnumData
   Name="_STRING"
   ID="_INT"
   Tag="0"
   Description="_STRING"
   VarType="4"
   UseDataDescriptor=”0”
   Persistent="_BOOL"
   Networked="_BOOL"
   VarRef="0"
   Current="_INT">
   <EnumStrings>
      <EnumString Name="ready" Value="0" Checked="0" Caption="" />
      <EnumString Name="disable" Value="1" Checked="0" Caption="" />
      <EnumString Name="error" Value="2" Checked="0" Caption="" />
   </EnumStrings>
</EnumData>
```

Enum Data specific attributes:

- `Current` indicates the current selected index.

EnumString attributes:

- `Name` indicates the name of the enum.
- `Value` (optional) indicates the associated value of the enum.
- `Checked` (optional) indicates if the enum is selected. This is used when the enum structure is used as a multi-selection list.
- `Caption` (optional) when this attribute is set, it is displayed in place of Name attribute.

Note: Enums are by nature indexed, therefore the first enum has the index 0, the next one the index 1 and so on. If no values are given for `Value` attribute, the host may generate XML with automatic value using the index.

2.2.8 Private

```
<PrivateData
   Name="_STRING"
   ID="_INT"
   Class="_STRING">
</PrivateData >
```

Private Data specific attributes:

- `Class` Class name of the private data interface. This name is used by the host to instantiates object of this type when required.

2.3 Data Naming

Data/variables can be named. Names of variables are limited to 64 characters and must not contain reserved characters like the dot `.` or the space character ' '. If the variable is a system variable, its name is most of the time composed by the name of the owner + `.` + the local name of the variable: `Owner.LocalNameOfVar`. For instance the Host (device) system variables like: `Manager.CurrentTime`.

When the name of a variable has to be created or selected in an MXM, a Host interface `IObjectNameTools` can be used to ensure the name is valid.

2.4 ObjectParams

These element data structures can be packed inside an `ObjectParams` data structure which acts as a list of parameters. This list of parameters can then potentially be used for many other objects; the most obvious are the command parameters.

2.4.1 General Structure

```
<ObjectParams
   Version="_INT"
   ID="_INT"
   ParamsCount="_INT"
   Name="_STRING"
   Type="_INT"
   DefaultParamIndex="_INT">
   <ObjectParam
      VarType="_INT"
      ClassName="_STRING">
      <xxxData />
   </ObjectParam>
   ...
   <ObjectParam />
</ObjectParams>
```

The node `ObjectParams` embeds a series of `ObjectParam` nodes, one for each parameter. The node `ObjectParam` embeds an elementary data structure representing the parameter itself. The `xxxData` node is one of the elementary data structure described in Variable Data Format.

2.4.2 ObjectParams Attributes

- `Version` indicates the version of the ObjectParams structure.
- `ID` is the identifier associated with the list of parameter. It is depend of the owner of this list.
- `ParamCount` is the number of parameters in the list.
- `Name` is the name of parameter list. This attribute is optional.
- `Type` indicates the type of owner:
  - 0=Undefined
  - 1=Command
  - 2=Cue
  - 3=GraphicObject
  - 4=Track
  - 5=Task
  - 6=Variable
- `DefaultParamIndex` is the default parameter within the list. -1 indicates that there is no default parameter, which results in the first parameter to be considered as the default one.

2.4.3 ObjectParam Attributes

- `VarType` indicates the type of elementary data in the sub-node:
  - 1=Integer
  - 2=String
  - 3=Time
  - 4=Enum
  - 5=Date
  - 6=Reserved
  - 7=Real
- `ClassName` is the name of the class managing this data into the host. Generally this is the name of the elementary data proceeded by an `I` such as `IIntegerData` for integer.

2.4.4 Example of ObjectParams Data Structure

```xml
<ObjectParams
   Version="1001" ID="0" ParamsCount="5" Name="Medialon_I_O_23" Type="0" DefaultParamIndex="-1">
   <ObjectParam VarType="4" ClassName="IEnumData">
      <EnumData Name="Select By" ID="0" Tag="0" Description="" VarType="4" Persistent="0"
         Networked="0" VarRef="0" Current="0" EnumCount="2">
         <EnumStrings>
            <EnumString Name="Name" Value="0" />
            <EnumString Name="Index" Value="1" />
         </EnumStrings>
      </EnumData>
   </ObjectParam>
   <ObjectParam VarType="2" ClassName="IStringData">
      <StringData Name="I/O Name" ID="0" Tag="0" Description="" VarType="2" Persistent="0"
         Networked="0" VarRef="0" MaxLength="-1" String="OutputSwitch1">
      </StringData>
   </ObjectParam>
   <ObjectParam VarType="1" ClassName="IIntegerData">
      <IntegerData Name="I/O Index (0-X, 0=All)" ID="0" Tag="0" Description="" VarType="1"
         Persistent="0" Networked="0" VarRef="0" MinValue="-2147483647" MaxValue="2147483647"
         Value="0">
      </IntegerData>
   </ObjectParam>
   <ObjectParam VarType="4" ClassName="IEnumData">
      <EnumData Name="I/O Status" ID="0" Tag="0" Description="" VarType="4" Persistent="0"
         Networked="0" VarRef="0" Current="1" EnumCount="4">
      <EnumStrings>
         <EnumString Name="Off" Value="0" />
         <EnumString Name="On" Value="1" />
         <EnumString Name="Pulse" Value="2" />
         <EnumString Name="Toggle" Value="3" />
      </EnumStrings>
      </EnumData>
   </ObjectParam>
   <ObjectParam VarType="3" ClassName="ITimeData">
      <TimeData Name="Pulse Duration" ID="0" Tag="0" Description="" VarType="3" Persistent="0"
         Networked="0" VarRef="0" Time="0" TimeCodeType="4" ToTimeCodeType="4">
      </TimeData>
   </ObjectParam>
</ObjectParams>
```

3 Protocol Format

3.1 Message Generic Format

Each message is an xml structure starting with an `<opencap />` node:

```
<opencap type="" [session=""] [sequence=""]>
   // ... command specific data ... //
</opencap>
```

The inner of this node are other attributes are command specifics. Each session may have an associated ID and Sequence number:

- `session` parameter: can be omitted in TCP (clients are formally identified in TCP by connection). Mandatory in connectionless protocol (like HTTP, UDP).
- `sequence` parameter: sequence number under unreliable protocols (UDP) Each message has a sequence ID which is used to identify the message transaction. If the client uses the sequence ID, the server response to that command will contain the same sequence ID.
- `type` parameter: indicates the type of message. This type can be:
  - `command`
  - `reply`
  - `event`

3.2 Command Generic Format

A Command is sent from client to server. The message has the type attribute defined as `command`. The inner node(s) is (are) the command(s) name with its (their) specific sub nodes and attributes:

```
<opencap type="command" [session=""] [sequence=""]>
   <#commandname# // ... command specific attributes ... // >
      // ... command specific data ... //
   </#commandname#>
</opencap>
```

Or:

```
<opencap type="command" [session=""] [sequence=""]>
   <#commandname1# // ... command specific attributes ... // >
      // ... command specific data ... //
   </#commandname1#>
      ...
   <#commandnameX# // ... command specific attributes ... // >
      // ... command specific data ... //
   </#commandnameX#>
</opencap>
```

Although it is possible to send multiple commands within a single opencap frame, it is recommended to limit this usage. Indeed, this protocol is based on a command/answer couple mechanism which is somewhere broken when multiple commands are used.

3.3 Reply Generic Format

Replies are sent from the server to client as answers to commands. They include session and sequence if they were given in the command. The message has the type attribute defined as `reply`. The inner node is the command name with its specific sub nodes and attributes to which the reply applies. Reply messages are sent when the related command has been processed by the server:

```
<opencap type="reply" resultcode="" [resultstring=""] [session=""] [sequence=""] >
   <#commandname# // ... reply specific attributes ... // >
      // ... reply specific data ... //
   </#commandname#>
</opencap>
```

See section 3.5 below for results code definitions.

3.4 Event Generic Format

Events can be sent at any time from the server (typically from when a change is detected). The message has the type attribute defined as `event`. The inner node is the Event name with its specific sub nodes and attributes:

```
<opencap type="event" [session=""] [sequence=""]>
   <#eventname# // ... event specific attributes ... // >
      // ... event specific data ... //
   </#eventname#>
</opencap>
```

3.5 Error Codes Definition

`resultcode` attribute in a reply message indicates either a success or an error code. The meanings of these error codes are listed below:

Code		Meaning
0x0000	(0)	Operation succeed
0x0001	(1)	Invalid Argument
0x0002	(2)	The given buffer is too small
0x0003	(3)	Missing data
0x0004	(4)	The given ID is invalid or not defined
0x0010	(16)	The requested operation is not supported
0x0011	(17)	The requested operation is not known
0x0012	(18)	The requested operation has timed out

Code		Meaning
0x0013	(19)	The requested operation has failed
0x0014	(20)	The requested operation has been aborted
0x0020	(32)	The object is in a wrong state and cannot perform the requested operation
0x0021	(33)	Object access is denied
0x0022	(34)	The requested operation cannot be performed, limit reached
0x0023	(35)	The connection is not established
0x0024	(36)	The connection attempt has failed
0x0025	(37)	The connection is not yet identified (need identification)
0x0026	(38)	A network error occurred
0x0027	(39)	The connection is not yet registered
0x0028	(40)	The connection is already registered
0x0029	(41)	The connection registration failed
0x0030	(48)	The requested object doesn't exist
0x0031	(49)	The requested object is not shared
0x0032	(50)	The requested property is not part of this object
0x0033	(51)	The requested property cannot be set/get on this object
0x0034	(52)	The connection has been redirected
0x0035	(53)	The connection redirection has not been defined
0x0036	(54)	The requested protocol version is not supported

4 Commands

4.1 Connection Management

4.1.1 Connect

Supported in products: Manager/Showmaster/Scheduler.

Supported by protocol: version 1.0, version 1.1.

The connect command must be the first command sent after successful underlying protocol connection. The reply to this command may contain the session ID which will be used throughout the conversation. Session ID can be ignored if the client uses a connection-oriented transport protocol such as TCP.

Command:

```
<opencap type="command" [session=""] [sequence=""]>
   <connect
      netid=""
      timeout="#INT#"
      user=""
      password=""
      sharedgroup=""
      acceptevents="#BOOL#"
      protocolversion=""
   />
</opencap>
```

- `netid` is a client type identifier and must set to `mopencapapi` for OpenCapXML protocol.
- `timeout` specifies the expected maximum response time by the client expressed in milliseconds.
- `user` This field must be filled if the server has a user and password defined.
- `password` This field must be filled if the server has a user and password defined. It must be encoded in base64.
- `sharedgroup` specifies the shared group to which the client wants to subscribe.
- `acceptevents` indicates if the client accept events from the server. A value of 1 means TRUE, 0 means FALSE.
- `protocolversion` indicates the protocol version required by the client.
- `usesession` indicates if the session ID and message ID are managed (from protocol version 1.1).

Answer:

```
<opencap type="reply" resultcode="0" resultstring="Success" session="" sequence="" >
   <connect protocolversion="" netid="" netidext="" sharedgroup="" acceptevents="#BOOL#"
         usesession="" timeout="#INT#" >
      <environmentdata datetype="0" />
   </connect>
</opencap>
```

All the returned fields contain values which have been sent by the client except `protocolversion`, which indicates the protocol version supported by the server. If the version returned by the server differs from the version required by the client, the client is free to either disconnect or to adapt itself to the version of the server.

Also note that result code 0x0034 (52) is returned by the server if a redirection is needed. In that case, the client must get the redirected host net parameters via the `GetNetRedirectionParameters` described below, and connect to the redirected server. If the connection is already established and the server needs the client to be redirected, the `SetNetRedirectionParameters` event described below will be sent by the server.

4.1.2 Register Group

Supported in products: Manager/Showmaster/Scheduler.

Supported by protocol: version 1.0, version 1.1.

This command is used to subscribe to a shared group. Generally, the shared group is indicated in the `connect` command, but in some cases, it may be necessary to register to another group during the session.

Note that a client will receive `propertychange` events with the initial state of the variables of the specified group when it sends a `registergroup` command after it has sent a `connect` command with the `AcceptEvents` set to `1`. This allows receiving the initial state of the variables without requiring polling the variable values with `getobject`.

Command:

```
<opencap type="command" [session=""] [sequence=""]>
   <registergroup sharedgroup="" acceptevents="#BOOL#" />
</opencap>
```

- `sharedgroup` specifies the shared group the client wish to subscribe.
- `acceptevents` indicates if the client accept events from the server. A value of 1 means TRUE, 0 means FALSE.

Answer:

```
<opencap type="reply" resultcode="0">
   <registergroup sharedgroup="" />
</opencap>
```

4.1.3 Disconnect

Supported in products: Manager/Showmaster/Scheduler.

Supported by protocol: version 1.0, version 1.1.

Disconnect is recommended even in TCP in order to make sure that the client is properly done. However, in case of TCP disconnection, the session is considered terminated by the server.

Command:

```
<opencap type="command" [session=""] [sequence=""]>
   <disconnect />
</opencap>
```

Answer:

```
<opencap type="reply" resultcode="0">
   <disconnect />
</opencap>
```

4.1.4 Get Net Redirection Parameters

Supported in products: Showmaster.

Supported by protocol: version 1.0, version 1.1.

This command is used to get the redirection parameters of the server.

Example: When an OpenCapXML server must redirect incoming OpenCapXML requests to another server, it returns 0x0034 (52) as the result code of the Connect command. The connecting command client gets the new host redirection parameters with the `GetNetRedirectionParameters` command.

The connecting client then connect the redirected server using these parameters. Afterwards, whether this is a normal or abnormal disconnection, the client should always reconnect to the original address:port.

Command:

```
<opencap type="command" [session=""] [sequence=""]>
   <getredirectionparameters />
</opencap>
```

Answer:

```
<opencap type="reply" resultcode="0">
   <getredirectionparameters
      address="192.168.0.1"
      mask="255.255.255.0"
      port="9255"
   />
</opencap>
```

Note: redirection mechanism occurs when a Showmaster Editor connects a Showmaster. Any client to that Showmaster is redirected towards the Showmaster Editor until the Showmaster Editor disconnects from the Showmaster. In this case, the client gets disconnected from the Showmaster Editor and must reconnect the Showmaster (the original connection).

4.1.5 Set Net Redirection Parameters (Event)

Supported in products: Showmaster.

Supported by protocol: version 1.0, version 1.1.

This event is emitted when the server needs to redirect its current connection.

Event:

```
<opencap type="event" [session=""] [sequence=""]>
   <setredirectionparameters
      address="192.168.0.1"
      mask="255.255.255.0"
      port="9255"
   />
</opencap>
```

Upon reception of this event, the client should disconnect the server and re-connect using the provided parameters. Afterwards, whether this is a normal or abnormal disconnection, the client should always reconnect to the original address:port; if another redirection is needed, it will be notified either with this event on in the connect answer (with result code 0x0034 (52)).

Note: redirection mechanism occurs when a Showmaster Editor connects a Showmaster. Any client to that Showmaster is redirected towards the Showmaster Editor until the Showmaster Editor disconnects from the Showmaster. In this case, the client gets disconnected from the Showmaster Editor and must reconnect the Showmaster (the original connection).

4.2 DOM Discovery

4.2.1 Get Object List

Supported in products: Manager/Showmaster/Scheduler.

Supported by protocol: version 1.0, version 1.1.

This command retrieves the list of objects shared within the current shared group and within the `parentobject` scope.

Command:

```
<opencap type="command" [session=""] [sequence=""]>
   <getobjectlist parentobject="" objecttype=""/>
</opencap>
```

`parentobject` specifies the name of an object which holds the required list of objects. At minimum, this should be the shared group name.

- `objecttype` is an integer value which represent a particular object type following this list:

Type	Object	Returned Type	Returned Value
0	All Objects	* (one of above values)	* (one of above values)
1	Devices	Device	Device / ManagerOnNetwork
2	MXM Devices	–	–
3	Tasks	Task	StepBasedTask / TimeBasedTask
4	Timeline Tasks	Task	TimeBasedTask
5	Stepbased Tasks	Task	StepBasedTask
6	Variables	Variable	DateData / EnumData
7	User Variables	Variable	IntegerData / RealData / StringData / TimeData / DateData / EnumData
8	User Screens	UserScreen	UserScreen
9	Buttons	–	–
10	Sliders	–	–
11	Digital Sliders	–	–
12	Displays	–	–
13	Leds	–	–
14	Gauges	–	–
15	Squares	–	–
16	Circles	–	–
17	Lines	–	–
18	Images	–	–
19	Pages	–	–
20	Viewers	–	–
21	Texts	–	–
22	Containers	–	–
23	Lists	–	–
24	Edits	–	–
25	Device Graphic Windows	–	–

Type	Object	Returned Type	Returned Value
26	All Graphic Objects	–	–
27	User Groups	–	–
28	Cues	–	–
29	Commands	–	–
30	Resources	–	–

Answer:

```
<opencap type="reply" resultcode="#INT#" [session=""] [sequence=""] >
   <getobjectlist parentobject="" objecttype="" />
      <Objects>
         <Object Type="" Value="" Name="" Description="" />
         <Object Type="" Value="" Name="" Description="" />
      </Object>
   </getobjectlist>
</opencap>
```

- `type` is one of the values present in the above table (Returned Type column).
- `value` gives a detailed type when applicable (see Returned Value in the above table).
- `<object>` nodes may have some other attributes depending on the type of object. For instance, `Task` with value= `TimeBasedTask` has an extra `Duration` attribute.

4.2.2 Get Object

Supported in products: Manager/Showmaster/Scheduler.

Supported by protocol: version 1.0, version 1.1.

This command retrieves a complete or partial object description. The quantity of information in the description depends on the object itself and in the depth attribute.

Command:

```
<opencap type="command" [session=""] [sequence=""] >
   <getobject object="" depth="#INT#" />
</opencap>
```

- `object` indicates the name of the object.
- `depth` indicates the number of nodes in the hierarchy requested in the answer.

Answer:

```
<opencap type="reply" resultcode="#INT#" [session=""] [sequence=""]>
   <getobject object="" depth="#INT#">
      <#objecttypename# name="" id="" description="" /... attributes related to the </#objectname#>
   </getobject>
</opencap>
```

The format of the node describing the object depends of the object itself. See Data Format for details of those formats.

4.3 Object Control

4.3.1 Set Property

Supported in products: Manager/Showmaster/Scheduler.

Supported by protocol: version 1.0, version 1.1.

This command changes the property value of the given object.

Command:

```
<opencap type="command" [session=""] [sequence=""] >
   <setproperty object="" property="" value="" />
</opencap>
```

- `object` indicates the name of the object.
- `property` indicates the name of the property.
- `value` indicates the value of the property.

Answer:

```
<opencap type="reply" resultcode="#INT#" [session=""] [sequence=""] >
   <setproperty object="" property="" />
</opencap>
```

- `object` indicates the name of the object.
- `property` indicates the name of the property.

4.3.2 Get Property

Supported in products: Manager/Showmaster/Scheduler.

Supported by protocol: version 1.0, version 1.1.

This command retrieves the property value of the given object.

Command:

```
<opencap type="command" [session=""] [sequence=""] >
   <getproperty object="" property="" />
</opencap>
```

- `object` indicates the name of the object.
- `property` indicates the name of the property.

Answer:

```
<opencap type="reply" resultcode="#INT#" [session=""] [sequence=""]>
   <getproperty object="" property="" value=""/>
</opencap>
```

- `object` indicates the name of the object.
- `property` indicates the name of the property.
- `value` indicates the value of the property.

4.3.3 Perform

Supported in products: Manager/Showmaster/Scheduler.

Supported by protocol: version 1.0, version 1.1.

This command performs a specific action on the given object. The nature of the action depends on the object. The target of the action is an internal core object (such a Task or a UserScreen).

Command:

```
<opencap type="command" [session=""] [sequence=""] >
   <perform object="" action="" >
      <parameters>
         <parameter name="" value="" [type=""] />
            ...
         <parameter name="" value="" [type=""] />
      </parameters>
   </perform>
</opencap>
```

- `object` indicates the name of the object.
- `action` indicates the name of the action to perform.

`<parameter>` nodes are a list of parameter for the action.

- `name` indicates the name of the parameter.
- `value` indicates the value of the parameter.
- `type` optionally indicates the type of the parameter (integer|float|string|time|date).

Answer:

```
<opencap type="reply" resultcode="#INT#" [session=""] [sequence=""]>
   <perform object="" action=""/>
</opencap>
```

- `object` indicates the name of the object.
- `action` indicates the name of the action.

Detailed Action/Object Type

Core objects support a predefined set of commands detailed below.

4.3.3.1 UserScreen Perform Commands

- openuserscreen
  - mode: enum [0=Default, 1=Absolute]
  - left: integer (left position)
  - top: integer (top position)
- closeuserscreen
- gotopage
  - pagenumber: integer

4.3.3.2 Task Perform Commands

- starttask
- pausetask
- stoptask
  - mode: enum [0=Specified Task, 1=All inclusive, 2=All exclusive]

4.3.3.3 StepBasedTask Perform Commands

- gotolabel
  - label: string (label name)
- gotoline
  - line: integer (line number)

4.3.3.4 TimeBasedTask Perform Commands

- gototime
  - time: integer (frames)

Example: performing a "GoToTime 00:00:10/00" command on the task "Show1":

```
<opencap type="command">
   <perform object="Show1" action="gototime">
      <parameters>
         <parameter name="Timecode" value="1000" />
      </parameters>
   </perform>
</opencap>
```

4.4 Change Management

4.4.1 Property Change Event

Supported in products: Manager/Showmaster/Scheduler.

Supported by protocol: version 1.0, version 1.1.

Change events are fired each time a property change is detected on the server. This event is only sent if client connection parameter `acceptevents` has been set to 1 at connection time.

Event:

```
<opencap type="event" [session=""] [sequence=""] >
   <propertychanges>
      <property object="" property="" value="" />
         ...
      <property object="" property="" value="" />
   <propertychanges>
</opencap>
```

`<property>` nodes are a list of property which have changed.

- `object` indicates the name of the object.
- `property` indicates the name of the property within the object.
- `value` indicates the value of the property.

`<property>` node can have a `<subprop>` sub node with attributes which depend on the property itself.

4.4.2 Notification of Variable Change

When a variable change has to be notified, sub nodes have the following meaning.

4.4.2.1 Integer Variable

Property is `value`

```
<subprop type="IntegerData" />
```

4.4.2.2 Real Variable

Property is `value`

```
<subprop type="RealData" />
```

4.4.2.3 String Variable

Property is `string`

```
<subprop type="StringData" />
```

4.4.2.4 Time Variable

Property is `time`

The value attribute gives the time in text format: `hh:mm:ss/ff`

```
<subprop type="TimeData" tctype="#INT#" frame="#INT#"/>
```

- `tctype` indicates the framerate with the following values: 0=24fps, 1=25fps, 2=30fps, 3=30DP, 4=100fps (1/100s), 5=1000fps (1/1000s).
- `frame` indicates the time value in frames (units).

4.4.2.5 Enum Variable

Property is `current`

```
<subprop type="EnumData" >
   <EnumStrings>
      <EnumString Name=”” Value=”#INT#” />
         ...
      <EnumString Name=”” Value=”#INT#” />
   </EnumStrings>
</subprop>
```

`<EnumStrings>` node gives the list of enumeration string.

4.4.2.6 Date Variable

Property is `date`

The value attribute gives the date in text format: `dd/mm /yyyy`:

```
<subprop type="DateData" year="#INT#" month="#INT#" day="#INT#" weekday="#INT#" />
```

5 Examples

5.1 Simple Message Examples

5.1.1 Set Variable

Command:

```xml
<opencap type="command" >
   <setproperty object="#variablename#" property="value" value="#variablevalue#" />
</opencap>
```

Answer:

```xml
<opencap type="reply" resultcode="0" >
   <setproperty object="#variablename#" property="value" />
</opencap>
```

5.1.2 Get Variable

Command:

```xml
<opencap type="command" >
   <getproperty object="#variablename#" property="value" />
</opencap>
```

Answer:

```xml
<opencap type="reply" resultcode="0" >
   <getproperty object="#variablename#" property="value" value="#variablevalue#" />
</opencap>
```

5.1.3 Open User Screen

Command:

```xml
<opencap type="command" >
   <perform object="#userscreenname#" action="openuserscreen" >
      <parameters>
         <parameter name="left" value="" />
         <parameter name="top" value="" />
      </parameters>
   </perform>
</opencap>
```

Examples

Answer:

```
<opencap type="reply" resultcode="0" >
   <perform object="#userscreenname#" action="openuserscreen" />
</opencap>
```

5.1.4 Close User Screen

Command:

```
<opencap type="command" >
   <perform object="#userscreenname#" action="closeuserscreen" >
   </perform>
</opencap>
```

Answer:

```
<opencap type="reply" resultcode="0" >
   <perform object="#userscreenname#" action="closeuserscreen" />
</opencap>
```

5.1.5 Goto To Page

Command:

```
<opencap type="command" >
   <perform object="#userscreenname#" action="gotopage" >
      <parameters>
         <parameter name="pagenumber" value="#pagenumber#" />
      </parameters>
   </perform>
</opencap>
```

Answer:

```
<opencap type="reply" resultcode="0" >
   <perform object="#userscreenname#" action="gotopage" />
</opencap>
```

5.1.6 Start Task

Command:

```
<opencap type="command" >
   <perform object="#taskname#" action="starttask" >
   </perform>
</opencap>
```

Examples

Answer:

```
<opencap type="reply" resultcode="0" >
   <perform object="#taskname#" action="starttask" />
</opencap>
```

5.1.7 Pause Task

Command:

```
<opencap type="command" >
   <perform object="#taskname#" action="pausetask" >
   </perform>
</opencap>
```

Answer:

```
<opencap type="reply" resultcode="0" >
   <perform object="#taskname#" action="pausetask" />
</opencap>
```

5.1.8 Stop Task

Command:

```
<opencap type="command" >
   <perform object="#taskname#" action="stoptask" >
      <parameters>
         <parameter name="mode" value="0" />
      </parameters>
   </perform>
</opencap>
```

Answer:

```
<opencap type="reply" resultcode="0" >
   <perform object="#taskname#" action="stoptask" />
</opencap>
```

5.1.9 Goto Label

Command:

```
<opencap type="command" >
   <perform object="#taskname#" action="gotolabel" >
      <parameters>
         <parameter name="label" value="#labelname#" />
      </parameters>
   </perform>
</opencap>
```

Examples

Answer:

```
<opencap type="reply" resultcode="0" >
   <perform object="#taskname#" action="gotolabel" />
</opencap>
```

5.1.10 Goto Line

Command:

```
<opencap type="command" >
   <perform object="#taskname#" action="gotoline" >
      <parameters>
         <parameter name="line" value="#linenumber#" />
      </parameters>
   </perform>
</opencap>
```

Answer:

```
<opencap type="reply" resultcode="0" >
   <perform object="#taskname#" action="gotoline" />
</opencap>
```

5.1.11 Goto Time

Command:

```
<opencap type="command" >
   <perform object="#taskname#" action="gototime" >
      <parameters>
         <parameter name="time" value="#framenumber#" />
      </parameters>
   </perform>
</opencap>
```

Answer:

```
<opencap type="reply" resultcode="0" >
   <perform object="#taskname#" action="gototime" />
</opencap>
```

5.2 Medialon Scheduler Target Example

This example is an application to Medialon Scheduler Target control. It explains how a Medialon Scheduler uses OpenCapXML to address targets other than Medialon Manager instances.

5.2.1 Pre-requisite

- Target has defined a shared-group named 'tgt1'.
- Target shares 2 tasks 'T1' & 'T2' and one enum variable 'MyCondition' The login user name has been defined as 'demo'.
- No password.
- Connection is over a TCP/IP connection.

5.2.2 Commands/Answers Sequence

Scheduler TCP/IP Connection Scheduler SENDS:

```
<opencap type="command" >
   <login
      netid="scheduler"
      timeout="0"
      user="demo"
      password=""
      sharedgroup="tgt1"
      acceptevents="1"
      protocolversion="1.0"
   />
</opencap>
```

Target REPLIES:

```
<opencap type="reply" resultcode="0" >
   <login />
</opencap>
```

Scheduler SENDS:

```
<opencap type="command">
   <registergroup sharedgroup="tgt1" />
</opencap>
```

Target REPLIES:

```
<opencap type="reply" resultcode="0" >
   <registergroup />
</opencap>
```

Examples

Scheduler SENDS:

```
<opencap type="command" >
   <getobjectlist parentobject="tgt1" objecttype="usergroups" />
</opencap>
```

Target REPLIES:

```
<opencap type="reply" resultcode="0" >
   <getobjectlist parentobject="" objecttype="" />
      <Objects>
         <Object Type="Task" Name="T1" Value="StepBasedTask" Description="" />
         <Object Type="Task" Name="T2" Value="TimeBasedTask" Description="" Duration="00:00:02/23"/>
         <Object Type="Variable" Value="EnumData" Name="MyCondition" Description="" />
      </Objects>
   </getobjectlist>
</opencap>
```

Scheduler SENDS:

```
<opencap type="command" >
   <perform object="T1" action="starttask" />
</opencap>
```

Target REPLIES:

```
<opencap type="reply" resultcode="0" >
   <perform object="T1" action="starttask" />
</opencap>
```

Because the Scheduler has specified the property `acceptevents="1"`, the target is allowed to notify object properties change. In this example, any change in the `MyCondition` variable is reported like this:

Target EVENT:

```
<opencap type="event" >
   <propertychanges>
      <property object="MyCondition" property="value" value="0" >
         <subprop type="EnumData" >
            <EnumStrings>
               <EnumString Name=”ready” Value=”0” />
               <EnumString Name=”disabled” Value=”1” />
               <EnumString Name=”error” Value=”2” />
            </EnumStrings>
         </subprop>
      </property>
   </propertychanges>
</opencap>
```

6 Revisions

Version	
0.1	Initial version protocol version 1.0
0.2	Replaced: Generic <msg ... /msg> by <opencap ... /opencap> Added: netid parameter for connect command Renamed: Execute command to Perform to match current MON protocol def.
0.3	Removed: Get Changed Properties Added: Name parameter in node Renamed: Deprecated Messages to Message Samples
0.4	Updated: Protocol generic format Changed: Reply format (which replace error message)
0.5	Removed: Unused notes Changed: 'connect' command to login and registergroup commands Changed: 'disconnect' to 'unlogin' command Added: Example case for Medialon Scheduler Target
0.6	Added: Complementary for property change notification
0.7	Corrected: Wrong <object> node properties in getobjectlist command reply Added: Example of object change notification
0.8	Changed: format of returned event. Protocol version 1.1 Added: Detailed description and examples
0.9	Changed: variable property attribute names
0.10	Changed: Some graphic object name/property values
0.11	Added: Error code definition
0.16	Modified: Perform command description
0.17	Modified: <Params>/<Param> nodes are renamed to <ObjectParams>/<ObjectParam>
0.18	Removed: Unused Direction attribute from Added: Protocol/Product compatibility
0.19	Fixed: Typos in XML examples Fixed: Perform/PerformEx specifications are incorrect
0.20	Added: Notes about Register Group command Removed: getobjectproperties command description (not implemented)
0.21	Removed: Description of commands not implemented yet
0.23	Added: Protocol version and supported products per command
0.24	Added: Accept event for Register Group
0.25	Fixed: parentobject must be specified in 'Get Object List' Added: GetNetRedirectionParameters command and SetNetRedirectionParameters event
1.0.0	Updated: Final version 1.0.0

medialon
BY 7THSENSE

E: info@7thsense.one
W: 7thsense.one

7thSense Design Ltd

2 The Courtyard Shoreham Road
Upper Beeding
Steyning
West Sussex
BN44 3TN
UK

T: +44 (0) 1903 812299

7thSense LLC, Michigan

332 E Lincoln Ave
Suite 100
Royal Oak, MI 48067
USA

T: +1 248 599 2717

7thSense LLC, Orlando

4207 Vineland Rd
Suite M1
Orlando, FL 32811
USA

T: +1 407 505 5200