Barco Device Interface

Barco Device Interface

Table of Contents

Glossary 1
Introduction ii
1. Service Overview 3
    Version 3
    Device Overview 3
2. Agent Message Overview 4
    Generic Agent Message API 4
    Generic Agent Message Data For Request Type Agent Messages 5
    Generic Agent Message Data For Response Type Agent Messages 7
    Generic Agent Message Data For Event Type Agent Messages 7
3. Generic Agent API Overview 9
    Startup Operations 9
        XMPP Login 9
        DeviceLogin 9

List of Examples

2.1. Message 4
2.2. Request 5
2.3. Request(RPC) 5
2.4. Request(without Type) 6
2.5. Response 7
2.6. Response 7
2.7. Event 8
3.1. DevciceLoginRequest extended from AgentRequest 10
3.2. Service Login Response extended from AgentResponse 11
3.3. Service Login Response error extended from AgentResponse 11

Glossary

Device Interface	This represents the external interface to a device.
Device Interface API	The API accepted by the device on its external interface
Service	The device provides the following services : DeviceAdminAgent, Media, Layout, MediaStoreAgent
Handler	The code that handles a specific API message for a given service
Agent	The Handler invokes methods on the Agent for the accepted by the device on its external interface
Engine	The Agent invokes methods on the Engine that implement the services on the device

Introduction

This document contains the description of the generic device interface API.

Chapter 1. Service Overview

A service is a software/firmware/hardware capability of a device that is made available to a user The scope of this document is limited to 4 types of services within namely device administration, media, layout and media storage services DeviceAdmin - Device management including configuration, monitoring, diagnostics Media - Sending and receiving media streams Layout - Rendering media in specific layouts MediaStore - Managing the storage of recordings and file systems

Version

Document Version = 0.0.1
Service Version = x.x.x
Software Version = 4.0.x.x

Device Overview

The Device is capable of providing services
These services are available thru an API
The API is handled by code modules called "handlers" which are organized on the same lines as the service
The handlers invoke methods on "agents" which are organized on the same lines as the service
The agents have direct access to the software/firmware services which are called Engines
The message used to access a service is called an AgentMessage

Chapter 2. Agent Message Overview

The messages have been described here are as extensions of standard "XMPP Message" packets. However the AgentMessage itself can be sent over HTTP or any other messaging bus

All agent message payloads will use the namespace "com.barco.agentmessage"

Generic Agent Message API

The messaging between the agentts is realized as a set of XML messages over an XMPP message bus. These messages constitute the Barco Device API. Handlers will need to implement the relevant sub-set of this API based on their intended MediaRoom Role.

Example 2.1. Message

```
<message id="12rLk-221" to="tx1@localhost/tx1"
   from="managementserver@localhost/managementserver">
  <x xmlns='com.barco.agentmessage'>
    <AgentMessage from='xpi@localhost/xpi' to='xpi@localhost/xpi'
          type='Request/Response/Event' logLevel='5' version=''>
      <!-- Message Data goes here -->
   </AgentMessage>
  <x>
</message>
```

Generic Agent Message Data For Request Type Agent Messages

Example 2.2. Request

```
<Request>
  <Header serviceName='Media' type='PUT' requestName='Content'
       userJID='user@localhost/pc' requestNID='1234'>
     <ClientData>cd123</ClientData>
     <ClientCallback></ClientCallback>
  </Header>
  <Data>
     <!-- Request data goes here -->
     <PUTContentRequestData>
             <!-- PUTContentRequestData payload goes here -->
     </PUTContentRequestData>
  </Data>
</Request>
```

Example 2.3. Request(RPC)

```
<Request>
  <Header serviceName='DeviceAdmin' type='RPC' requestName='Upgrade'
        userJID='user@localhost/pc' requestNID='1234'>
     <ClientData>cd123</ClientData>
     <ClientCallback></ClientCallback>
  </Header>
  <Data>
     <!-- Request data goes here -->
     <RPCUpgradeRequestData>
             <!-- RPCUpgradeRequestData payload goes here -->
     </RPCUpgradeRequestData>
  </Data>
</Request>
```

Example 2.4. Request(without Type)

```
<Request>
  <Header serviceName='Layout' type='' requestName='SetupRequest'
        userJID='user@localhost/pc' requestNID='1234'>
     <ClientData>cd123</ClientData>
     <ClientCallback></ClientCallback>
  </Header>
  <Data>
     <!-- Request data goes here -->
     <SetupRequestData>
            <!-- SetupRequestData payload goes here -->
     </SetupRequestData>
  </Data>
</Request>
```

serviceName - The service being requested i.e. DeviceAdmin, Media, Layout, MediaStore
type - The type of request to be handled, GET/PUT/DELETE/RPC.
requestName - The request within the service points to the resource or remote procedure
XYZRequestData - The type is prefixed with the requestName to generate the name of the requestdata element
userJID - The original user making the request(could be different than the fromJID
requestNID - unique ID for the request
state - Status of the request, 0 is Error and 200 is OK
ClientData - data provided by the requestor which needs to be returned in the response as is
CallbackData - data provided by the requestor which needs to be returned in the response as is

Generic Agent Message Data For Response Type Agent Messages

Example 2.5. Response

```
<Response>
  <Header serviceName='Layout' type='' requestName='SetupRequest'
       userJID='user@localhost/pc' requestNID='1234' state='200'>
<ClientData>cd123</ClientData>
<ClientCallback></ClientCallback>
  </Header>
  <Data>
     <!-- Request data goes here -->
     <SetupResponseData>
             <!-- SetupResponseData payload goes here -->
     </SetupResponseData>
  </Data>
</Response>
```

The serviceName, requestName, userJID, requestNID, ClientData are same as those passed in the request
The ResponseData may be replaced by an error element in case there is an error

Example 2.6. Response

```
<Response>
  <Header serviceName='Layout' type='' requestName='SetupRequest'
       userJID='user@localhost/pc' requestNID='1234' state='0'>
<ClientData>cd123</ClientData>
<ClientCallback></ClientCallback>
  </Header>
  <Data>
   <Error code="201">
    <Description>Destination Busy</Description>
   </Error>
  </Data>
</Request>
```

Generic Agent Message Data For Event Type Agent Messages

An notification can be an info event or an error event. This is indicated by the state element in the payload

Example 2.7. Event

```
<Event>
  <Header serviceName='Media' eventName='StreamStatusUpdate'
     eventAgentJID='srcrelay1@localhost/srcrelay1'
     eventWallclock='1288358866404' eventLevel='INFO'/>
  <Data>
   <StreamStatusUpdateData>
        <!-- StreamStatusUpdate data payload goes here -->
   </StreamStatusUpdateData>
  </Data>
</Event>
```

serviceName - The service within which this event is being generated i.e. DeviceAdmin, Media, Layout, MediaStore
eventName - The request within the service
eventAgentJID - The original user making the request(could be different than the fromJID)
eventWallclock - wallclock at which this event was generated
eventLevel - 0 is Error and 200 is OK

Chapter 3. Generic Agent API Overview

The following set of operations will be typically performed by the devices hosting the agents outside of the agent API

Startup Operations

XMPP Login

First the Client does a standard XMPP Login. The credentials needed for this need to be provided to the Client by the Administrator by some out-of-band means.

- Username@DomainName.com (Who)
- Password
- XMPPResourceID (From Where)

DeviceLogin

After the base XMPP login the device has to discover the contactJID for the Central Server The Device then does a "DeviceLogin" which registers the Device with the Server.

- The registration process includes publishing its userJID as the contactJID for the XMPPResource
- Device version number and licensing is also checked at this time
- The Device version number can be obtained from the Version section of the API document corresponding to the client implementation

This will change the status to not ready indicating that it has logged in but cannot participate in a session as a source or destination.

Example 3.1. DevciceLoginRequest extended from AgentRequest

```
      <DeviceLoginRequestData serviceVersion="" softwareVersion="" assetTemplat
<DevicePrivateData></DevicePrivateData>
<DevicePrivateKeyData></DevicePrivateKeyData>
<DeviceAdminServiceInfo>
 <Port type='SrcPort/DstPort/NetworkPort/RelayPort/StorePort' id='1/2' ready=''
  <NetworkStatus />
  <DisplayConfig />
  <DisplayStatus />
 </Port>
</DeviceAdminServiceInfo>
<MediaServiceInfo>
  <StreamInfo/>
</MediaServiceInfo>
<LayoutServiceInfo>
 <StreamInfo/>
</LayoutServiceInfo>
<MediaStoreServiceInfo>
 <Dir uuid='' storeRevision=''/>
</MediaStoreServiceInfo>
</DeviceLoginRequestData>
```

- *serviceVersion - The service version of the software on the device (note this is different from software version)*
- *assetTemplateTitle - used if the device is to be dynamically created using a template*
- *type - device type*
- *ready - Flag indicating if all the resources are to be marked ready*
- *AssetPrivateData - data that is private to the device used to cache information to be retrieved on reboot*
- *AssetPrivateKeyData - the private key for the device which is used for authentication*
- *PortList - Sends the list of ports device owns*
  - *Port: type - type of the port. SrcPort/DstPort/StorePort/RelayPort/NetworkPort*
  - *Port: id - Id of the port*
  - *Port: state - flag indicates mark the port state as Ready/NotReady*
  - *Port: NetworkStatus - Flag includes the status of network (ipaddress/nat etc etc). Applicalbe for NetworkPort*
  - *Port: DisplayConfig - Flag icludes the display configuration information. Applicalbe for DstPort*

- *Port: DisplayStatus - Flag icludes the display status information. Applicalbe for DstPort []*
- *Port: DirList - List of the direcotries owned by StorePort*
  - *Dir: UUID - Unique id to associate DB and Store entry*
  - *Dir: storeRevision - indicates store at this revision. used in directory sync*
- *StreamInfo: - Flag icludes the list of the streams currenlty running*

If all is well, the response will contain the Device's private data(preferences) that has been configured/ stored on the server.

- The current date/time is also returned

Example 3.2. Service Login Response extended from AgentResponse

```
 <DeviceLoginResponseData>
<Device NID="default.assetresourcelist.arl_43a559b7-3437-41c2-9f4a-4080743118f
 <PortList>
  <Port id="1" type="NetworkPort">
   <NetworkStatus />
  </Port>
  <Port id="1" maxInstanceCount="1" streamType="V2D" type="SrcPort"
   NID="default.mediastreamioportresourcelist.msioprl_5108d614-8a83-4490-899c-
 </PortList>
 <DeviceXML
  NID="default.devicelist.dl_91a18a88-76b7-46af-8826-3210e084bbd8" />
 <DiscardStreamList>
  <MediaServiceInfo />
  <LayoutServiceInfo />
 </DiscardStreamList>
</Device>
</DeviceLoginResponseData>
```

Example 3.3. Service Login Response error extended from AgentResponse

```
 <Error code="1200">
  <Description>Service Version Mismatch serverVersion="2.30.4"</Description>
 </Error>
```