Car Connectivity Consortium

MirrorLink®

High Speed Media Link (HSML)

Version 1.2.2
(CCC-TS-054)

VERSION HISTORY

Version	Date	Comment
1.2.0	25 September 2013	Approved Version
1.2.1	21 August 2014	Approved Errata Version
1.2.2	30 April 2015	Approved Errata Version

LIST OF CONTRIBUTORS

Benesch, Matthias	Mercedes-Benz Research & Development North America
Brakensiek, Jörg	Microsoft Corporation
Chien, Joey	HTC Corp.
Christopher Seubert	Carmeq (for Volkswagen AG)
Falconnet, Gautier	PSA
Fernahl, Dennis	Carmeq (for Volkswagen AG)
Kirschner, Alexander	jambit GmbH
Langhammer, Matthias	jambit GmbH
Wei, Joe (Editor)	HTC Corp.

LEGAL NOTICE

The copyright in this Specification is owned by the Car Connectivity Consortium LLC ("CCC LLC"). Use of this Specification and any related intellectual property (collectively, the "Specification"), is governed by these license terms and the CCC LLC Limited Liability Company Agreement (the "Agreement").

Use of the Specification by anyone who is not a member of CCC LLC (each such person or party, a "Member") is prohibited. The legal rights and obligations of each Member are governed by the Agreement and their applicable Membership Agreement, including without limitation those contained in Article 10 of the LLC Agreement.

CCC LLC hereby grants each Member a right to use and to make verbatim copies of the Specification for the purposes of implementing the technologies specified in the Specification to their products ("Implementing Products") under the terms of the Agreement (the "Purpose"). Members are not permitted to make available or distribute this Specification or any copies thereof to non-Members other than to their Affiliates (as defined in the Agreement) and subcontractors but only to the extent that such Affiliates and subcontractors have a need to know for carrying out the Purpose and provided that such Affiliates and subcontractors accept confidentiality obligations similar to those contained in the Agreement. Each Member shall be responsible for the observance and proper performance by such of its Affiliates and subcontractors of the terms and conditions of this Legal Notice and the Agreement. No other license, express or implied, by estoppel or otherwise, to any intellectual property rights are granted herein.

Any use of the Specification not in compliance with the terms of this Legal Notice, the Agreement and Membership Agreement is prohibited and any such prohibited use may result in termination of the applicable Membership Agreement and other liability permitted by the applicable Agreement or by applicable law to CCC LLC or any of its members for patent, copyright and/or trademark infringement.

THE SPECIFICATION IS PROVIDED "AS IS" WITH NO WARRANTIES, EXPRESS OR IMPLIED, INCLUDING WITHOUT LIMITATION ANY WARRANTY OF MERCHANTABILITY, FITNESS FOR A PARTICULAR PURPOSE, NONINFRINGEMENT OF ANY THIRD PARTY INTELLECTUAL PROPERTY RIGHTS, AND COMPLIANCE WITH APPLICABLE LAWS.

Each Member hereby acknowledges that its Implementing Products may be subject to various regulatory controls under the laws and regulations of various jurisdictions worldwide. Such laws and regulatory controls may govern, among other things, the combination, operation, use, implementation and distribution of Implementing Products. Examples of such laws and regulatory controls include, but are not limited to, road safety regulations, telecommunications regulations, technology transfer controls and health and safety regulations. Each Member is solely responsible for the compliance by their Implementing Products with any such laws and regulations and for obtaining any and all required authorizations, permits, or licenses for their Implementing Products related to such regulations within the applicable jurisdictions.

Each Member acknowledges that nothing in the Specification provides any information or assistance in connection with securing such compliance, authorizations or licenses**.**

NOTHING IN THE SPECIFICATION CREATES ANY WARRANTIES, EITHER EXPRESS OR IMPLIED, REGARDING SUCH LAWS OR REGULATIONS. ALL LIABILITY, INCLUDING LIABILITY FOR INFRINGEMENT OF ANY INTELLECTUAL PROPERTYRIGHTS OR FOR NONCOMPLIANCE WITH LAWS, RELATING TO USE OF THE SPECIFICATION IS EXPRESSLY DISCLAIMED. BY USE OF THE SPECIFICATION, EACH MEMBER EXPRESSLY WAIVES ANY CLAIM AGAINST CCC LLC AND ITS MEMBERS RELATED TO USE OF THE SPECIFICATION.

CCC LLC reserve the right to adopt any changes or alterations to the Specification as it deems necessary or appropriate.

TABLE OF CONTENTS

LIST OF FIGURES

LIST OF TABLES

TERMS AND ABBREVIATIONS

EP	Endpoint
HSML	High Speed Media Link
ML	MirrorLink
UPnP	Universal Plug and Play
USB	Universal Serial Bus

HDCP is a registered trademark of Digital Content Protection LLC

MirrorLink is a registered trademark of Car Connectivity Consortium LLC

UPnP is a registered trademark of UPnP Forum.

Other names or abbreviations used in this document may be trademarks of their respective owners.

1 ABOUT

This document is part of the MirrorLink specification, which specifies an interface for enabling remote user interaction of a mobile device via another device. This specification is written having a car head-unit to interact with the mobile device in mind, but it will similarly apply for other devices, which do provide a colored display, audio input/output and user input mechanisms.

This specification describes the High Speed Media Link, a video transmission mechanism that utilizes the USB to project the screen of one device onto another device with a larger screen.

The specification lists a series of requirements, either explicitly or within the text, which are mandatory elements for a compliant solutions. Recommendations are given, to ensure optimal usage and to provide suitable performance. All recommendations are optional.

The key words "MUST", "MUST NOT", "REQUIRED", "SHALL", "SHALL NOT", "SHOULD", "SHOULD NOT", "RECOMMENDED", "MAY", and "OPTIONAL" in this document are following the notation as described in RFC 2119 [1].

1. MUST: This word, or the terms "REQUIRED" or "SHALL", mean that the definition is an absolute requirement of the specification.
2. MUST NOT: This phrase, or the phrase "SHALL NOT", mean that the definition is an absolute prohibition of the specification.
3. SHOULD: This word, or the adjective "RECOMMENDED", mean that there may exist valid reasons in particular circumstances to ignore a particular item, but the full implications must be understood and carefully weighed before choosing a different course.
4. SHOULD NOT: This phrase, or the phrase "NOT RECOMMENDED" mean that there may exist valid reasons in particular circumstances when the particular behavior is acceptable or even useful, but the full implications should be understood and the case carefully weighed before implementing any behavior described with this label.
5. MAY: This word, or the adjective "OPTIONAL", means that an item is truly optional. One vendor may choose to include the item because a particular marketplace requires it or because the vendor feels that it enhances the product while another vendor may omit the same item. An implementation which does not include a particular option MUST be prepared to interoperate with another implementation which does include the option, though perhaps with reduced functionality. In the same vein an implementation which does include a particular option MUST be prepared to interoperate with another implementation which does not include the option (except, of course, for the feature the option provides.)

2 INTRODUCTION

High Speed Media Link (HSML) is a screen out technology. The main purpose is to let mobile users project their phones' screens to a larger one, like the display inside a car infotainment system or PC, and users can control their phones via an automotive head unit or PC. With bigger screens, users can have much better usage experience. Of course, this document doesn't limit the usage scenarios only on mobile phones and automotive head units. The HSML can be applied to any device that conforms to this specification. As shown in Figure 1.

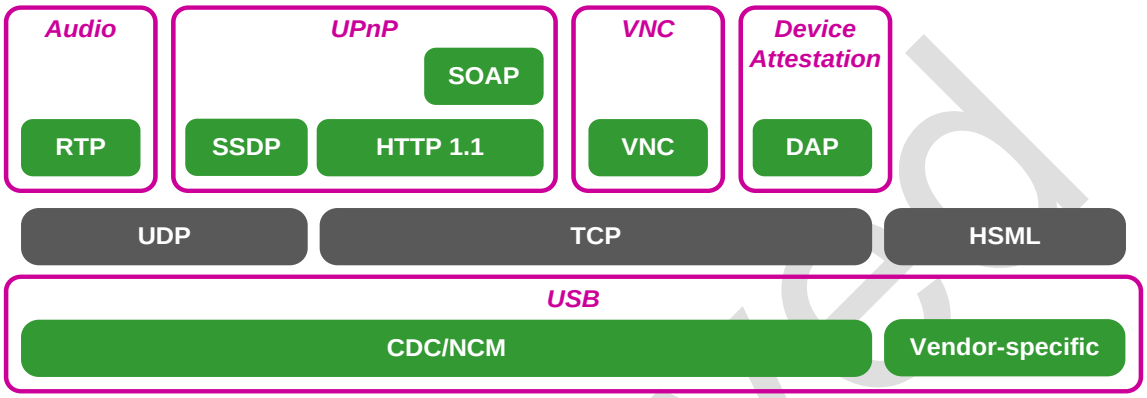

Figure 1: HSML Protocol Stack

There are two roles in the HSML architecture. The HSML source is the source of video data and the HSML sink is sink side. On the other hand, the control data is sent from HSML sink to HSML source.

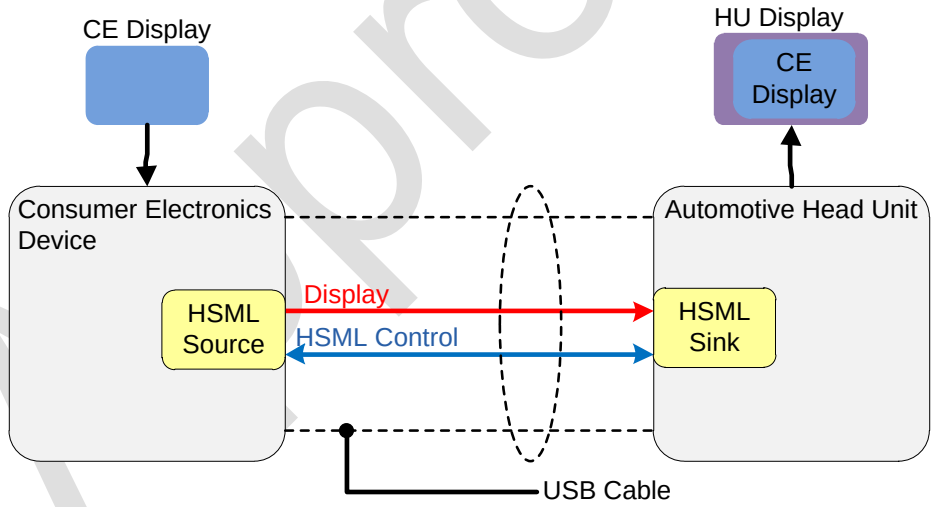

Figure 2: HSML Overview

The audio and control data are handled by following MirrorLink requirements: [2][3]. Therefore, any device that wants to comply with this specification MUST implement MirrorLink as well.

The MirrorLink client, providing HSML functionality, MUST implement the HSML sink functionality.

The MirrorLink server, providing HSML functionality, MUST implement the HSML source functionality.

3 HSML USB ARCHITECTURE

HSML is a USB function that can transfer display data efficiently. The figure below shows the USB architecture of HSML. Two pipes are established. The control pipe is used to send HSML specific requests. The framebuffer pipe is established for transmitting the uncompressed or compressed display data.

The device and host will be used in this section to refer to HSML source and HSML sink respectively because this section mainly describes HSML in the context of USB.

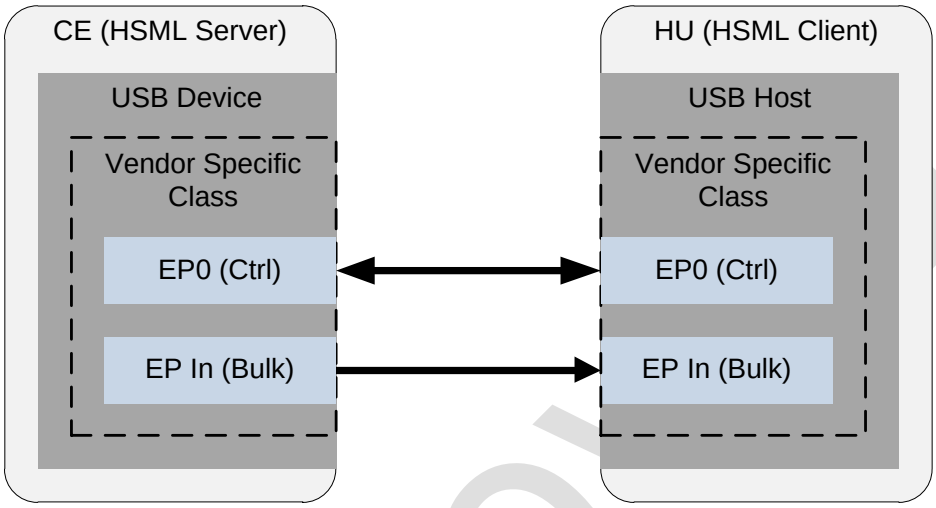

Figure 3: HSML USB Architecture

3.1 Functional Characteristics

The MirrorLink server MUST support at least two functions: one is HSML and the other is CDC/NCM which is compliant with HSML. The MirrorLink Server MUST include the HSML USB interface into the same USB configuration as the CDC/NCM USB interface. The HSML function is used for video transmission and the CDC/NCM is used for carrying MirrorLink traffic.

3.1.1 Interface

The HSML interface MUST be one of several interfaces the MirrorLink Server has in order to conform to this specification.

3.1.2 Endpoints

The device MUST contain two endpoints: Ctrl (Default) and Bulk In (Framebuffer).

3.1.2.1 Default

The default endpoint uses the control transfers as defined in the USB specification [4]. All the standard and vendor-specific requests are transmitted through this endpoint. The endpoint number MUST be zero (0).

3.1.2.2 Framebuffer

This endpoint is used to receive the framebuffer data from the device. This endpoint MUST use bulk transfers and the direction MUST be IN. The maximum packet size for USB 2.0 MUST be 512 bytes and for USB 3.0 MUST be 1024 bytes.

3.2 Vendor-Specific Codes

The below table defines the interface class, subclass and protocol used in the HSML interface descriptor.

Fields	Code	Description
Class	0xFF	Vendor specific class
Subclass	0xCC	CCC
Protocol	0x01	HSML

Table 1: Vendor-Specific Codes

To comply with this specification, the device SHOULD NOT have another USB vendor-specific interface whose subclass field is 0xCC and protocol field is 0x01. The detail of descriptor class definition rule is following USB specification in [4].

3.3 Interface Descriptor

The HSML interface descriptor is just like a standard USB interface descriptor, except some fields are dedicated to HSML as follows.

Offset	Fields	Size (Bytes)	Value	Description
0	bLength	1	Number	Size of this descriptor. (9 bytes)
1	bDescriptorType	1	Constant	Interface descriptor (0x04)
2	bInterfaceNumber	1	Number	Number of interface
3	bAlternateSetting	1	Number	Value used to select alternative setting
4	bNumEndpoints	1	Number	Number of Endpoints. This number MUST be 1 for a Bulk IN.
5	bInterfaceClass	1	Class	Interface Class Code. (0xFF)
6	bInterfaceSubClass	1	SubClass	Interface Subclass Code. (0xCC)
7	bInterfaceProtocol	1	Protocol	Interface Protocol Code. (0x01)
8	bInterface	1	Index	Index of a string descriptor that describes this interface

Table 2: HSML Interface Descriptor

Standard USB interface descriptor is defined in [4].

3.4 Endpoint Descriptors

The HSML interface requires 2 endpoints: one is default Control endpoint (endpoint 0), another is BULK IN endpoint as follows.

Offset	Fields	Size	Value	Description
0	bLength	1	Number	Size of this descriptor. (7 bytes)
1	bDescriptorType	1	Constant	Endpoint descriptor (0x05)
2	bEndpointAddress	1	Number	Endpoint number and direction. (The bit 7 SHOULD be 1 to indicates its direction is IN)
3	bmAttributes	1	Constant	Transfer type, Bulk. (0x02)
4	wMaxPacketSize	2	Number	Maximum packet size supported. (For USB 2.0, this value MUST be 512 and for USB 3.0, this value MUST be 1024.)
6	bInterval	1	Number	Service interval. (not used)

Table 3: HSML Bulk IN Endpoint Descriptor

Standard endpoint descriptor is defined in [4].

3.5 HSML Requests

Table 4 lists all of the HSML specific requests that are valid for the HSML interface. Requests marked as "Yes" in the mandatory field MUST be implemented by any conforming HSML device.

Request	Code	Mandatory	Description
GetVersion	0x40	Yes	Get and provide HSML versions of the HSML source and sink.
GetParameters	0x41	Yes	Request the device to report its capabilities and configurations.
SetParameters	0x42	Yes	Configure the device according to the device's and the host's capabilities.
StartFramebuffer-Transmission	0x43	Yes	Request the device to start sending framebuffer via the Bulk IN endpoint.
PauseFramebuffer-Transmission	0x44	Yes	Request the device to pause the framebuffer transmission if it's in streaming mode
StopFramebuffer-Transmission	0x45	Yes	Request the device to stop sending framebuffer if it's in streaming mode.
SetMaxFrameRate	0x46	Yes	Request the device to set the maximum framebuffer update rate. The device MUST NOT send framebuffer updates at a rate beyond the specified value.
GetIdentifier	0x47	Yes	Request the device to return a unique identifier.

Table 4: HSML Request List

The request format is defined in [4], Chapter 9. Note: Based on [4], all HSML requests MUST use little endian for any value 16, 32 or 64 bit value.

3.5.1 GetVersion

This request retrieves the maximum version that the device can support and tells the device which version the host can support at the same time.

bmRequestType	bRequestCode	wValue	wIndex	wLength	Data
11000001B	0x40	HSML Sink version	The interface number	2	First byte: Major version of HSML source Second byte: Minor version of HSML source

Table 5: GetVersion Request

The first byte of *wValue* field MUST be HSML sink major version and the second byte of *wValue* field MUST be HSML sink minor version. The value of *wValue* field and Data field returned from the HSML source MUST be both 0x0100 for this version of the specification.

HSML source and HSML sink MUST provide this function for backward compatibility.

3.5.2 GetParameters

This request is used by the host to get the capabilities and configuration of the device.

bmRequestType	bRequestCode	wValue	wIndex	wLength	Data
11000001B	0x41	0	The interface number	16	HSML Parameter structure. (See Table 7)

Table 6: GetParameters Request

Below is the HSML parameter structure.

Offset	Field	Size	Value	Description
0	bmCapabilities	4	Bitmap	Bit 0: BigEndian used Bit 1: FBUpdateOnChange supported Bit 2 to 31: Reserved. (Must be all zeroes)
4	wWidth	2	Number	The width of framebuffer that the device wants to use.
6	wHeight	2	Number	The height of framebuffer that the device wants to use.
8	bmPixelFormatSupported	4	Bitmap	Bit 0: 32-bit ARGB 888 (mandatory for HSML source) Bit 1: 24-bit RGB 888 Bit 2: 16-bit RGB 565 (mandatory for HSML source) Bit 3: 16-bit RGB 555 Bit 4: 16-bit RGB 444 Bit 5: 16-bit RGB 343 Bit 6 to 31: Reserved. (Must be all zeroes)
12	bmEncodingSupported	4	Bitmap	Bit 0: RAW data (Mandatory) Bit 1: SRLE (Scan Line based Run Length Encoding)[1] Bit 2 to 31: Reserved. (Must be all zeros)

Table 7: HSML Parameter Structure

If the endianness of framebuffer data is big endian, the device MUST set bit 0 of *bmCapabilities* field to 1. Otherwise, it MUST set this bit to 0 to indicate that its native framebuffer is in little endian. If the device supports sending the framebuffer only when its display data changed, it MUST set bit 1 of *bmCapabilities* field to 1.

The device MUST set *wWidth* and *wHeight* according to its framebuffer resolution.

The device MUST support ARGB 888 and RGB 565 pixel formats.

The *wEncodingSupported* field indicates encodings of framebuffer that the device supports. The device MUST support the RAW encoding, i.e. the bit 0 MUST be set to 1.

3.5.3 SetParameters

This request is used to tell the device which configuration that the host wants to use.

bmRequestType	bRequestCode	wValue	wIndex	wLength	Data
01000001B	0x42	0	The interface number	12	HSML Configuration structure. (See Table 9)

[1] See Section 6.4 in [3].

Table 8: SetParameters Request

Below is the HSML configuration structure.

Offset	Field	Size	Value	Description
0	bmCapabilities	4	Bitmap	Bit 0: BigEndian used Bit 1: FBUpdateOnChange used. Bit 2 to 31: Reserved. (Must be all zeroes)
4	bPixelFormat	1	Number	0: 32-bit ARGB 888 1: 24-bit RGB888 2: 16-bit RGB 565 3: 16-bit RGB 555 4: 16-bit RGB 444 5: 16-bit RGB 343 6 to 31: Reserved. 32 to 255: Undefined
5	bPadding	1	0	Padding
6	wPadding	2	0	Padding
8	bmEncodingSup-ported	4	Bitmap	Bit 0: RAW (Mandatory) Bit 1: SRLE (Scan Line based Run Length Encoding) Bit 2 to 31: Reserved. (Must be all zeroes.)

Table 9: HSML Configuration Structure

The host MUST set bit 0 of *bmCapabilities* field to 1, if the endianness of its framebuffer data is big endian. Otherwise, it MUST set this bit to 0 to indicate that its native framebuffer is in little endian. The device MUST follow the host's capability and send framebuffer data accordingly.

The host SHOULD set the *FBUpdateOnChange* bit to avoid unnecessary USB bandwidth usage if the device supports it.

The host MUST select only color formats supported from the server. The host MAY send `SetParameters` request any time when the host wants to change the pixel format.

The host can use *bmEncodingSupported* field to indicate the device what encodings it supports. If both host and device support multiple encodings, then device can take advantage of it by encoding the framebuffer based on display contents. For example, if the current display content is a pure UI, then the device MAY encode the framebuffer with RLE. On the other hand, if the content is a movie, then the device MAY use RAW encoding.

The device MUST respond with a STALL if it can't handle the configuration from the host.

The host MUST NOT send this request without sending the `GetParameters` request first.

3.5.4 StartFramebufferTransmission

This request asks the device to start sending framebuffer via the Bulk IN endpoint.

bmRequestType	bRequestCode	wValue	wIndex	wLength	Data
01000001B	0x43	0x0000: Streaming Mode 0x0001: On-Demand Mode	The in-terface number	0	None

Table 10: StartFramebufferTransmission Request

The device will send the framebuffer continuously in the streaming mode. This mode is purely designed for performance. It gives you the minimum latency and the highest possible frame rate. The host should try to keep up with the device, otherwise the user may experience some delay.

The On-Demand mode lets the host decide when it needs an updated framebuffer. Every time the device receives this request with the On-Demand Mode set in the *wValue* field, it MUST send a framebuffer to the host. This mode saves the bandwidth on expense of slightly increased latency.

3.5.5 PauseFramebufferTransmission

This request asks the device to pause framebuffer transmission via the Bulk IN endpoint.

bmRequestType	bRequestCode	wValue	wIndex	wLength	Data
01000001B	0x44	0	The interface number	0	None

Table 11: PauseFramebufferTransmission Request

The device MUST stop any pending framebuffer transmissions when receiving this request but maintain all configurations and the host MUST discard all queued data after sending this request. The host can restart the framebuffer transmission by sending the `StartFramebufferTransmission` at a later time without configuring the device again. This request MUST NOT be sent at the on-demand mode.

3.5.6 StopFramebufferTransmission

This request stops the device from sending framebuffer at the streaming mode.

bmRequestType	bRequestCode	wValue	wIndex	wLength	Data
01000001B	0x45	0	The interface number	0	None

Table 12: StopFramebufferTransmission Request

The device MUST cease all activities on the Bulk IN endpoint and clean all remaining data in the queue when it receives this request. This request MUST NOT be sent at the on-demand mode.

3.5.7 SetMaxFrameRate

This request sets the upper bound of the device's frame rate.

bmRequestType	bRequestCode	wValue	wIndex	wLength	Data
01000001B	0x46	Maximum frames per second	The interface number	0	None

Table 13: SetMaxFrameRate Request

The frame rate is defined as 1 second divided by the number of framebuffer updates. The device SHOULD send the framebuffer at a regular interval according the value in the *wValue* field. For instance, if the *wValue* is 30, the device SHOULD send a framebuffer every 33 milliseconds or more.

The device MUST NOT send framebuffer at a rate higher than the value specified in the *wValue* field and the host can send this request any time after the initialization phase.

3.5.8 GetIdentifier

This request can be used to identify multiple HSML devices when they're all connected to the same host.

bmRequestType	bRequestCode	wValue	wIndex	wLength	Data
11000001B	0x47	0	The interface number	16	UUID version 4

Table 14: GetIdentifier

The device MUST return a Universally Unique IDentifier (UUID) version 4, as defined in [5], upon receiving this request.

Note: The UUID value in HSML is represented as 16 RAW bytes. The UUID in the UPnP XML is represented as a 36 bytes String. The HSML UUID and UPnP Server Device XML UUID MAY represent the same underlying value.

Example, where the UPnP and HSML UUID represent the same value:

UPnP UUID:

```
"uuid:2fac1234-31f8-11b4-a222-08002b34c003"
```

HSML UUID:

```
Byte(0x2f), Byte(0xac), Byte(0x12), Byte(0x34), Byte(0x31),
Byte(0xf8), Byte(0x11), Byte(0xb4), Byte(0xa2), Byte(0x22),
Byte(0x08), Byte(0x00), Byte(0x2b), Byte(0x34), Byte(0xc0),
Byte(0x03)
```

4 HSML FRAMEBUFFER TRANSMISSION PROTOCOL

There are two phases of the HSML protocol:

- **Initialization Phase**: To let HSML source and HSML sink to load proper drivers that can communicate with each other and exchange the configuration messages, as shown in Figure 3.
- **Transmission Phase**: HSML source sends the display data and HSML sink sends the control data at this phase.

4.1 Managing an HSML Connection

4.1.1 Identifying Remote Applications

HSML is identified as specific encoding within an existing VNC connection. Therefore remote applications MUST be listed and handled as VNC based applications with *protocolID* set to "`VNC`".

During a VNC session the MirrorLink client MUST list HSML pseudo encoding (-527) within VNC Set Encoding message, to indicate the support of HSML. If MirrorLink server supports the HSML as well, it MUST send VNC Framebuffer Update messages with Context Information (-524) and HSML pseudo encoding (-527) in response to Framebuffer Update Request messages from the MirrorLink client.

4.1.2 Establishing the HSML Connection

A VNC connection MUST be established from the MirrorLink client prior the HSML connection as defined in [3]. The HSML connection MUST be established after VNC initialization phase as defined in chapter 4.2.1.

4.1.3 Intentionally Terminating the HSML Connection

The MirrorLink client and server can initiate terminating both connections anytime by sending a VNC `ByeBye` message as specified in [3].

The MirrorLink client MUST terminate the HSML connection by sending a `StopFramebufferTransmission` request, as specified in 3.5.6, when the MirrorLink Client is in streaming mode, as shown in Figure 4, and MUST not send any further `StartFramebufferTransmission` requests, as specified in 3.5.4.

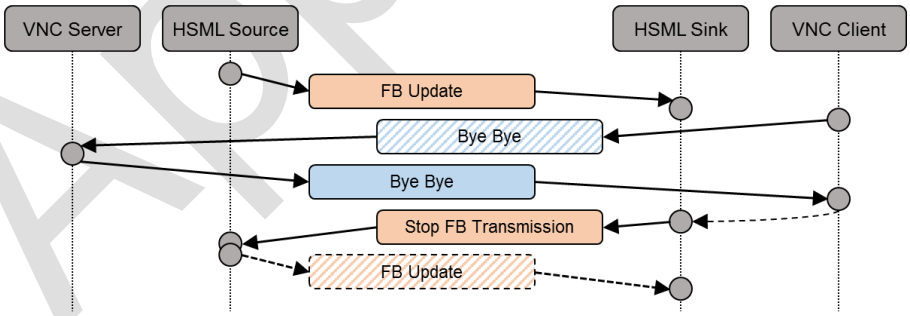

Figure 4: Message Sequence for Intentionally Terminating HSML (Streaming Mode)

If the MirrorLink Client decides to re-establish the VNC session, it MUST follow the steps given in section 4.1.2. The MirrorLink Server MUST keep the USB HSML

4.1.4 Unintentionally Terminating the HSML Connection

The HSML connection may be disconnected unintentionally in case of any error conditions. The recognizing entity MUST then terminate the VNC connection by sending a VNC `ByeBye` message as specified in [3].

The VNC connection may be disconnected unintentionally in case of any error conditions. The MirrorLink client MUST then terminate the HSML connection, following the steps defined in 4.1.3.

If the MirrorLink Client decides to re-establish the VNC session, it MUST follow the steps given in section 4.1.2.

4.2 HSML Protocol Phases

4.2.1 Initialization Phase

Before establishing the HSML connection, a VNC connection MUST be established, because HSML depends on it to configure display parameters, provide context information and handle user input events. Therefore, the whole initialization sequence MUST be as shown in Figure 5.

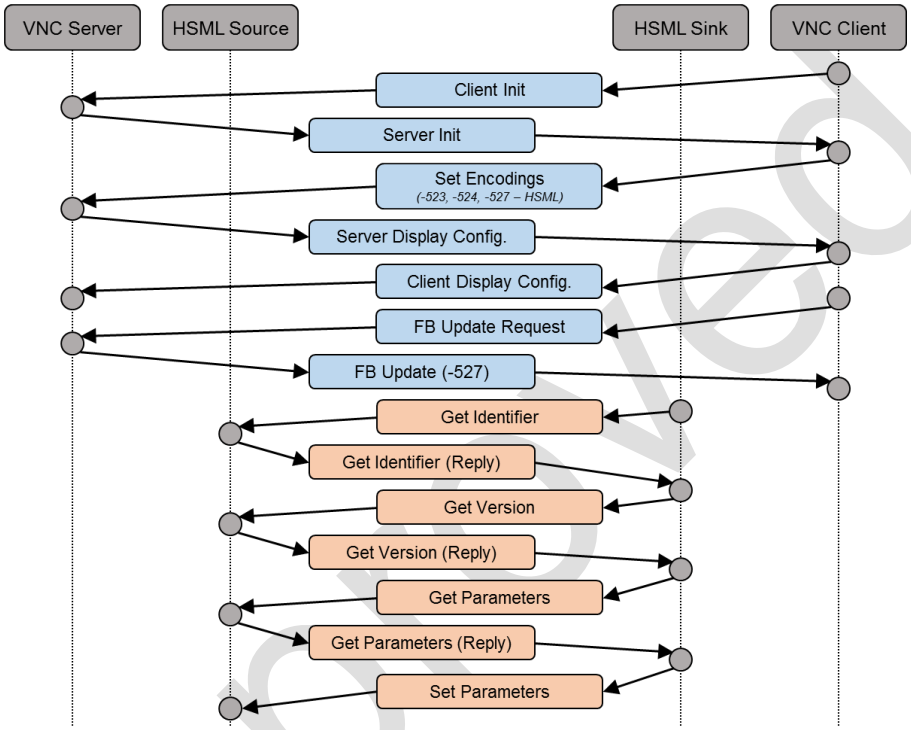

Figure 5: Message Sequence for HSML Initialization Phase

After VNC handshaking phase as defined in [3] and exchanging `ClientInit` and `ServerInit` messages, the MirrorLink Client MUST indicate the support for HSML by listing HSML pseudo encoding (-527) within the `SetEncodings` message during the initialization phase of the VNC connection. The encoding order of supported framebuffer encodings MAY be used from the HSML Source as an indication on the HSML sink's priority order (first entry has highest priority). After receiving the initial `FramebufferUpdate` message with HSML pseudo encoding (-527), the MirrorLink Client MUST initialize the HSML connection on USB layer.

The MirrorLink Client MAY establish connections to multiple MirrorLink servers that all support HSML simultaneously. In this case, the MirrorLink client MUST use the `GetIdentifier` request to distinguish between them. The MirrorLink server MUST send the same UUID value in both the returned value of the `GetIdentifier` request and the HSML pseudo encoding.

The `GetVersion` request tells HSML sink what HSML version the HSML source is going to use for the subsequent communications. HSML sink MAY send `GetParameters` request after receives the reply of `GetVersion` request.

The `GetParameters` request is used to learn about the capabilities and configurations of HSML source. HSML sink MUST use the returned values to select the configuration which HSML sink can support. Once selected the preferred configuration, HSML sink MUST send `SetParameters` request to HSML source. HSML source MUST adopt the configuration from HSML sink.

The HSML resolution *wWidth* and *wHeight* as provided via `GetParameters` request MUST follow the resolution negotiated within the VNC context.

The HSML initialization MUST be completed in 1 sec.

4.2.2 Transmission Phase

4.2.2.1 Handling Context Information

The VNC connection is used to exchange context information via framebuffer update request – response mechanism. Instead of any framebuffer pixel data a specific HSML pseudo encoding (-527) is used to identify the out-of-band transfer via HSML.

# bytes	Type	Value	Description
2	U16	0	X-position of rectangle (top left corner)
2	U16	0	Y-position of rectangle (top left corner)
2	U16	width	Width of the negotiated framebuffer resolution
2	U16	height	Height of the negotiated framebuffer resolution
4	S32	-527	Encoding type
2	U16	16	The length of unique identifier
16	Array of U8		UUID version 4.

Table 15: HSML Pseudo Encoding

After receiving the initial `FramebufferUpdate` message and in response to any VNC Framebuffer Update messages, the MirrorLink client MUST immediately send a `VNC FramebufferUpdateRequest` message. The VNC `FramebufferUpdateRequest` MUST include the whole framebuffer area, with X and Y position set to 0 (zero) and width and height according to the negotiated framebuffer resolution.

The MirrorLink server MUST respond by sending a `FramebufferUpdate` message with Context Information (-524) and HSML pseudo encoding (-527) instead of any framebuffer pixel data. While the HSML connection is active the MirrorLink Client MUST ignore and the MirrorLink Server MUST NOT send any framebuffer pixel data within `FramebufferUpdate` messages.

Depending on incremental flag within `FramebufferUpdateRequest` message, the MirrorLink server:

- MUST immediately send context information for the requested area, independent of whether it has changed or not, if incremental flag is set to '0' (non-incremental), as shown in Figure 6.
- MUST send context information for the requested area, once context information has changed, if incremental flag is set to '1' (incremental), as shown in Figure 7.
- MAY send context information independent of whether it has changed or not, if incremental flag is set to '1' (incremental). In that case it is RECOMMENDED to wait at least 100ms.

The MirrorLink client and server MUST also follow additional requirements defined in [3].

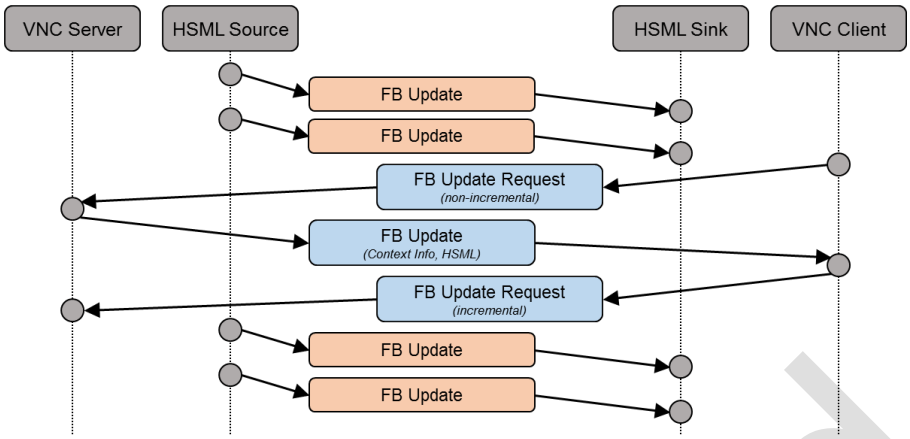

Figure 6: Message Sequence for on-Demand Context Information

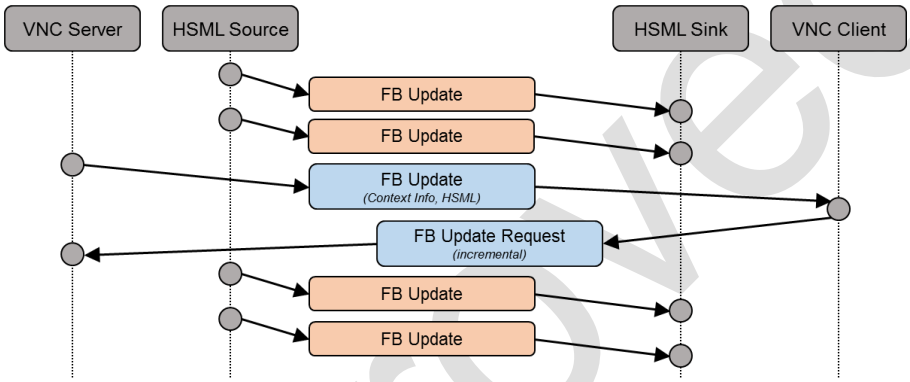

Figure 7: Message Sequence for on-Change Context Information

4.2.2.2 Handling Display Data

Once HSML source and HSML sink exchange the required information, HSML source can start to send the display data on USB layer.

There are two ways to transmit the display data from HSML source to HSML sink.

1. Streaming Mode
2. On-Demand Mode

HSML source MUST support both Streaming Mode and On-Demand Mode.

HSML source MUST send framebuffer continuously in the streaming mode, as shown in the following figure.

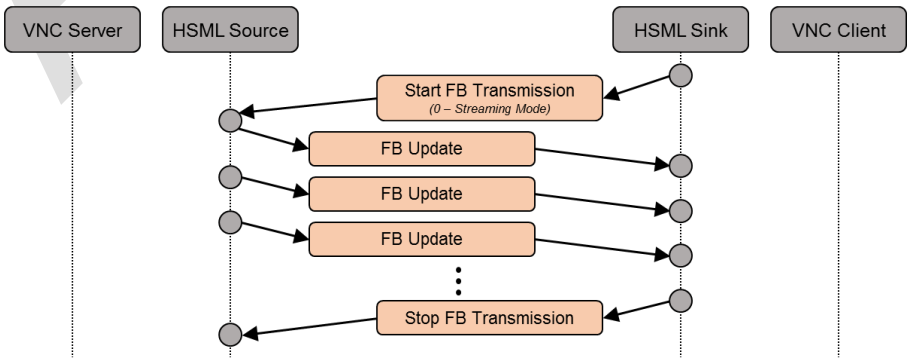

Figure 8: Message Sequence for Transmission Phase (Streaming Mode)

In on demand mode the HSML source MUST only send framebuffer if requested by the HSML sink regardless of whether the *FBUpdateOnChange* in the `SetParameters` is set, as shown in the following figure.

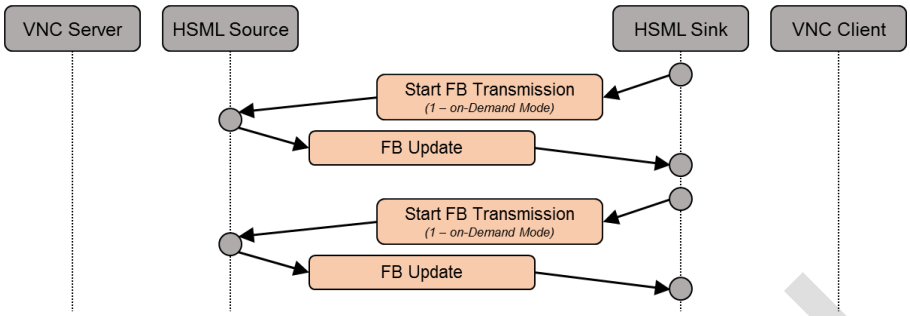

Figure 9: Message Sequence for Transmission Phase (On-Demand Mode)

Every framebuffer has to be preceded by a header as shown in Table 16 and it MUST send the header's individual fields in network byte order (most significant byte first).

# bytes	Type	Value	Description
8	Array of U8	0xFFFF48534D4CFFFF	Signature of frame header
4	U32		Sequence number
4	U32		Timestamp
4	U32		Framebuffer data size in byte, not including this header.
2	U16		Width
2	U16		Height
1	U8	0: 32-bit ARGB 888 1: 24-bit RGB888 2: 16-bit RGB 565 3: 16-bit RGB 555 4: 16-bit RGB 444 5: 16-bit RGB 343 6 to 31: Reserved. 32 to 255: Undefined	Pixel format
1	U8	0: RAW 1: SRLE 2 to 31: Reserved 32 to 255: Undefined	Encoding
486 for USB 2.0 and 998 for USB 3.0	Array of U8	All zeros	Reserved.

Table 16: Framebuffer Header

The signature field is used to synchronize the framebuffer when some data is lost in transit.

The sequence number is incremented by one for each framebuffer sent. It MAY be used to calculate the frame rate on the HSML sink side. The timestamp field MUST be recorded at the time of the frame being captured and the unit MUST be millisecond. This value MUST be derived from a clock that increments monotonically and linearly in time and the initial value of it is random.

The width and height fields MUST follow the negotiated framebuffer resolution within the VNC context. The pixel format and encoding fields MUST be equivalent to those of `SetParameters` request.

The bulk transfer with framebuffer header and framebuffer data MUST be formatted like Figure 10 below.

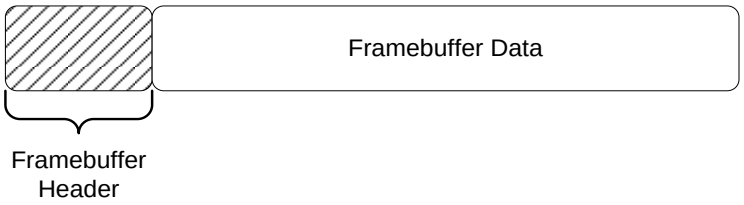

Figure 10: Bulk Transfer Framebuffer Format

Framebuffer Data MUST include full screen information no matter if it is raw data or compressed data. Framebuffer Data MUST be orientated and rotated as negotiated within the VNC context. Framebuffer Data MUST be formatted according the *PixelFormat* set via `SetParameter` request and as defined in [3].

The HSML source MUST use a USB short packet mechanism to implement segment delineation as specified in [6].When a framebuffer transfer (Header + Data) spans *N* USB packets, the first packet through packet *N-1* MUST be the maximum packet size defined for the USB endpoint. If the *N*th packet is less than maximum packet size the USB transfer of this short packet will identify the end of the segment. If the *N*th packet is exactly the maximum packet size, it shall be followed by a zero-length packet (which is a short packet) to assure the end of segment is properly identified [6].

4.2.2.3 Handling Framebuffer Resolution Change

The MirrorLink client and server can change framebuffer resolution as defined in [3].

The HSML source MUST follow changed resolutions immediately and send HSML Framebuffer Data with updated resolution. The HSML sink MUST render Framebuffer Data with updated resolution.

The message sequence the HSML source and sink MUST follow is shown in Figure 11, in case the resolution is changed during run-time, i.e. after the HSML transfer has started. The HSML source MUST stop sending HSML packets immediately after having received the `StopFramebufferTransmission` message, but it MUST still finish an already ongoing transfer of an HSML frame. Therefore, the HSML Sink MUST be able to receive further HSML packets, containing the old size, after sending the `StopFramebuffer-Transmission` message.

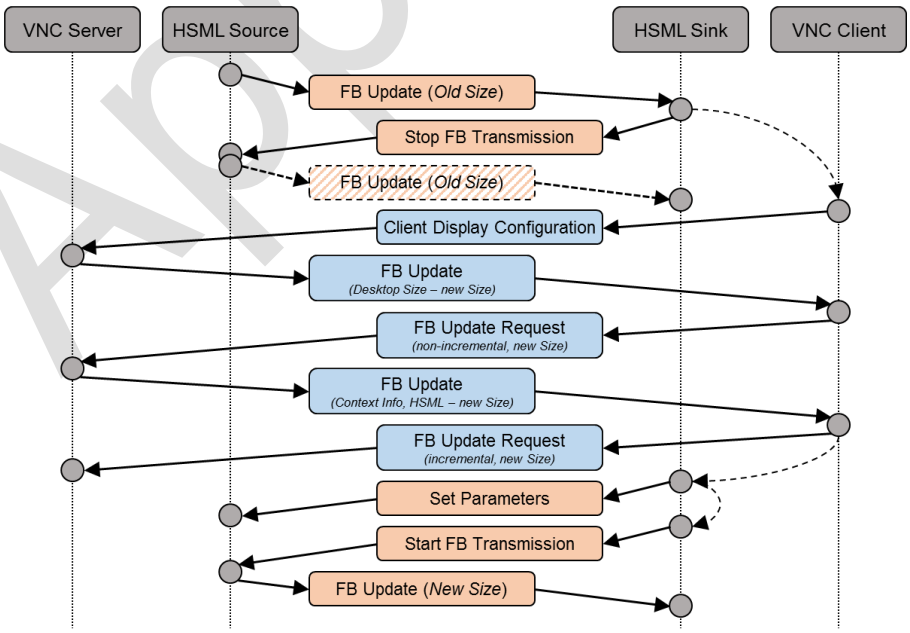

Figure 11: Message Sequence for successful run-time Framebuffer Resolution Change

On reception of the `ClientDisplayConfiguration` message, the VNC Server MUST send a `FramebufferUpdate` message containing a `DesktopSize` pseudo encoding rectangle with the new framebuffer resolution. In case the VNC Server does not support the new resolution, it MUST send a `FramebufferUpdate` message containing a `DesktopSize` pseudo encoding rectangle with the old framebuffer resolution, as shown in Figure 12

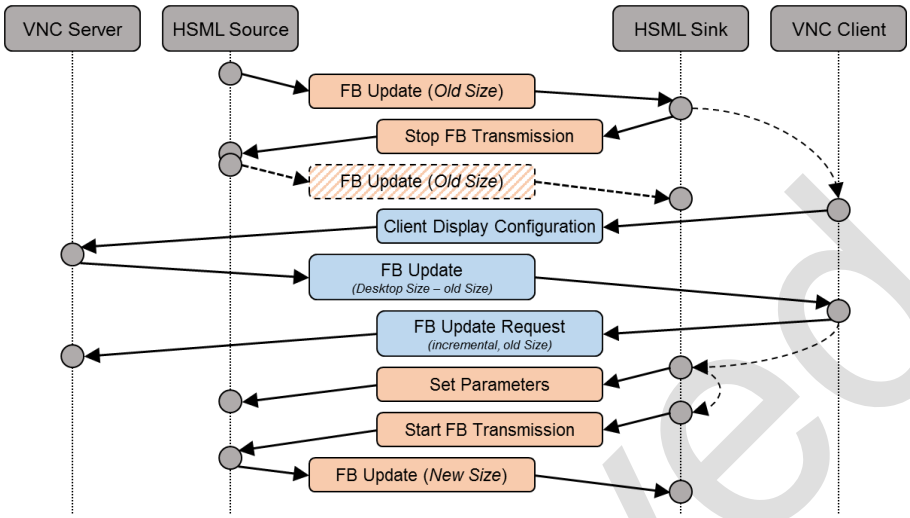

Figure 12: Message Sequence for unsuccessful run-time Framebuffer Resolution Change

In case the MirrorLink Server wants to change the framebuffer resolution during the initial handshake, the following very similar message sequence MUST be followed, as given in Figure 13:

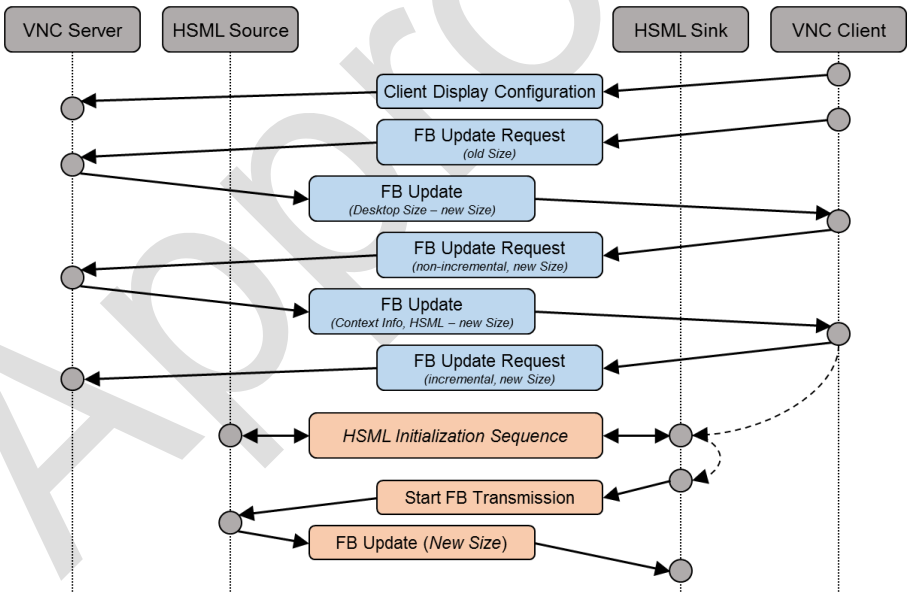

Figure 13: Message Sequence for initial Framebuffer Resolution Change

The MirrorLink Server and Client MUST both follow the framebuffer scaling and aspect ratio requirements defined in [3].

4.2.2.4 Handling Framebuffer Format Change

The HSML sink can change the pixel format and encoding of framebuffer during the transmission phase by sending the `SetParameters` request. It MUST follow the procedure in Figure 14.

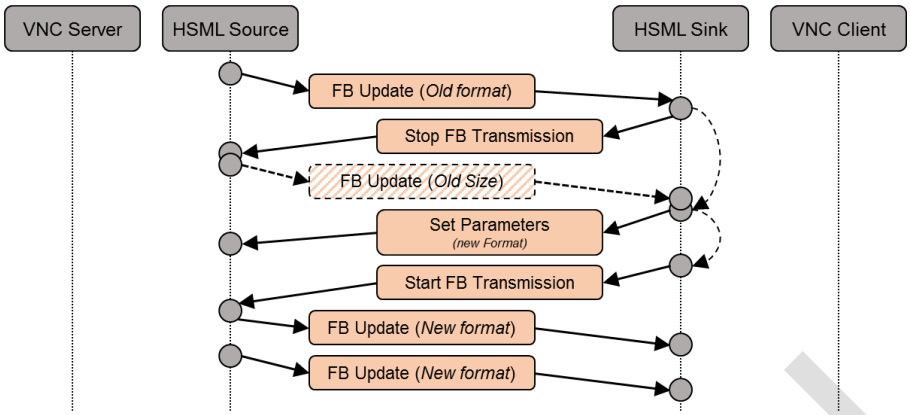

Figure 14: Message Sequence for Framebuffer Format Change in Streaming Mode

After changing the pixel format, the HSML sink MUST ignore framebuffer whose format is not identical to the format values in `SetParameters` request because it may receive several framebuffers that are still using the old format in streaming mode. The HSML source MUST keep its current pixel format, if the HSML sink attempts to set a pixel format value in `SetParamters` request, which is not supported from the HSML source.

4.2.2.5 Handling Framerate Adjustment

The HSML sink can limit the maximum frame rate that the HSML source can send by following the procedure in Figure 15.

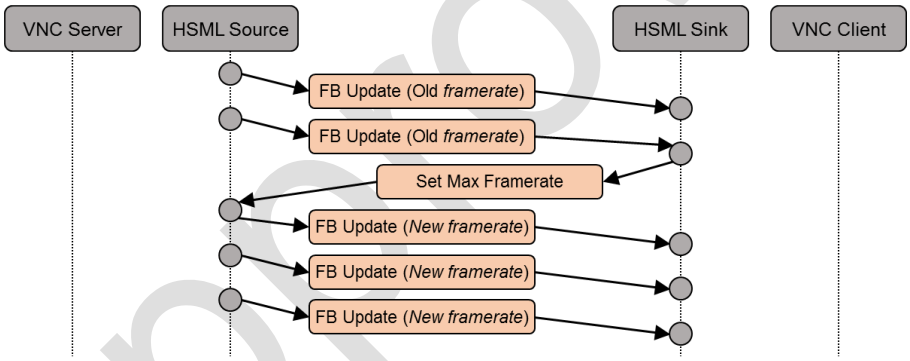

Figure 15: Message Sequence for Framerate Adjustment

4.2.2.6 Handling Framebuffer Blocking Notification

The HSML source MUST be compliant with the Framebuffer Blocking Notification mechanism in [3], the `FramebufferBlocking` Notification is sent by VNC Client still and the example flow is shown in Figure 16 for the example situation, when the MirrorLink Client is blocking the framebuffer after receiving new Context Information.

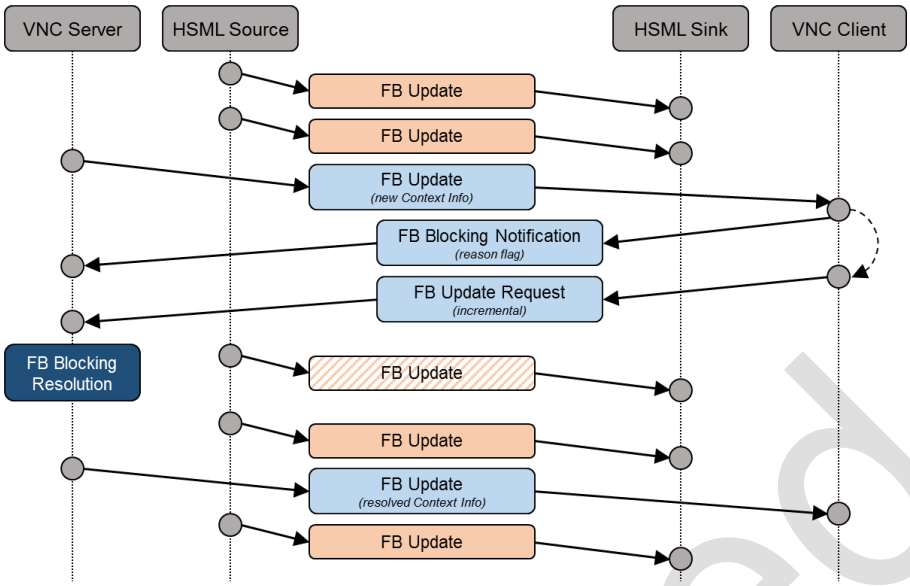

Figure 16: Message Sequence for Framebuffer Blocking on new Framebuffer Context Info

HSML source MUST follow one of the actions defined in [3] on reception of a framebuffer blocking notification message to resolve the framebuffer blocking.

In case the MirrorLink Client is moving the MirrorLink screen to the background the HSML Sink MUST pause the HSML transfer, sending a `PauseFramebufferTransmission` message in streaming mode, as shown in Figure 17. In case of on-demand mode, the HSML Sink MUST stop sending `FramebufferUpdateRequest` messages.

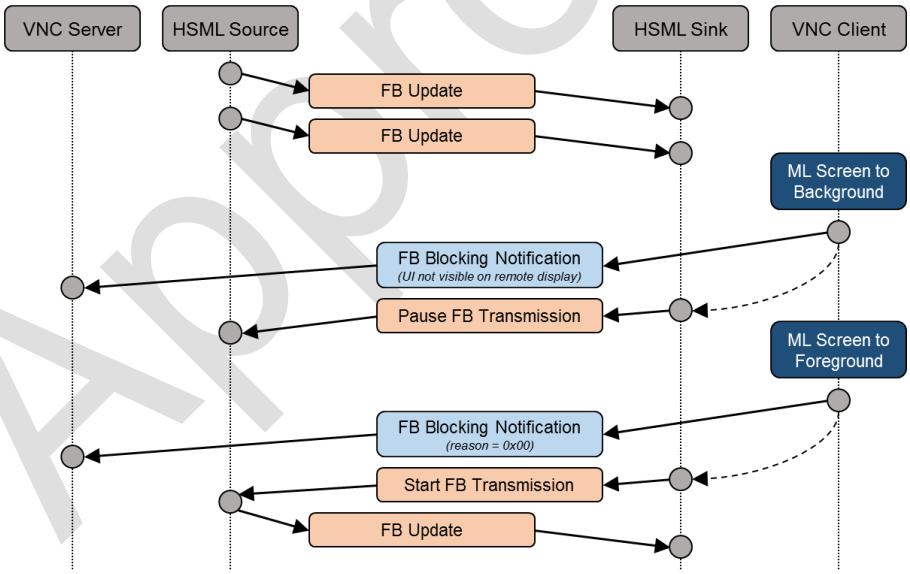

Figure 17: Message Sequence moving the MirrorLink Screen to the Background

The HSML Sink MUST resume the HSML transfer sending a `StartFramebufferTransmission` message (in streaming mode) or a `FramebufferUpdateRequest` message (in on-demand mode), as soon as the MirrorLink screen is in the foreground again.

4.2.2.7 Handling Orientation Changes

The HSML Sink MUST provide the framebuffer content in landscape orientation. Portrait orientation is not supported by the current specification and MUST NOT be used. The MirrorLink Client and Server SHOULD

therefore disable the orientation switch support flag (bit 0 of the *Framebuffer Configuration* word) in the VNC Client and Server Configuration messages.

In case the MirrorLink Client nevertheless attempts to change the orientation, the MirrorLink Server MUST reject the change, using the message exchange shown in Figure 18.

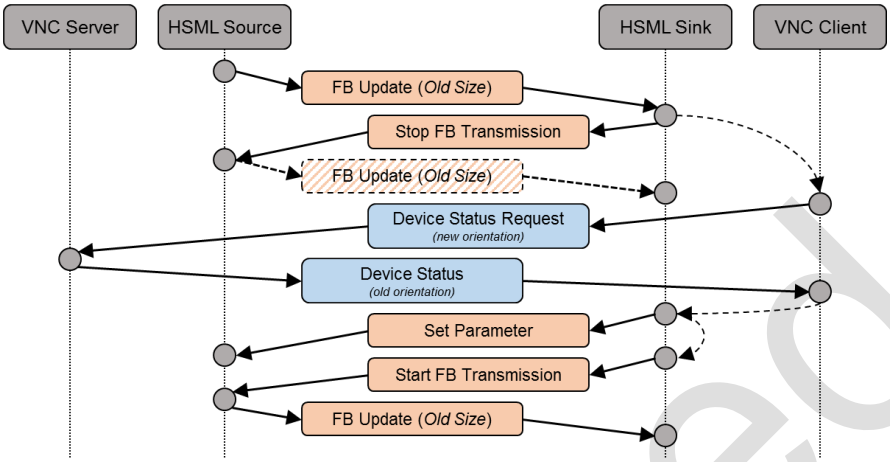

Figure 18: Message Sequence rejecting Orientation Change from MirrorLink Client

In case the MirrorLink Server nevertheless attempts to change the orientation, the MirrorLink Client MUST reject the change, using the message exchange shown in Figure 18.

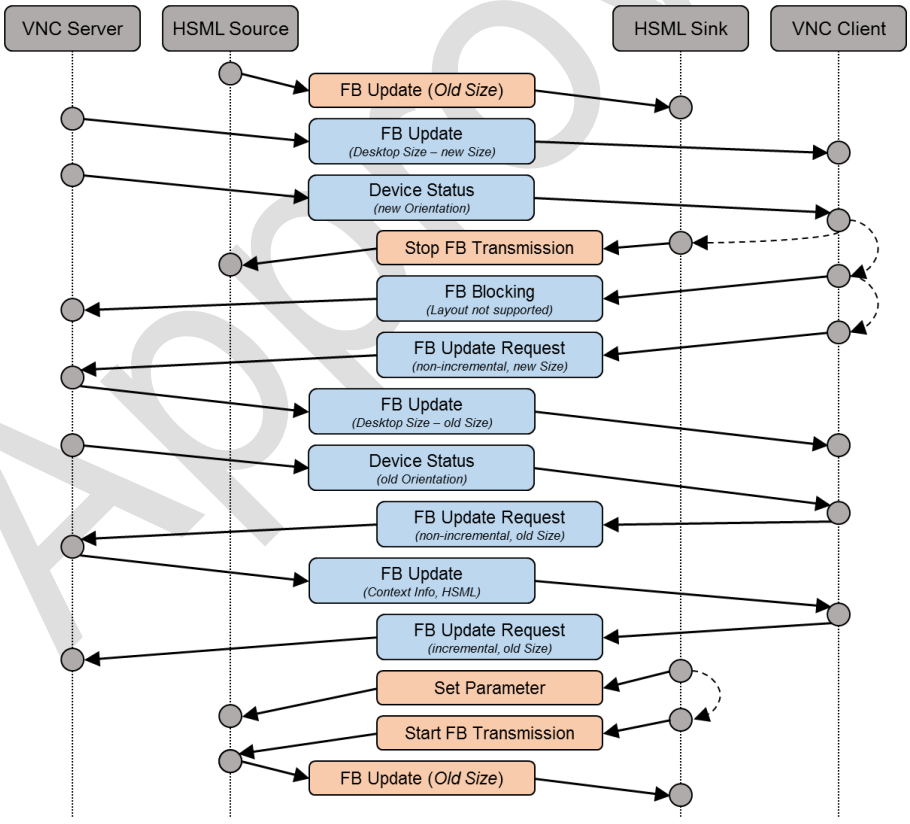

Figure 19: Message Sequence rejecting Orientation Change from MirrorLink Server

In case the MirrorLink Client has disabled bit 0 of the *Framebuffer Configuration* word in the `ClientDisplayConfiguration` message, the MirrorLink Client MAY intentionally terminate the current foreground application or the current VNC Session, rather than using the message sequence of Figure 18.

4.2.2.8 Handling Content Attestation

The current HSML specification only supports the attestation of the Context Information via the exchange VNC Content Attestation messages. See [3] for details.

4.2.3 HSML Protocol Finite State Machine

Below is a diagram describing state transition of the HSML source and sink after receiving various requests from the HSML sink.

- `Uninitialized` - Initial State; HSML Source and Sink are uninitialized.
- `Unconfigured` - HSML Source and Sink are initialized but not configured.
- `Configured` - HSML Source and Sink are initialized and configured; HSML streaming stopped.
- `Started` - HSML Source and Sink are initialized and configured; HSML streaming ongoing.

On termination of the related VNC session, the HSML source and sink MUST return to an uninitialized state, i.e. the HSML sink MUST follow the HSML initialization phase, as defined in section 4.1, when a HSML connection is setup the next time.

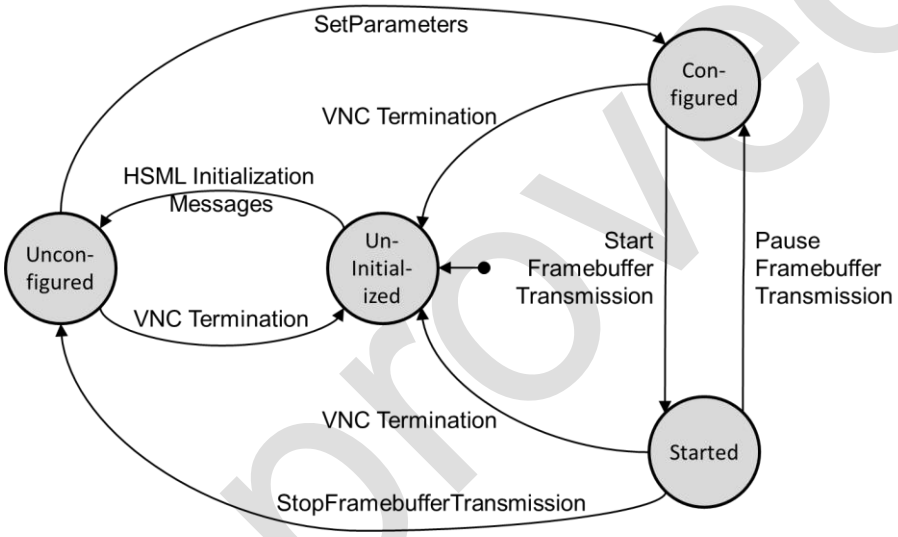

Figure 20: HSML Protocol Finite State Machine Diagram

5 REFERENCES

[1] IETF, RFC 2119, "Keys words for use in RFCs to Indicate Requirement Levels", March 1997. http://www.ietf.org/rfc/rfc2119.txt

[2] Car Connectivity Consortium, "MirrorLink – Audio", Version 1.1, March 31, 2012. CCC-TS-012.

[3] Car Connectivity Consortium, "MirrorLink – VNC based Display and Control", Version 1.1, March 31, 2012. CCC-TS-010.

[4] USB 2.0, "Universal Serial Bus Specification", Revision 2.0, April 27, 2000.

[5] IETF, RFC 4122, "A Universally Unique Identifier (UUID) URN Namespace", July 2005. http://www.ietf.org/rfc/rfc4122.txt

[6] USB IF, "Universal Serial Bus, Communications Class, Subclass Specification for Ethernet Control Model Devices", Revision 1.2, February 9, 2007.