ETSI TS 103 544-21 V1.3.1 (2019-10)

TECHNICAL SPECIFICATION

Publicly Available Specification (PAS); Intelligent Transport Systems (ITS); MirrorLink®; Part 21: High Speed Media Link (HSML)

Reference

RTS/ITS-98-21

Keywords

interface, ITS, PAS, smartphone

ETSI

650 Route des Lucioles
F-06921 Sophia Antipolis Cedex - FRANCE

Tel.: +33 4 92 94 42 00 Fax: +33 4 93 65 47 16

Siret N° 348 623 562 00017 - NAF 742 C
Association à but non lucratif enregistrée à la
Sous-Préfecture de Grasse (06) N° 7803/88

Contents

Intellectual Property Rights 5

Foreword 5

Modal verbs terminology 5

1 Scope 6

2 References 6
2.1 Normative references 6
2.2 Informative references 6

3 Definition of terms, symbols and abbreviations 7
3.1 Terms 7
3.2 Symbols 7
3.3 Abbreviations 7

4 Introduction 7

5 HSML USB Architecture 8
5.1 General 8
5.2 Functional Characteristics 9
5.2.1 General 9
5.2.2 Interface 9
5.2.3 Endpoints 9
5.2.3.1 General 9
5.2.3.2 Default 9
5.2.3.3 Framebuffer 9
5.3 Vendor-Specific Codes 9
5.4 Interface Descriptor 9
5.5 Endpoint Descriptors 10
5.6 HSML Requests 10
5.6.1 General 10
5.6.2 GetVersion 11
5.6.3 GetParameters 11
5.6.4 SetParameters 12
5.6.5 StartFramebufferTransmission 13
5.6.6 PauseFramebufferTransmission 14
5.6.7 StopFramebufferTransmission 14
5.6.8 SetMaxFrameRate 14
5.6.9 GetIdentifier 15

6 HSML Framebuffer Transmission Protocol 15
6.1 General 15
6.2 Managing an HSML Connection 15
6.2.1 Identifying Remote Applications 15
6.2.2 Establishing the HSML Connection 15
6.2.3 Intentionally Terminating the HSML Connection 16
6.2.4 Unintentionally Terminating the HSML Connection 16
6.3 HSML Protocol Phases 16
6.3.1 Initialization Phase 16
6.3.2 Transmission Phase 17
6.3.2.1 Handling Context Information 17
6.3.2.2 Handling Display Data 19
6.3.2.3 Handling Pausing and Resuming Data Transfer 21
6.3.2.4 Handling Framebuffer Resolution Change 21
6.3.2.5 Handling Framebuffer Format Change 23
6.3.2.6 Handling Framerate Adjustment 23
6.3.2.7 Handling Framebuffer Blocking Notification 24
6.3.2.8 Handling Orientation Changes 25
6.3.2.9 Handling Content Attestation 26

6.3.3 HSML Protocol Finite State Machine 27

Annex A (informative): Authors and Contributors 28

History 29

Intellectual Property Rights

Foreword

This Technical Specification (TS) has been produced by ETSI Technical Committee Intelligent Transport Systems (ITS).

The present document is part 21 of a multi-part deliverable. Full details of the entire series can be found in part 1 [i.1].

Modal verbs terminology

In the present document "**shall**", "**shall not**", "**should**", "**should not**", "**may**", "**need not**", "**will**", "**will not**", "**can**" and "**cannot**" are to be interpreted as described in clause 3.2 of the ETSI Drafting Rules (Verbal forms for the expression of provisions).

"**must**" and "**must not**" are **NOT** allowed in ETSI deliverables except when used in direct citation.

1 Scope

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

The present document 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.

2 References

2.1 Normative references

References are either specific (identified by date of publication and/or edition number or version number) or non-specific. For specific references, only the cited version applies. For non-specific references, the latest version of the referenced document (including any amendments) applies.

Referenced documents which are not found to be publicly available in the expected location might be found at https://docbox.etsi.org/Reference.

NOTE: While any hyperlinks included in this clause were valid at the time of publication, ETSI cannot guarantee their long-term validity.

The following referenced documents are necessary for the application of the present document.

[1] ETSI TS 103 544-3 (V1.3.1): "Publicly Available Specification (PAS); Intelligent Transport Systems (ITS); MirrorLink®; Part 3: Audio".

[2] ETSI TS 103 544-2 (V1.3.1): "Publicly Available Specification (PAS); Intelligent Transport Systems (ITS); MirrorLink®; Part 2: Virtual Network Computing (VNC) based Display and Control".

[3] USB IF: "Universal Serial Bus Specification", Revision 2.0, April 27, 2000.

NOTE: Available at http://sdphca.ucsd.edu/lab_equip_manuals/usb_20.pdf.

[4] IETF RFC 4122: "A Universally Unique Identifier (UUID) URN Namespace", July 2005.

NOTE: Available at http://www.ietf.org/rfc/rfc4122.txt.

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

NOTE: Available at https://usb.org/sites/default/files/CDC1.2_WMC1.1_012011.zip - File ECM120.pdf.

2.2 Informative references

References are either specific (identified by date of publication and/or edition number or version number) or non-specific. For specific references, only the cited version applies. For non-specific references, the latest version of the referenced document (including any amendments) applies.

NOTE: While any hyperlinks included in this clause were valid at the time of publication, ETSI cannot guarantee their long-term validity.

The following referenced documents are not necessary for the application of the present document but they assist the user with regard to a particular subject area.

[i.1] ETSI TS 103 544-1 (V1.3.1): "Publicly Available Specification (PAS); Intelligent Transport Systems (ITS); MirrorLink®; Part 1: Connectivity".

3 Definition of terms, symbols and abbreviations

3.1 Terms

Void.

3.2 Symbols

Void.

3.3 Abbreviations

For the purposes of the present document, the following abbreviations apply:

ARGB	Alpha-Red-Green-Blue
CCC	Car Connectivity Consortium
HSML	High Speed Media Link
IN	INput
PC	Personal Computer
RGB	Red-Green-Blue
RLE	(Scan-line based) Run Length Encoding
UI	User Interface
UPnP	Universal Plug and Play
USB	Universal Serial Bus
UUID	Universally Unique IDentifier
VNC	Virtual Network Computing
XML	eXtensible Markup Language

4 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, the present document does not limit the usage scenarios only on mobile phones and automotive head units. The HSML can be applied to any device that conforms to the present document. As shown in Figure 1.

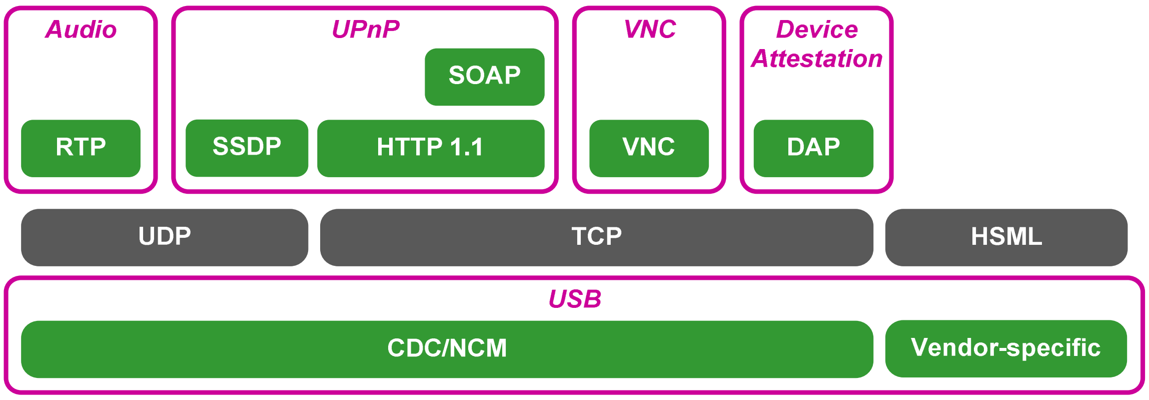

Figure 1: HSML Protocol Stack

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.

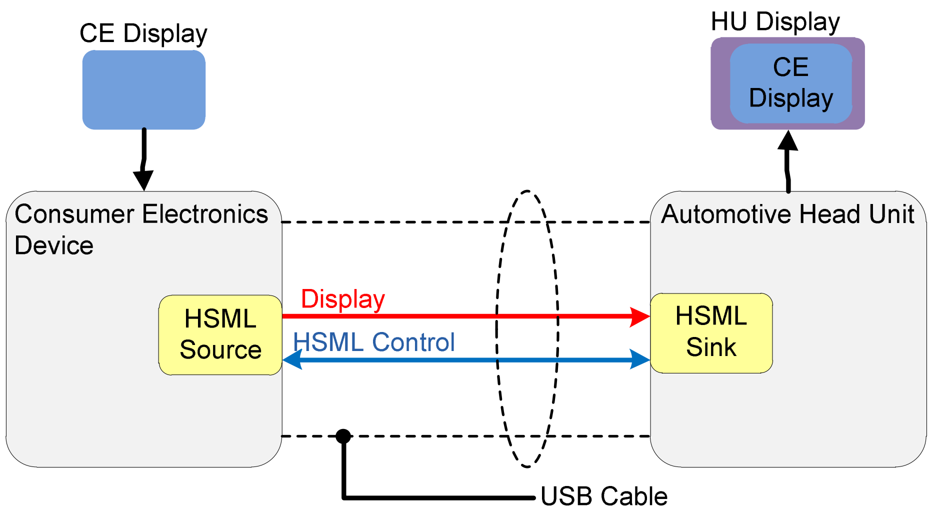

Figure 2: HSML Overview

The audio and additional display control mechanisms are handled by MirrorLink requirements defined in [1] and [2]. Therefore, any device that wants to comply with the present document shall implement MirrorLink as well.

The MirrorLink Client, providing HSML functionality, shall implement the HSML sink functionality.

The MirrorLink Server, providing HSML functionality, shall implement the HSML source functionality.

5 HSML USB Architecture

5.1 General

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 clause to refer to HSML source and HSML sink respectively because this clause mainly describes HSML in the context of USB.

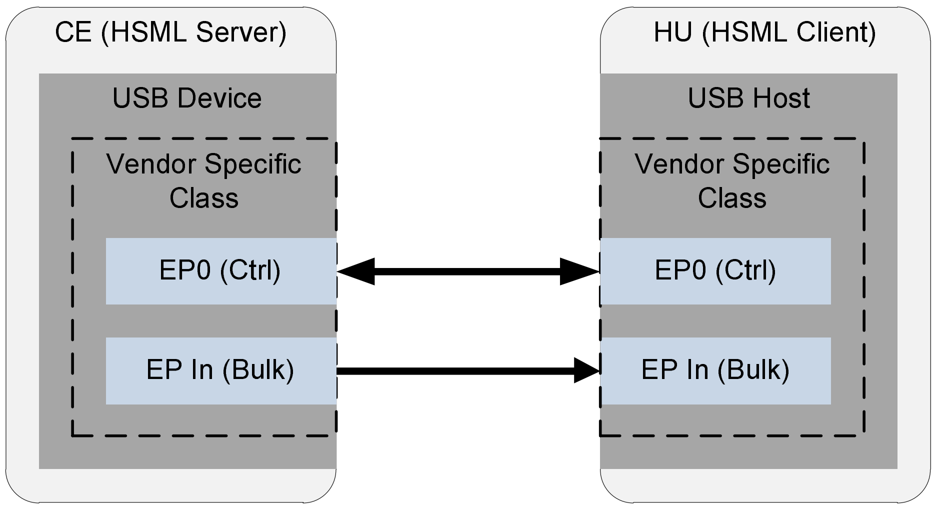

Figure 3: HSML USB Architecture

5.2 Functional Characteristics

5.2.1 General

The MirrorLink Server shall support at least two functions: one is HSML and the other is CDC/NCM which is compliant with HSML. The MirrorLink Server shall 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.

5.2.2 Interface

The HSML interface shall be one of several interfaces the MirrorLink Server has in order to conform to the present document.

5.2.3 Endpoints

5.2.3.1 General

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

5.2.3.2 Default

The default endpoint uses the control transfers as defined in the USB specification [3]. All the standard and vendor-specific requests are transmitted through this endpoint. The endpoint number shall be zero (0).

5.2.3.3 Framebuffer

This endpoint is used to receive the framebuffer data from the device. This endpoint shall use bulk transfers and the direction shall be IN. The maximum packet size for USB 2.0 shall be 512 bytes and for USB 3.0 shall be 1 024 bytes.

5.3 Vendor-Specific Codes

Table 1 defines the interface class, subclass and protocol used in the HSML interface descriptor.

Table 1: Vendor-Specific Codes

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

To comply with the present document, 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 [3].

5.4 Interface Descriptor

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

Table 2: HSML Interface Descriptor

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 shall 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

Standard USB interface descriptor is defined in [3].

5.5 Endpoint Descriptors

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

Table 3: HSML Bulk IN Endpoint Descriptor

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 shall be 512 and for USB 3.0, this value shall be 1 024)
6	bInterval	1	Number	Service interval. (not used)

Standard endpoint descriptor is defined in [3].

5.6 HSML Requests

5.6.1 General

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

Table 4: HSML Request List

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.

Request	Code	Mandatory	Description
StartFramebufferTransmission	0x43	Yes	Request the device to start sending framebuffer via the Bulk IN endpoint.
PauseFramebufferTransmission	0x44	Yes	Request the device to pause the framebuffer transmission if it's in streaming mode.
StopFramebufferTransmission	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 shall not send framebuffer updates at a rate beyond the specified value.
GetIdentifier	0x47	Yes	Request the device to return a unique identifier.

The request format is defined in [3], clause 9.

Based on [3], all HSML requests shall use little endian for any value 16-bit, 32-bit or 64-bit value.

5.6.2 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.

Table 5: GetVersion Request

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

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

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

5.6.3 GetParameters

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

Table 6: GetParameters Request

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

Below is the HSML parameter structure.

Table 7: 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. (it shall 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 **(deprecated)** Bit 2: 16-bit RGB 565 (mandatory for HSML source) Bit 3: 16-bit RGB 555 **(deprecated)** Bit 4: 16-bit RGB 444 **(deprecated)** Bit 5: 16-bit RGB 343 **(deprecated)** Bit 6 to 31: Reserved. (it shall be all zeroes)
12	bmEncodingSupported	4	Bitmap	Bit 0: RAW data (Mandatory) Bit 1: RLE (Run Length Encoding) as defined in [2]. Bit 2 to 31: Reserved. (it shall be all zeros)

If the endianness of framebuffer data is big endian, the device shall set bit 0 of *bmCapabilities* field to 1. Otherwise, it shall 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 shall set bit 1 of *bmCapabilities* field to 1.

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

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

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

5.6.4 SetParameters

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

Table 8: SetParameters Request

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

Below is the HSML configuration structure.

Table 9: 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. (it shall be all zeroes)
4	bPixelFormat	1	Number	0: 32-bit ARGB 888 1: 24-bit RGB888 **(deprecated)** 2: 16-bit RGB 565 3: 16-bit RGB 555 **(deprecated)** 4: 16-bit RGB 444 **(deprecated)** 5: 16-bit RGB 343 **(deprecated)** 6 to 31: Reserved. 32 to 255: Undefined
5	bPadding	1	0	Padding
6	wPadding	2	0	Padding
8	bmEncodingSupported	4	Bitmap	Bit 0: RAW (Mandatory) Bit 1: RLE (Run Length Encoding) Bit 2 to 31: Reserved. (it shall be all zeroes)

The host shall set bit 0 of *bmCapabilities* field to 1, if the endianness of its framebuffer data is big endian. Otherwise, it shall set this bit to 0 to indicate that its native framebuffer is in little endian. The device shall 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 shall select only colour 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 shall respond with a STALL if it cannot handle the configuration from the host.

The host shall not send this request without sending the *GetParameters* request first.

5.6.5 StartFramebufferTransmission

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

Table 10: StartFramebufferTransmission Request

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

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 shall send a framebuffer to the host. This mode saves the bandwidth on expense of slightly increased latency.

5.6.6 PauseFramebufferTransmission

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

Table 11: PauseFramebufferTransmission Request

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

The device shall stop any pending framebuffer transmissions when receiving this request but maintain all configurations and the host shall 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 shall not be sent at the on-demand mode.

5.6.7 StopFramebufferTransmission

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

Table 12: StopFramebufferTransmission Request

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

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

5.6.8 SetMaxFrameRate

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

Table 13: SetMaxFrameRate Request

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

The frame rate is defined as 1 s 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 ms or more.

The device shall 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.

5.6.9 GetIdentifier

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

Table 14: GetIdentifier

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

The device shall return a Universally Unique IDentifier (UUID) version 4, as defined in [4], 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)
```

6 HSML Framebuffer Transmission Protocol

6.1 General

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.

6.2 Managing an HSML Connection

6.2.1 Identifying Remote Applications

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

During a VNC session the MirrorLink Client shall list HSML pseudo encoding (-527) within VNC *SetEncodings* message, to indicate the support of HSML. If MirrorLink Server supports the HSML as well, it shall send VNC *FramebufferUpdate* messages with Context Information (-524) and HSML pseudo encoding (-527) in response to *FramebufferUpdateRequest* messages from the MirrorLink Client.

6.2.2 Establishing the HSML Connection

A VNC connection shall be established from the MirrorLink Client prior the HSML connection as defined in [2]. The HSML connection shall be established after VNC initialization phase as defined in clause 6.3.1.

6.2.3 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 [2].

The MirrorLink Client shall terminate the HSML connection by sending a *StopFramebufferTransmission* request, as specified in clause 5.6.7, when the MirrorLink Client is in streaming mode, as shown in Figure 4, and shall not send any further *StartFramebufferTransmission* requests, as specified in clause 5.6.5.

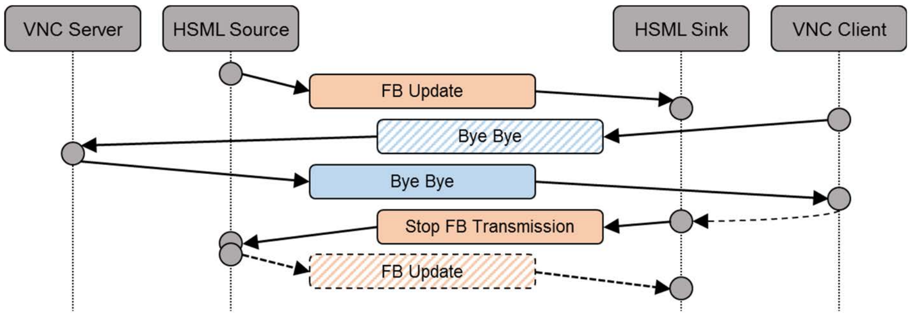

Figure 4: Message Sequence for Intentionally Terminating HSML (Streaming Mode)

If the MirrorLink Client decides to re-establish the VNC session, it shall follow the steps given in clause 6.2.2. The MirrorLink Server shall keep the USB HSML

6.2.4 Unintentionally Terminating the HSML Connection

The HSML connection may be disconnected unintentionally in case of any error conditions. The recognizing entity shall then terminate the VNC connection by sending a VNC *ByeBye* message as specified in [2].

The VNC connection may be disconnected unintentionally in case of any error conditions. The MirrorLink Client shall then terminate the HSML connection, following the steps defined in clause 6.2.3.

If the MirrorLink Client decides to re-establish the VNC session, it shall follow the steps given in clause 6.2.2.

6.3 HSML Protocol Phases

6.3.1 Initialization Phase

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

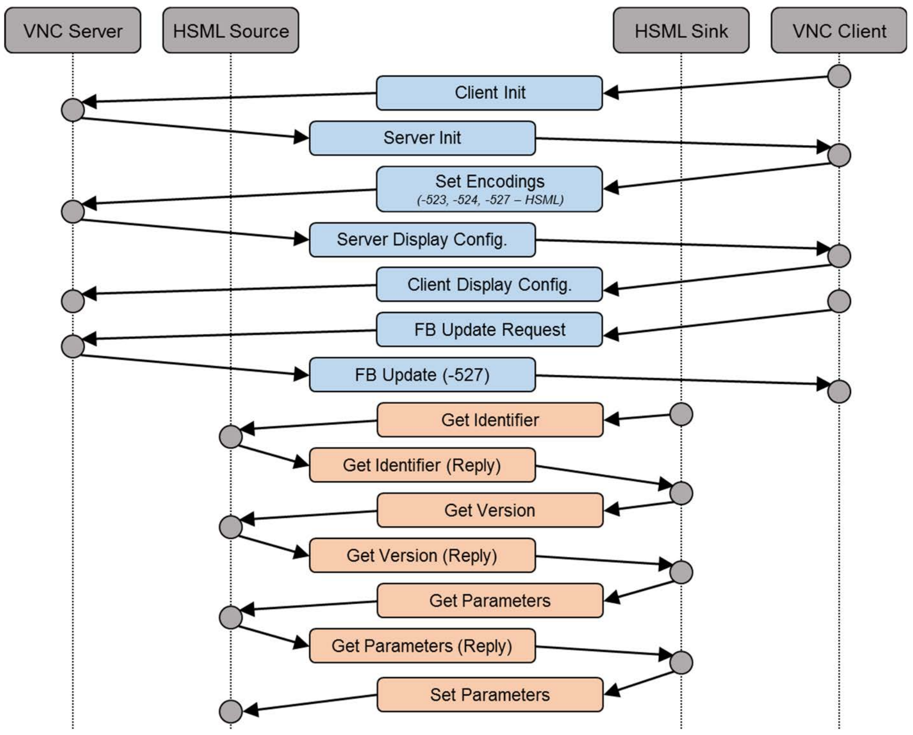

Figure 5: Message Sequence for HSML Initialization Phase

After VNC handshaking phase as defined in [2] and exchanging *ClientInit* and *ServerInit* messages, the MirrorLink Client shall 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 shall 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 shall use the *GetIdentifier* request to distinguish between them. The MirrorLink Server shall 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 shall use the returned values to select the configuration which HSML sink can support. Once selected the preferred configuration, HSML sink shall send *SetParameters* request to HSML source. HSML source shall adopt the configuration from HSML sink.

The HSML resolution *wWidth* and *wHeight* as provided via *GetParameters* request shall follow the resolution negotiated within the VNC context, i.e. as soon as the Desktop Size Pseudo Encoding has been sent.

The HSML initialization shall be completed in 1 s.

6.3.2 Transmission Phase

6.3.2.1 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.

Table 15: HSML Pseudo Encoding

# 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

After receiving the initial *FramebufferUpdate* message and in response to any VNC *FramebufferUpdate* messages, the MirrorLink Client shall immediately send a VNC *FramebufferUpdateRequest* message. The VNC *FramebufferUpdateRequest* shall 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 shall 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 Server shall not send any framebuffer pixel data within *FramebufferUpdate* messages and the MirrorLink Client shall either ignore any framebuffer pixel data within *FramebufferUpdate* messages or intentionally terminate the VNC session.

Depending on incremental flag within *FramebufferUpdateRequest* message, the MirrorLink Server:

- Shall 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.
- Shall 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 100 ms.

The MirrorLink Client and Server shall also follow additional requirements defined in [2].

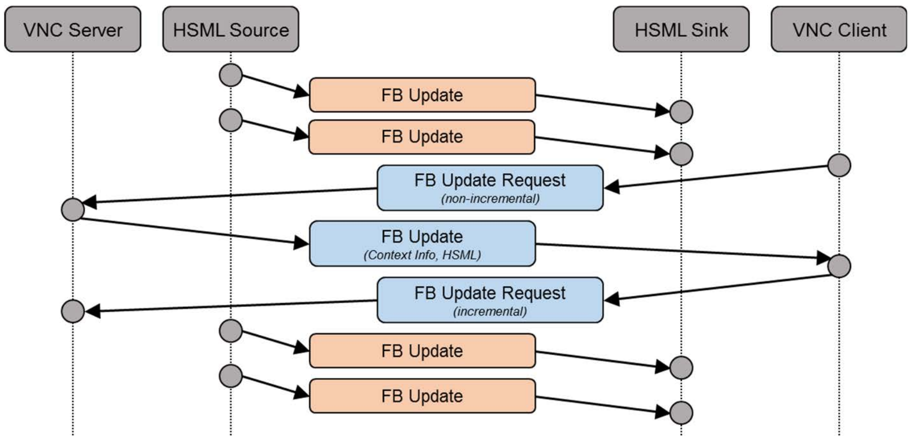

Figure 6: Message Sequence for on-Demand Context Information

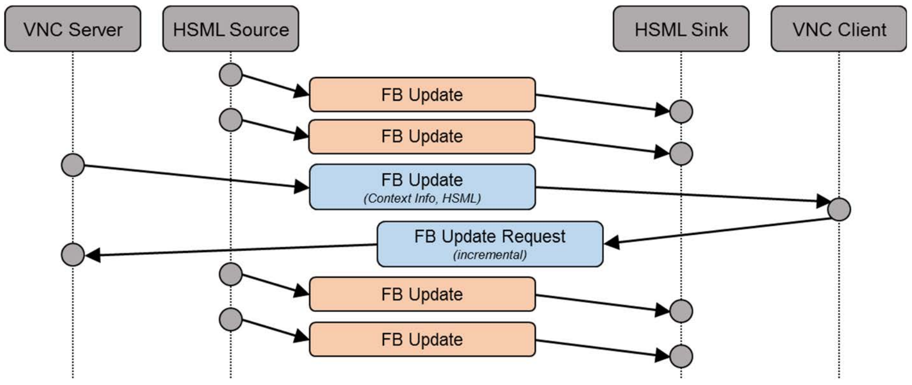

Figure 7: Message Sequence for on-Change Context Information

6.3.2.2 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 shall support both Streaming Mode and On-Demand Mode.

HSML source shall send framebuffer continuously in the streaming mode, as shown in Figure 8.

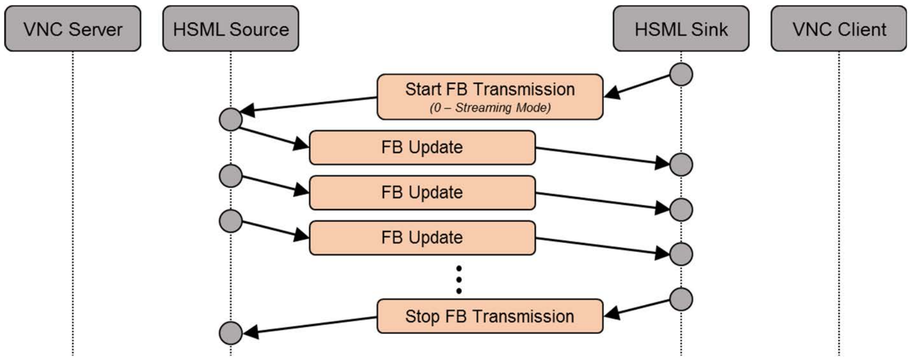

Figure 8: Message Sequence for Transmission Phase (Streaming Mode)

In on demand mode the HSML source shall only send framebuffer if requested by the HSML sink regardless of whether the *FBUpdateOnChange* in the *SetParameters* is set, as shown in Figure 9.

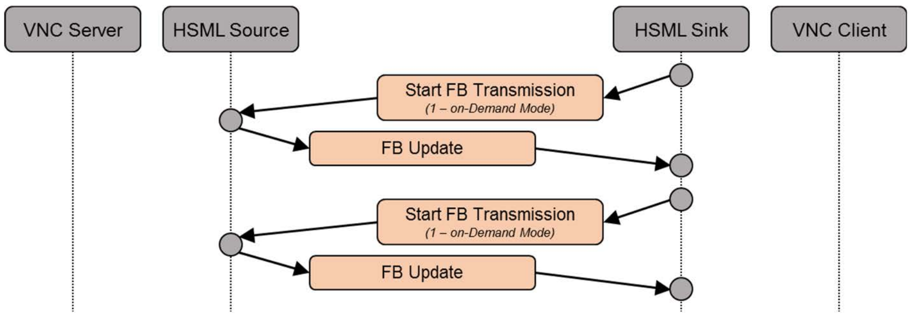

Figure 9: Message Sequence for Transmission Phase (On-Demand Mode)

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

Table 16: Framebuffer Header

# 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: RLE 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

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 shall be recorded at the time of the frame being captured and the unit shall be ms. This value shall 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 shall follow the negotiated framebuffer resolution within the VNC context. The pixel format and encoding fields shall be equivalent to those of *SetParameters* request.

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

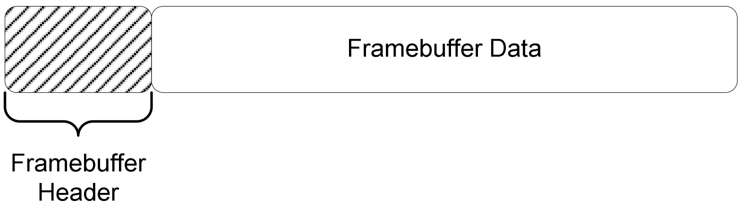

Figure 10: Bulk Transfer Framebuffer Format

Framebuffer Data shall include full screen information no matter if it is raw data or compressed data. Framebuffer Data shall be orientated and rotated as negotiated within the VNC context. Framebuffer Data shall be formatted according the *PixelFormat* set via *SetParameter* request and as defined in [2].

The HSML source shall use a USB short packet mechanism to implement segment delineation as specified in [5].When a framebuffer transfer (Header + Data) spans *N* USB packets, the first packet through packet *N-1* shall 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 [5].

6.3.2.3 Handling Pausing and Resuming Data Transfer

The HSML source can pause and later on resume the HSML streaming of framebuffer data at any point in time.

The message sequence, the HSML source and sink shall follow for pausing and resuming the framebuffer stream in Streaming Mode, is shown in Figure 11.

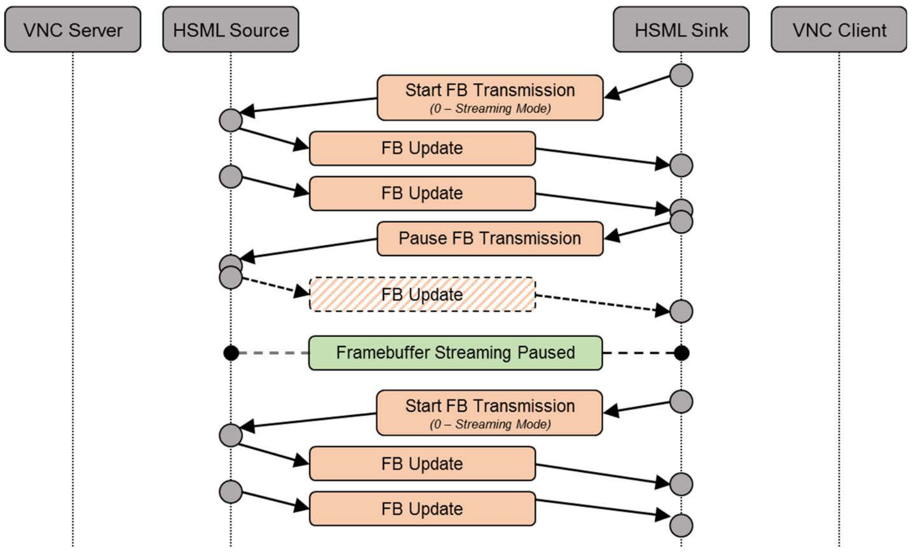

Figure 11: Message Sequence for Pausing and Resuming the Data Transfer in Streaming Mode

In case the HSML Sink has enabled *FBUpdateOnChange*, the HSML Source shall send an initial framebuffer, when resuming the framebuffer streaming. I.e. the HSML Source shall not wait until an actual change of the framebuffer. This allows the HSML Sink to not keep the last framebuffer content, before pausing the stream.

The message sequence, the HSML source and sink shall follow for pausing and resuming the framebuffer stream in on-Demand Mode, is shown in Figure 12.

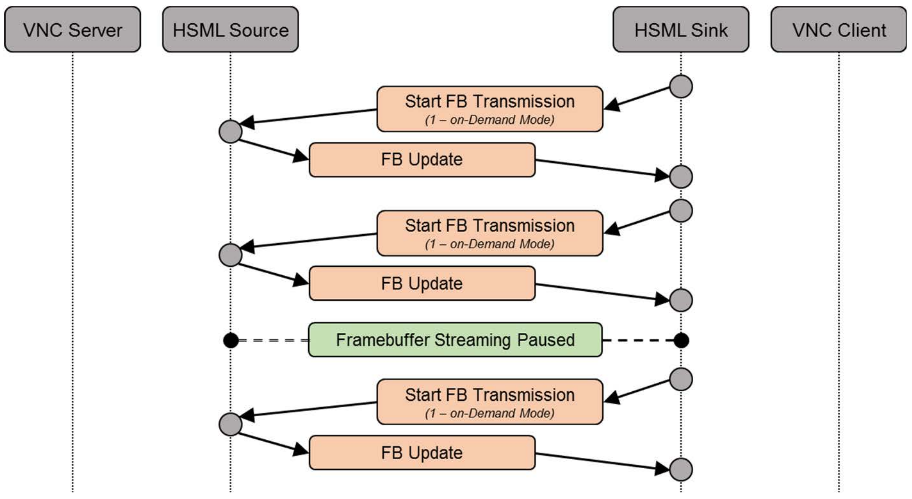

Figure 12: Message Sequence for Pausing and Resuming the Data Transfer in On-Demand Mode

6.3.2.4 Handling Framebuffer Resolution Change

The MirrorLink Client and Server can change framebuffer resolution as defined in [2].

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

The message sequence the HSML source and sink shall follow is shown in Figure 13, in case the resolution is changed during run-time, i.e. after the HSML transfer has started. The HSML source shall stop sending HSML packets immediately after having received the *StopFramebufferTransmission* message, but it shall still finish an already ongoing transfer of an HSML frame. Therefore, the HSML Sink shall be able to receive further HSML packets, containing the old size, after sending the *StopFramebufferTransmission* message.

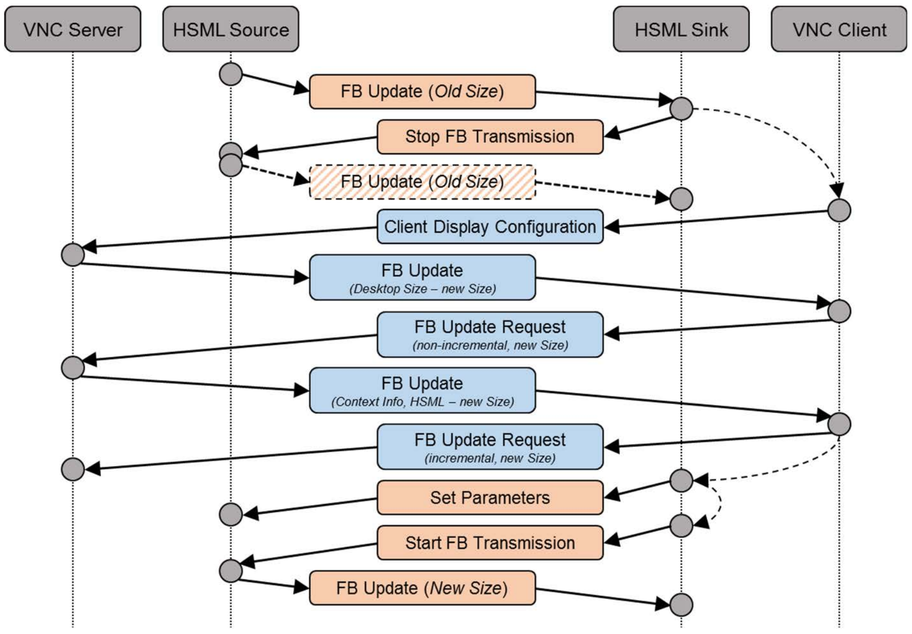
VNC Server
HSML Source
HSML Sink
VNC Client
FB Update (Old Size)
Stop FB Transmission
FB Update (Old Size)
Client Display Configuration
FB Update (Desktop Size – new Size)
FB Update Request (non-incremental, new Size)
FB Update (Context Info, HSML – new Size)
FB Update Request (incremental, new Size)
Set Parameters
Start FB Transmission
FB Update (New Size)

Figure 13: Message Sequence for successful run-time Framebuffer Resolution Change

On reception of the *ClientDisplayConfiguration* message, the VNC Server shall 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 shall send a *FramebufferUpdate* message containing a `DesktopSize` pseudo encoding rectangle with the old framebuffer resolution, as shown in Figure 14.

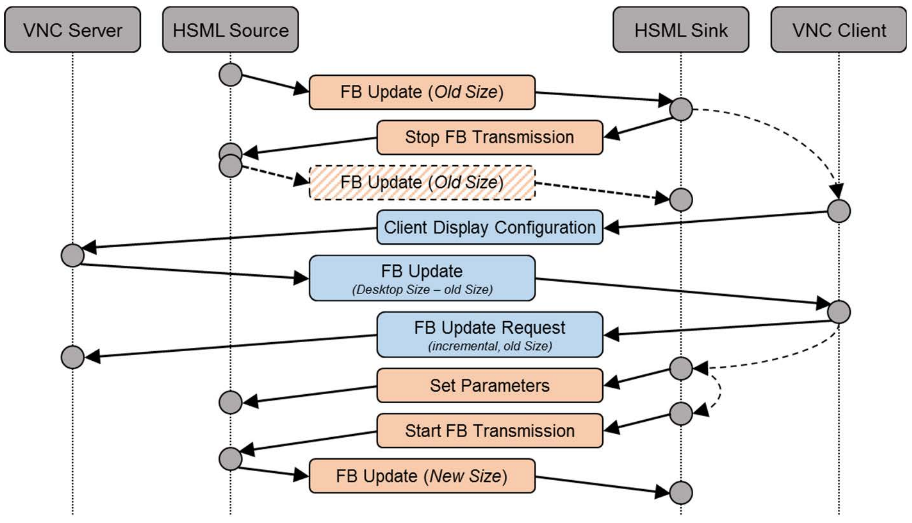

Figure 14: Message Sequence for unsuccessful run-time Framebuffer Resolution Change

In case the MirrorLink Server wants to change the framebuffer resolution during the initial handshake, the following very similar message sequence shall be followed, as given in Figure 15.

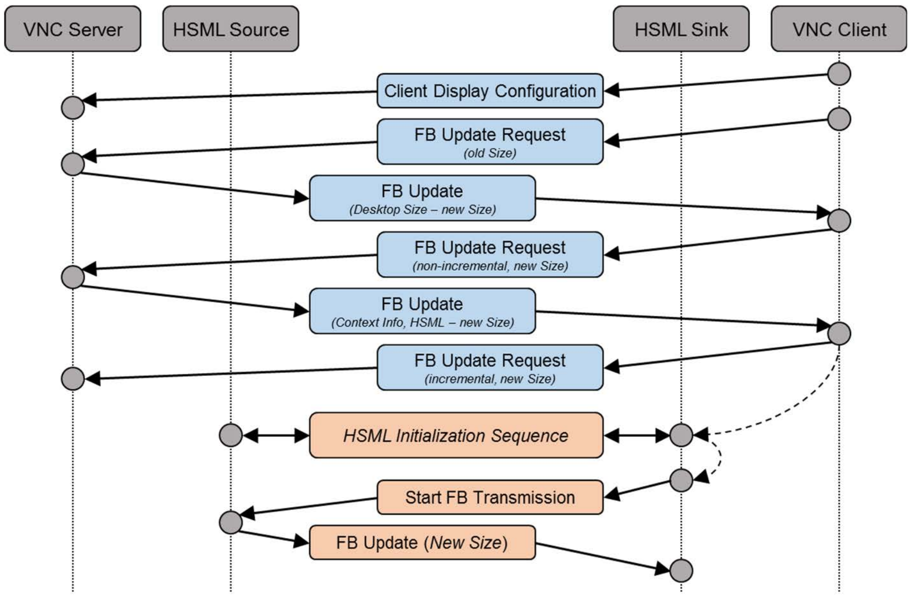
VNC Server
HSML Source
HSML Sink
VNC Client
Client Display Configuration
FB Update Request (old Size)
FB Update (Desktop Size – new Size)
FB Update Request (non-incremental, new Size)
FB Update (Context Info, HSML – new Size)
FB Update Request (incremental, new Size)
HSML Initialization Sequence
Start FB Transmission
FB Update (New Size)

Figure 15: Message Sequence for initial Framebuffer Resolution Change

The MirrorLink Server and Client shall both follow the framebuffer scaling and aspect ratio requirements defined in [2].

6.3.2.5 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 shall follow the procedure in Figure 16.

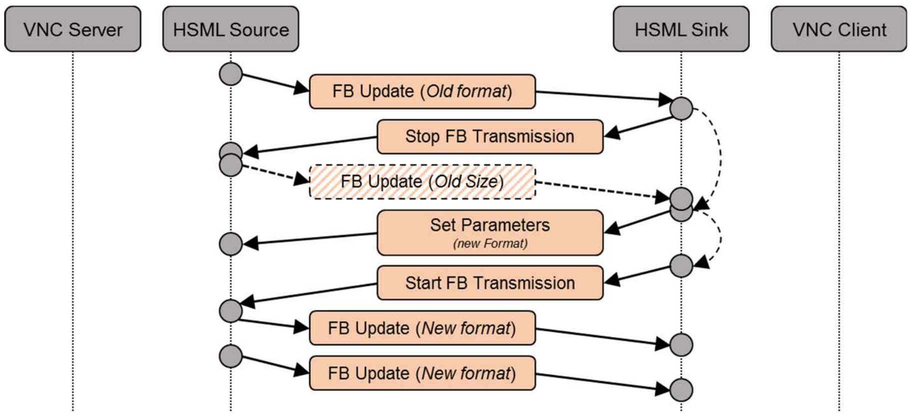

Figure 16: Message Sequence for Framebuffer Format Change in Streaming Mode

After changing the pixel format, the HSML sink shall 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 shall 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.

6.3.2.6 Handling Framerate Adjustment

The HSML sink can limit the maximum frame rate that the HSML source can send by following the procedure in Figure 17.

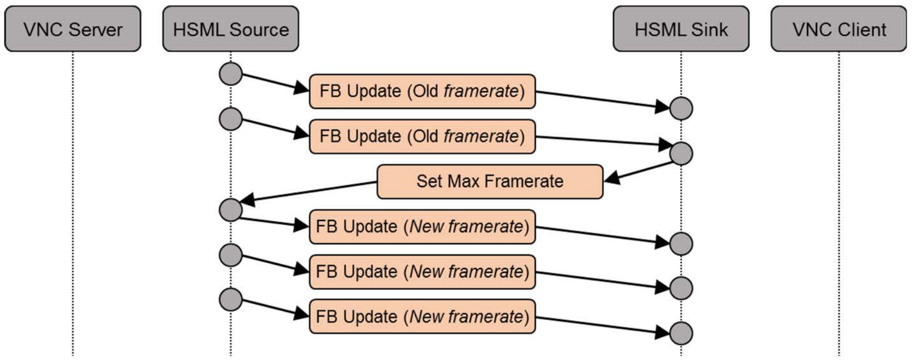
VNC Server
HSML Source
HSML Sink
VNC Client
FB Update (Old *framerate*)
FB Update (Old *framerate*)
Set Max Framerate
FB Update (*New framerate*)
FB Update (*New framerate*)
FB Update (*New framerate*)

Figure 17: Message Sequence for Framerate Adjustment

6.3.2.7 Handling Framebuffer Blocking Notification

The HSML source shall be compliant with the Framebuffer Blocking Notification mechanism in [2], the *FramebufferBlocking* Notification is sent by VNC Client still and the example flow is shown in Figure 18 for the example situation, when the MirrorLink Client is blocking the framebuffer after receiving new Context Information.

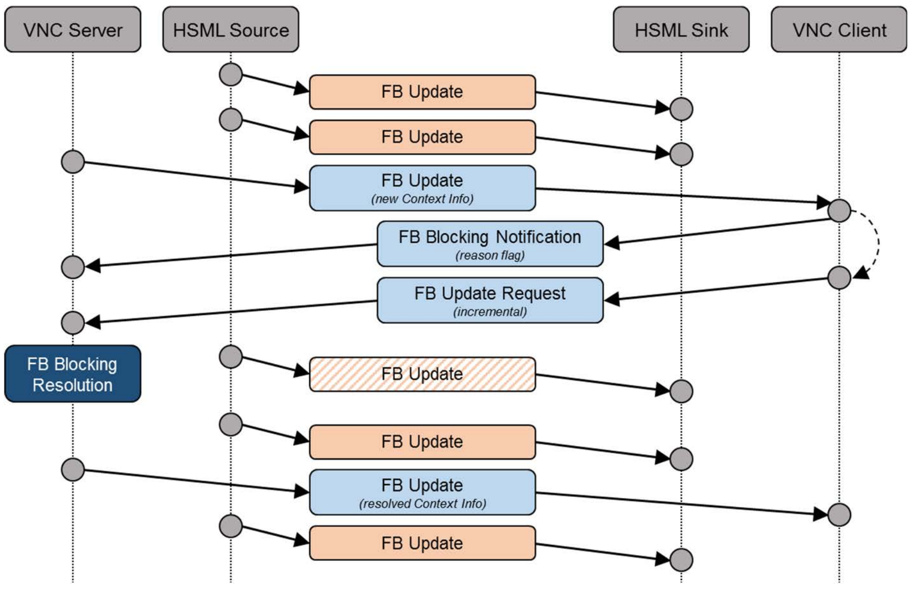

Figure 18: Message Sequence for Framebuffer Blocking on new Framebuffer Context Info

HSML source shall follow one of the actions defined in [2] 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 shall pause the HSML transfer, sending a *PauseFramebufferTransmission* message in streaming mode, as shown in Figure 19. In case of on-demand mode, the HSML Sink shall stop sending *FramebufferUpdateRequest* messages.

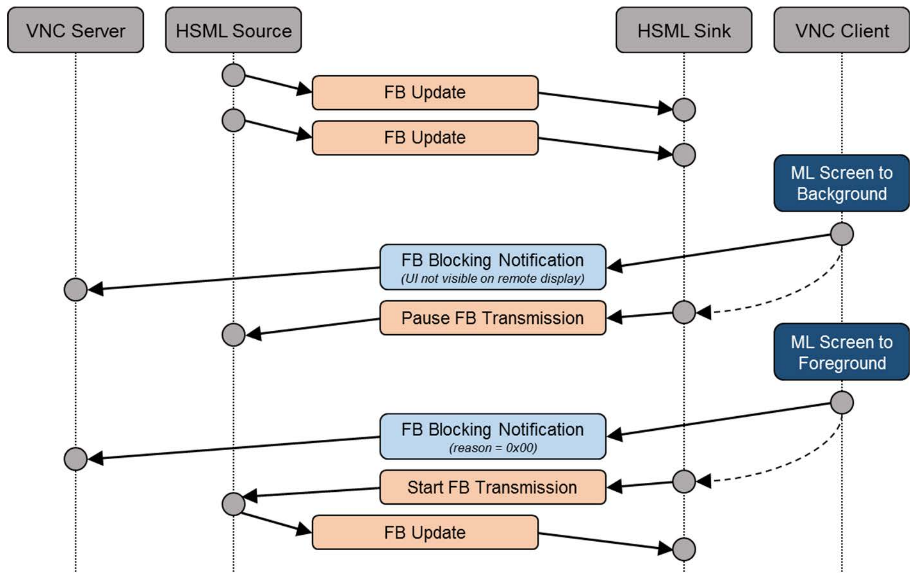

Figure 19: Message Sequence moving the MirrorLink Screen to the Background

The HSML Sink shall 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.

6.3.2.8 Handling Orientation Changes

The HSML Sink shall provide the framebuffer content in landscape orientation. Portrait orientation is not supported by the present document and shall 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 an application on the MirrorLink Server attempts to change the orientation to Portrait and the application cannot be switched back into Landscape, e.g. the Portrait-only Android native home screen application, the MirrorLink Server shall not follow the orientation request and send a Portrait framebuffer via HSML. Instead, the MirrorLink Server should stop sending further HSML frames (if in streaming mode) and send VNC *ContextInformationPseudoEncoding* message, with the application category set to `"Switch to MirrorLink Client native UI"`. The framebuffer dimensions within that message shall match the original landscape orientation. The MirrorLink Client shall stop the HSML framebuffer transmission and switch to its native UI, as shown in below Figure 20.

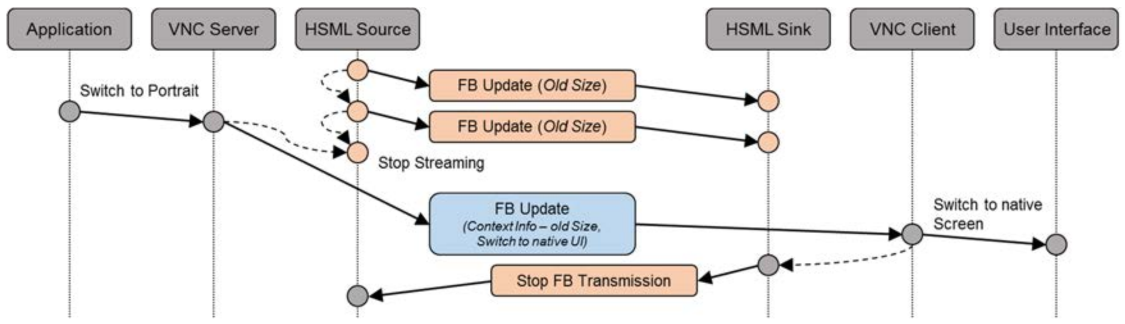

Figure 20: Recommended Message Sequence to respond to Applications switching to Portrait Mode

In case the MirrorLink Server does not use the mechanism described in Figure 20, it shall switch either to a certified Home Screen or to another certified application, which are certified for the given driving mode and locale and which displays a landscape user-interface.

In case the MirrorLink Client nevertheless attempts to change the orientation, the MirrorLink Server shall reject the change, using the message exchange shown in Figure 21.

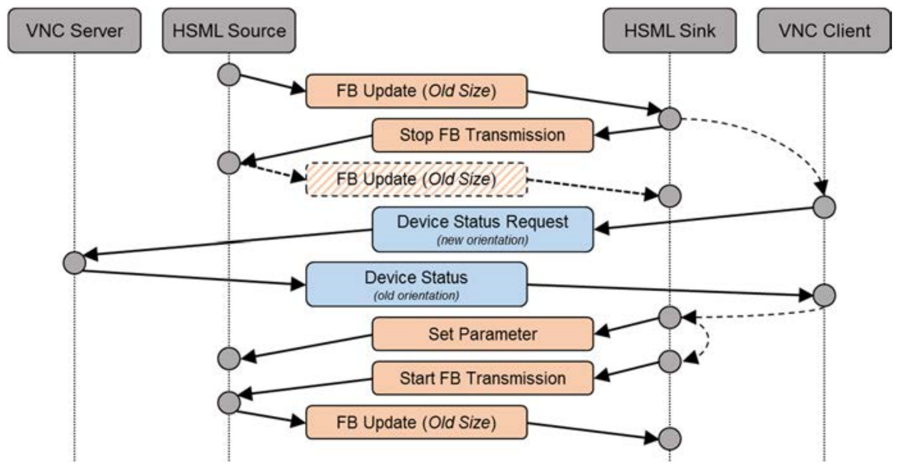

Figure 21: Message Sequence rejecting Orientation Change from MirrorLink Client

In case the MirrorLink Server nevertheless attempts to change the orientation, the MirrorLink Client shall reject the change, using the message exchange shown in Figure 22.

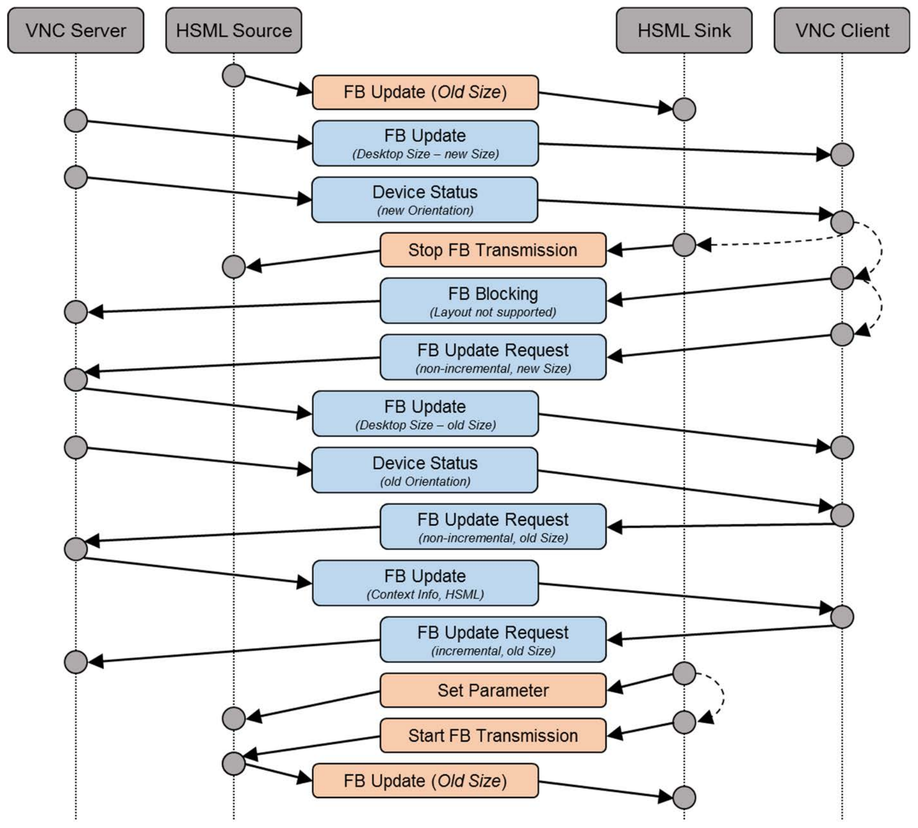

Figure 22: Message Sequence rejecting Orientation Change from MirrorLink Server

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 22.

6.3.2.9 Handling Content Attestation

The current HSML specification only supports the attestation of the Context Information via the exchange VNC *ContentAttestation* messages. See [2] for details.

6.3.3 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 shall return to an uninitialized state, i.e. the HSML sink shall follow the HSML initialization phase, as defined in clause 6.2, when a HSML connection is setup the next time.

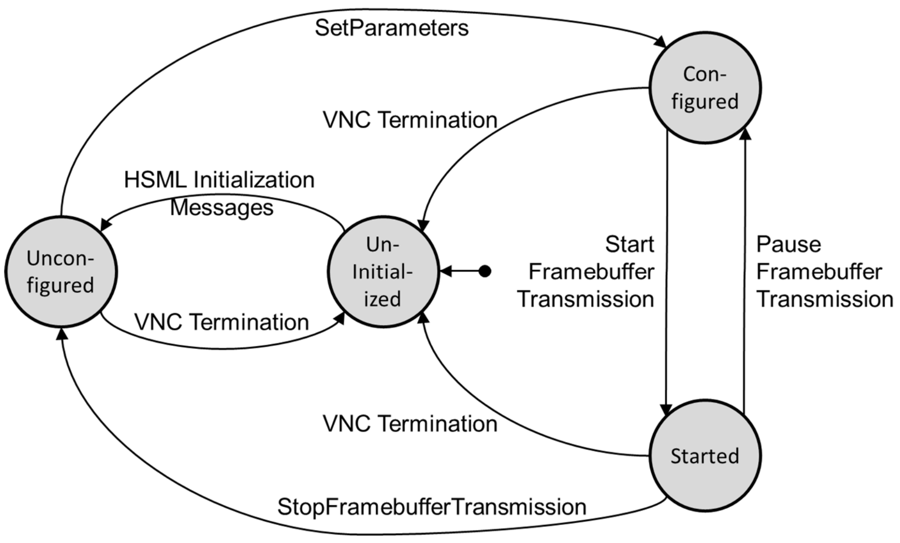
SetParameters
Con-figured
VNC Termination
HSML Initialization Messages
Uncon-figured
Un-Initial-ized
Start Framebuffer Transmission
Pause Framebuffer Transmission
VNC Termination
VNC Termination
Started
StopFramebufferTransmission

Figure 23: HSML Protocol Finite State Machine Diagram

Annex A (informative): Authors and Contributors

The following people have contributed to the present document:

Rapporteur:	Dr. Jörg Brakensiek, E-Qualus (for Car Connectivity Consortium LLC)
Other contributors:	Matthias Benesch, Mercedes-Benz Research & Development North America
	Joey Chien, HTC Corp.
	Seubert Christopher, Carmeq (for Volkswagen AG)
	Gautier Falconnet, PSA
	Dennis Fernahl, Carmeq (for Volkswagen AG)
	Alexander Kirschner, jambit GmbH
	Matthias Langhammer, jambit GmbH
	Joe Wei, HTC Corp.

History

Document history		
V1.3.0	October 2017	Publication
V1.3.1	October 2019	Publication