[MS-WDHCE]:

Wi-Fi Display Protocol: Hardware Cursor Extension

Intellectual Property Rights Notice for Open Specifications Documentation

- **Technical Documentation.** Microsoft publishes Open Specifications documentation for protocols, file formats, languages, standards as well as overviews of the interaction among each of these technologies.
- **Copyrights**. This documentation is covered by Microsoft copyrights. Regardless of any other terms that are contained in the terms of use for the Microsoft website that hosts this documentation, you may make copies of it in order to develop implementations of the technologies described in the Open Specifications and may distribute portions of it in your implementations using these technologies or your documentation as necessary to properly document the implementation. You may also distribute in your implementation, with or without modification, any schema, IDL's, or code samples that are included in the documentation. This permission also applies to any documents that are referenced in the Open Specifications.
- **No Trade Secrets**. Microsoft does not claim any trade secret rights in this documentation.
- **Patents**. Microsoft has patents that may cover your implementations of the technologies described in the Open Specifications. Neither this notice nor Microsoft's delivery of the documentation grants any licenses under those or any other Microsoft patents. However, a given Open Specification may be covered by Microsoft Open Specification Promise or the Community Promise. If you would prefer a written license, or if the technologies described in the Open Specifications are not covered by the Open Specifications Promise or Community Promise, as applicable, patent licenses are available by contacting iplg@microsoft.com.
- **Trademarks**. The names of companies and products contained in this documentation may be covered by trademarks or similar intellectual property rights. This notice does not grant any licenses under those rights. For a list of Microsoft trademarks, visit www.microsoft.com/trademarks.
- **Fictitious Names**. The example companies, organizations, products, domain names, e-mail addresses, logos, people, places, and events depicted in this documentation are fictitious. No association with any real company, organization, product, domain name, email address, logo, person, place, or event is intended or should be inferred.

Reservation of Rights. All other rights are reserved, and this notice does not grant any rights other than specifically described above, whether by implication, estoppel, or otherwise.

Tools. The Open Specifications do not require the use of Microsoft programming tools or programming environments in order for you to develop an implementation. If you have access to Microsoft programming tools and environments you are free to take advantage of them. Certain Open Specifications are intended for use in conjunction with publicly available standard specifications and network programming art, and assumes that the reader either is familiar with the aforementioned material or has immediate access to it.

Revision Summary

Date	Revision History	Revision Class	Comments
6/30/2015	1.0	New	Released new document.
10/16/2015	2.0	Major	Significantly changed the technical content.

Table of Contents

1 Introduction

This documentation specifies an extension to the Miracast v1.1 Wireless Display protocol [WF-DTS1.1].

Sections 1.8, 2, and 3 of this specification are normative and can contain the terms MAY, SHOULD, MUST, MUST NOT, and SHOULD NOT as defined in [RFC2119]. Sections 1.5 and 1.9 are also normative but do not contain those terms. All other sections and examples in this specification are informative.

1.1 Glossary

MAY, SHOULD, MUST, SHOULD NOT, MUST NOT: These terms (in all caps) are used as defined in [RFC2119]. All statements of optional behavior use either MAY, SHOULD, or SHOULD NOT.

1.2 References

Links to a document in the Microsoft Open Specifications library point to the correct section in the most recently published version of the referenced document. However, because individual documents in the library are not updated at the same time, the section numbers in the documents may not match. You can confirm the correct section numbering by checking the Errata.

1.2.1 Normative References

We conduct frequent surveys of the normative references to assure their continued availability. If you have any issue with finding a normative reference, please contact dochelp@microsoft.com. We will assist you in finding the relevant information.

[MS-ERREF] Microsoft Corporation, "Windows Error Codes".

[RFC2119] Bradner, S., "Key words for use in RFCs to Indicate Requirement Levels", BCP 14, RFC 2119, March 1997, http://www.rfc-editor.org/rfc/rfc2119.txt

[WF-DTS1.1] Wi-Fi Alliance, "Wi-Fi Display Technical Specification v1.1", April 2014, https://www.wi-fi.org/wi-fi-display-technical-specification-v11

Note There is a charge to download the specification.

1.2.2 Informative References

None.

1.3 Overview

The Miracast v1.1 protocol [WF-DTS1.1] only supports a single stream from the source to the sink, this means that the mouse image has to be part of the desktop image that is encoded and streamed to the sink. This binds the mouse cursor refresh rate and latency to that of the desktop image. Currently the Miracast stream is typically 30 Hz with a screen to screen latency of around 200ms. As a reference point the screen to screen latency over HDMI wire is around 32ms. We know from usability studies that mouse cursor latency is more noticeable to the user than the rest of the desktop. So it would be desirable to decouple the mouse cursor from the rest of the desktop image.

It is possible to send the mouse cursor information in a separate stream to the sink so the update rate of the cursor can be de-coupled from the update rate of the desktop image. This would require the graphics driver to send the information and the Miracast receiver to be able to receive 2 streams and then blend the cursor image with the desktop image.

This document describes the hardware capabilities that a sink would need in order to process and display the cursor information and additional detail the protocol changes required to send this additional stream to the sink.
This document assumes that the mouse cursor and position information is sent by the source and received by the sink. Any bitmap/cursor shape information has been successfully transmitted in a lossless manner.

For the Miracast scenario, the cursor image size would typically be smaller than 48x48, even in high DPI scenarios the cursor image is expected to be less than 64x64. Ultimately the application is in control of the cursor shape (max 256x256) so the solution has to work with arbitrary cursors.

In an artificial scenario where the user is moving an animated cursor constantly, we observed a max of 100 mouse positions updates per second and 20 mouse cursor shape changes per second.

In some configurations, it is possible that a device might not support the Microsoft Hardware Cursor extension but might support the previous standard of Intel Fast Cursor. Applicable information for Intel Fast Cursor is described in the appropriate sections for devices that support both extensions.

1.4 Relationship to Other Protocols

This document describes an extension to the existing Miracast v1.1 standard for Wireless Display over WiFi Direct [WF-DTS1.1]. The protocol communicates over RTP/RTSP as described in the Miracast standard and this extension adds an additional capability query and support for side channel UDP communication between a source and receiver device during a Miracast session.

1.5 Prerequisites/Preconditions

This extension is to be implemented by a device (either source or receiver) that implements the Miracast v1.1 protocol [WF-DTS1.1]. This extension is only available when used in conjunction with other devices that support this extension, otherwise the functionality will be ignored.

A source that implements this extension sends mouse cursor information to the sink in a similar format to an OS passing cursor information to a local display driver. Some sinks cannot support the XOR operation used by monochrome and color mask cursors, and in order to allow hardware cursor functionality on those sinks, the sink capabilities include a flag noting whether the sink supports the XOR operation. If the sink does not support XOR, the source converts any XOR masks into non-XOR masks. If a sink supports this extension, then it must support the alpha color cursor image type.

With the alpha cursor color image, a 32 bits per pixel ARGB image is supplied, with the 8 bit alpha value used to blend between the RGB values in the display image and the RGB values in the cursor image. The result of the blending operation is sent to the display.

1.6 Applicability Statement

The goal for this extension is to improve the end-to-end latency of the mouse cursor when streaming over Miracast. This is a valuable improvement to the baseline Miracast protocol particularly in scenarios such as productivity or gaming where low-latency responsive user input with a mouse is a high priority. If both devices participating in a Miracast session support this extension, it is recommended for use 100% of the time.

1.7 Versioning and Capability Negotiation

Supported Transports: This protocol must be implemented on top of TCP/UDP as specified in the Miracast v1.1 specification [WF-DTS1.1].

Capability Negotiation: This extension performs explicit capability negation in the M3 capabilities message as defined in the Miracast protocol.

A new optional RTSP parameter 'microsoft_cursor' is used to query for hardware cursor support and capabilities. The following is the ABNF format for the response from the sink to the microsoft_cursor using the GET_PARAMETER command as specified in [WF-DTS1.1] section 6.2.2.

```
microsoft_cursor   = "none" / sink-cursor-caps
      ;  none'means the sink does not support this hardware cursor extension
               ;  otherwise it is assumed that the sink can support per pixel 8bit
      ;  alpha blending of hardware cursor.

sink-cursor-caps   = xor-support SP x-max SP y-max SP port

xor-support= "none" / "full"
             ;  'none' means the sink does not support XOR blending operations at
             ;  all, 'full' indicates the sink can perform the XOR blending
      ;  operations.

x-max      = 4HEXDIG
             ;  the maximum width of the cursor the sink supports in pixels, the sink
;  should support this width for all cursor formats.

y-max      = 4HEXDIG
             ;  the maximum height of the cursor the sink supports in pixels, the
             ;  sink should support this height for all cursor formats.

port       = 4HEXDIG
             ;  this is the UDP port the source should send the mouse cursor position
             ;  and shape changes to
```

In the case that a device supports Intel Fast Cursor as well, the following information also applies. A new optional RTSP parameter 'intel_fast_cursor' is used to query for Intel Fast Cursor support and capabilities. The following is the ABNF format for the response from the sink to the intel_fast_cursor using the GET_PARAMTETER command, as specified in [WF-DTS1.1] section 6.2.2.

```
intel-fast-cursor = "intel_fast_cursor:" SP "port=" fast-cursor-port
fast-cursor-port = IPPORT
```

Fast cursor ports must be a value from the port range (49152 to 65535) with the exception of older Intel WiDi devices that may only support fast cursor messages on port 1232.

Note that a sink that supports the Intel Fast Cursor extension ignores a fast cursor message if: (1) it does not conform to the ABNF rules; (2) the x parameter is equal to or greater than the width parameter; (3) the y parameter is equal to or greater than the height parameter; or (4) the sink has transmitted a UIBC packet within the previous 100ms. If a sink receives a fast cursor message containing the current-position ABNF rule, the sink displays a locally rendered cursor at the indicated location. If a sink receives a fast cursor message containing the hidden ABNF rule (see section 2.2.2), the sink does not display any locally rendered cursors. If it has been more than 100ms since a fast cursor message was received, a sink does not display any locally rendered cursors.

1.8 Vendor-Extensible Fields

This protocol uses HRESULT values as defined in [MS-ERREF] section 2.1. Vendors can define their own HRESULT values, provided they set the C bit (0x20000000) for each vendor-defined value, indicating the value is a customer code.

1.9 Standards Assignments

None.

2 Messages

2.1 Transport

As defined by the Miracast v1.1 protocol [WF-DTS1.1], this extension uses RTSP, which uses TCP at the transport layer to maintain an end-to-end connection. The capability negotiation portion of this extension occurs over RTSP.

A new RTSP parameter is defined that is part of M3 capabilities request that will query for the sink's support of hardware cursor and capabilities. If supported, the source will send mouse pointer position and shape updates to the sink over UDP, and the sink MUST decide which UDP port to use.

There will not be any acknowledgment scheme to ensure delivery of the mouse point position or shape changes, but because mouse pointer image packets are bigger and missing one greatly affects the user experience, the source will send the update multiple times to ensure the sink receives the update.

The source will expand all monochrome cursors into masked color cursors before sending to the sink.

2.2 Message Syntax

Both the mouse position and shape packets use an RTP header as follows.

bit offset	0-1	2	3	4-7	8	9-15	16-31
0	Version	P	X	CC	M	PT	Sequence Number
32	Timestamp						
64	SSRC identifier						

We define a new 'Microsoft cursor' application profile. For this protocol extension, the following values are used in the RTP header.

Field Name (Size)	Value
Version (2 bits)	2
P (1 bit)	0 (No padding)
X (1 bit)	0 (No extension)
CC (4 bits)	0 (No CSRC)
M (1 bit)	0 (No marker used)
PT (7 bits)	0 (Mouse position/shape update payload for this profile)
Sequence Number (16 bits)	Incrementing sequence number starting at zero
Timestamp (32 bits)	0 (Not used)
SSRC identifier (32 bits)	0 (Not used)

All network data is transmitted in network byte order.

2.2.1 Namespaces

None.

2.2.2 Mouse pointer position message

When the graphics driver is informed of a mouse position change, it sends a message to the sink so that the sink can update the screen location of the mouse pointer on the wireless display. The UDP messages have an RTP header (as specified in section 2.2) followed by a binary message in the following format.

Field name	Type	Details
MsgType	8 bit unsigned	The type of cursor message, valid values 0x01 Mouse cursor position update 0x02 Mouse cursor shape update 0x03 Mouse cursor shape continuation For mouse cursor position update this will be 0x01
PacketMsgSize	16 bit unsigned	The total size of this message in bytes, for mouse pointer update this is 0x0007
XPos	16 bit signed	The X position of the upper-left corner of the pointer position relative to this Miracast display, see notes below.
YPos	16 bit signed	The Y position of the upper-left corner of the pointer position relative to this Miracast display, see notes below

The mouse position update packet MUST always be in its own UDP packet and hence will always be located directly after the RTP header.

If Intel Fast Cursor is supported<1> the following variation of the mouse pointer position message will be used. One 32-bit fast cursor message will be sent for each UDP packet, formatted as follows.

```
fast-cursor-message = "fast cursor=" hidden / current position
hidden = "0:0:0:0:0"
current-position = width ":" height ":" x ":" y ":" orientation
width = 1*4(DIGIT) ; total pixel width of the host screen (e.g. 1920)
height = 1*4(DIGIT) ; total pixel height of the host screen (e.g., 1080)
x = 1*4(DIGIT) ; x coordinate of the cursor position [0..width-1]
y = 1*4(DIGIT) ; y coordinate of the cursor [0..height-1]
orientation = "0" / "90" / "180" / "270" ; degrees of display rotation
```

2.2.3 Mouse pointer shape message

When the graphics driver is given a new mouse pointer shape, it sends a mouse pointer position update to the sink. The network message contains an RTP header, as specified in section 2.2, followed by a binary message in the following format.

Note Because the cursor shape packet can be bigger than the UDP packet, we split the mouse shape data into a single start mouse shape packet and potentially multiple mouse shape continuation packets. Below is the definition of the start packet.

Field name	Type	Details
MsgType	8 bit unsigned	The type of cursor message, valid values 0x01 Mouse cursor position update 0x02 Mouse cursor shape start 0x03 Mouse cursor shape continuation For mouse cursor image update this will be 0x02
PacketMsgSize	16 bit unsigned	The total size of this message in bytes; for mouse pointer shape update this includes this header and any image data that is contained within this packet; this does not include the size of any data contained within continuation packets.
TotalImageDataSize	32 bit unsigned	The total size of the image data for this cursor. **Note** The image data for a single cursor may be split between multiple packets.
CursorImageId	16 bit unsigned	The ID of this cursor images; this will be used to distinguish between new shapes and re-transmission of current shape
XPos	16 bit signed	The X position of the upper-left corner of the pointer position relative to this Miracast display, see notes below.
YPos	16 bit signed	The Y position of the upper-left corner of the pointer position relative to this Miracast display, see notes below
CursorImageType	8 bit unsigned	This indicates the type of cursor image being sent, valid values are: 0x01 Disabled 0x02 Masked color image, PNG compressed 0x03 Color cursor image, PNG compressed
HotSpotXPos	16 bit unsigned	The X-axis offset of the hot spot offset from the upper-left corner of the cursor image.
HotSpotYPos	16 bit unsigned	The Y-axis offset of the hot spot offset from the upper-left corner of the cursor image.
ImageData	8 bit unsigned array	Portion of the total cursor image data that is contained within this packet, the size of image data in this packet is PacketMsgSize-18. If PacketMsgSize-18 is equal to TotalImageDataSize, the complete cursor image is contained within this single packet and no continuation packet is needed for this cursor image update.

Below is the definition of the shape continuation packet that is used if the cursor shape data spans more than one UDP packet.

Field name	Type	Details
MsgType	8 bit unsigned	The type of cursor message, valid values 0x01 Mouse cursor position update 0x02 Mouse cursor shape start 0x03 Mouse cursor shape continuation For mouse cursor shape continuation this will be 0x03
PacketMsgSize	16 bit	The total size of this message in bytes; for mouse pointer shape

Field name	Type	Details
	unsigned	update this includes this header and any image data that is contained within this packet, this does not include the size of any data contained within continuation packets.
TotalImageDataSize	32 bit unsigned	The total size of the image data for this cursor. **Note** The image data for a single cursor may be split between multiple packets.
CursorImageId	16 bit unsigned	The ID of this cursor images; this will be used to distinguish between new shapes and re-transmission of current shape
PacketPayloadOffset	32 bit signed	The offset into the entire mouse shape data buffer (of compressed PNG data) where the ImageData in this packet should go. This allows the sink process the packets out of order as this gives them the information needed to copy this packets part of the mouse image into the correct location in the buffer.
ImageData	8 bit unsigned array	The portion of the total cursor image data that is contained within this packet, the size of image data in this packet is PacketMsgSize-13.

The mouse shape messages MUST always start at the beginning of a UDP packet, but may span multiple UDP packets because of its variable size. In this case, an RTP header is placed at the top of each UDP package.

The mouse pointer shape messages also contain the current mouse pointer position. Just like the mouse cursor position, it is updated only once per frame during the vertical blank period. The latest image replaces any previous image.

2.3 Directory Service Schema Elements

None.

3 Protocol Details

3.1 Source Details

Source sending mouse pointer position updates:

The source MUST send pointer position update messages as defined in section 2.2.2.

This mouse cursor position update message gives the new location of the upper-left side of the mouse cursor image. This is not affected by the location of the pointer's hot spot within the cursor image. If the sink needs to know the position of the hot spot (for example with UIBC), then it MUST add the hot spot offset that was sent in the last mouse cursor image update.

Note The X and Y position may be negative and the sink MUST perform any clipping that is necessary to ensure that only the visible part of the mouse cursor is displayed.

Because the UDP packets can be delivered out of order, the sink needs to maintain the RTP sequence number of the last mouse position update (either mouse position or shape packet) and store the new mouse position (or shape packet) if the RTP sequence number is higher (accounting for RTP sequence number wrap).

Source sending mouse pointer shape updates

The source MUST send mouse pointer shape update messages as defined in section 2.2.3.

Note If the image type is disabled, the sink stops displaying a hardware cursor from the start of the next frame.

The sink maintains the CursorImageId of the last shape update it received so when it receives a new mouse pointer shape packet, it can discard the packet if the CursorImageId is not greater than the previous shape update. In the case where the CursorImageId is the same, the XPos and the YPos can still be used to update the current pointer position (if RTP sequence number of shape update is greater than that of the last position update).

Because we do not use an acknowledgement mechanism, the source needs to transmit the cursor image multiple times to the sink to ensure that the sink displays the correct image. The source sends the same cursor image up to 4 times at 100ms intervals (1 transmission, then 3 retransmissions), this resent schedule is reset every time the mouse image is updated.

Testing Note: Even with a 64Kb UDP packet, it is still possible for the compressed image to span multiple packets, so the source and sink MUST both test with mouse shape updates that span multiple UDP packets. It is recommended to test with a 256x256 cursor image that compresses to more than 64Kb in size to verify this behavior. For source implementations, it is recommended to compile the graphics driver to use a very small UDP packet size to test this behavior.

3.1.1 Abstract Data Model

Mouse pointer position: The position of the mouse cursor on the source device's screen.

Mouse pointer shape: The image used to visually represent the mouse pointer, which may change shape contextually when the user performs actions like clicking links or resizing windows.

3.1.2 Timers

None.

3.1.3 Initialization

Initialization of this extension occurs during the capabilities query defined by the Miracast standard. A new optional RTSP parameter 'microsoft_cursor' is used to query for hardware cursor support and capabilities.

3.1.4 Higher-Layer Triggered Events

When a user causes a mouse cursor shape change, the source's graphics driver is informed, and MUST send a message to the sink communicating this shape change, as described in section 3.1.1.

3.1.5 Message Processing Events and Sequencing Rules

N/A. Refer to section 1.7 for capability query details.

3.1.6 Timer Events

None.

3.1.7 Other Local Events

None.

3.2 Sink Details

The mouse pointer position and shape on the display should be update once per frame during the vertical blank period to avoid any tearing of the mouse image.

3.2.1 Abstract Data Model

Mouse pointer position: The position of the mouse cursor on the source device's screen.

Mouse pointer shape: The image used to visually represent the mouse pointer, which may change shape contextually when the user performs actions like clicking links or resizing windows.

3.2.2 Timers

None.

3.2.3 Initialization

Initialization of this extension occurs during the capabilities query defined by the Miracast standard. A new optional RTSP parameter 'microsoft_cursor' is used to query for hardware cursor support and capabilities.

3.2.4 Higher-Layer Triggered Events

None.

3.2.5 Message Processing Events and Sequencing Rules

It is unnecessary for the sink to display the mouse pointer for every individual shape or position message received—just the most recent one. For example, if during one frame the sink receives 10

position updates and 5 shape changes, then the mouse pointer position and shape for the next frame should be the most recent position and shape that the sink received.

For example:

Event	Frame number being scanned out to TV/monitor	Mouse position displayed on screen	Mouse shape displayed on screen
VSync of TV/monitor	0	Pos1	Shape1
VSync of TV/monitor	1	Pos1	Shape1
Receive new mouse position Pos2	1	Pos1	Shape1
Receive new mouse position Pos3	1	Pos1	Shape1
Receive new mouse shape Shape2 Pos 4	1	Pos1	Shape1
VSync of TV/monitor	2	Pos4	Shape2
Receive new mouse position Pos5	2	Pos4	Shape2
Receive new mouse shape Shape3 Pos 6	2	Pos4	Shape2
Receive new mouse position Pos7	2	Pos4	Shape2
Receive new mouse shape Shape4 Pos 8	2	Pos4	Shape2
Receive new mouse position Pos9	2	Pos4	Shape2
Receive new mouse position Pos10	2	Pos4	Shape2
VSync of TV/monitor	3	Pos10	Shape4

3.2.6 Timer Events

None.

3.2.7 Other Local Events

Sources may support multiple cursor types, though the source MUST convert the cursor image and PNG compress it before sending it to the sink. The table below illustrates how the source converts the different types of cursors into cursor images to send to the sink. As can be seen below, a sink that supports the XOR operation only receives masked color and color cursor shapes, but in the case where a sink does not support the XOR operation, it is sent as a color cursor with 8bpp alpha.

Source cursor type	Sink that supports XOR	Sink that does not support XOR
Monochrome without XOR pixels	Masked color without XOR	Color cursor with 8bpp alpha
Monochrome with XOR pixels	Masked color with XOR	Color cursor with 8bpp alpha
Masked color without XOR pixels	Masked color without XOR	Color cursor with 8bpp alpha

Source cursor type	Sink that supports XOR	Sink that does not support XOR
Masked color with XOR pixels	Masked color with XOR	Color cursor with 8bpp alpha
Color cursor with 8bpp alpha	Color cursor with 8bpp alpha	Color cursor with 8bpp alpha

4 Protocol Examples

On initial connection, a source that has implemented the hardware cursor extension queries the capabilities of the sink through the M3 message defined in the Miracast protocol [WF-DTS1.1]. The sink responds with acknowledgement of capabilities and supported extensions.

If the sink reports that it does support the hardware cursor extension, the source sends cursor position and shape updates as specified by the extension. The following is an example response message from a sink that supports the hardware cursor extension.

M3 message	Sink response
microsoft_cursor	full 0x0200 0x0200 50001

The previous example indicates that this sink supports XOR cursor types (full), supports cursor shapes up to 512x512 pixels, and has chosen to communicate over port 50001.

If the sink does not explicitly report that it supports the hardware cursor extension, the source encodes the cursor image into the desktop image. Such a source would simply report no support in the following response.

M3 message	Sink response
microsoft_cursor	none

When a sink reports support of the hardware cursor extension, it will then receive cursor messages from the source for the duration of the Miracast session each time the cursor position or shape changes on the source. The sink will receive multiple mouse cursor message types.

The example below demonstrates a sample mouse cursor position update; when receiving a cursor position update the sink must then update its internal cursor position. The example message below indicates that the cursor position has been updated, and is located at the x, y coordinate (13,10).

Message parameter	Value
MsgType	0x1
PacketMsgSize	0x7
Xpos	12
Ypos	10

The example below demonstrates a sample mouse shape update message; when receiving a cursor shape update, the sink must update the cursor image displayed. The example below demonstrates the more complex example in which the shape change message is large enough that we require a shape change continuation message as well.

Initial shape change message

Message parameter	Value
MsgType	0x2
PacketMsgSize	0x112
TotalImageDataSize	0x200

Message parameter	Value
CursorImageID	0x1234
Xpos	12
Ypos	10
CursorImageType	0x3
HotSpotXPos	18
HotSpotYPos	15
ImageData	First 0x100 bytes of PNG cursor

Shape continuation message

Message parameter	Value
MsgType	0x3
PacketMsgSize	0x10D
TotalImageDataSize	0x200
CursorImageID	0x1234
PacketPayloadOffset	0x100
ImageData	Next 0x100 bytes of PNG cursor

For smaller shape change message, the continuation message is not necessary but this example has shown a larger message for completeness.

Intel Fast Cursor Message

Examples of valid fast cursor messages and the corresponding cursor position when in a 1920x1080 resolution are:

Message	Description
fast_cursor=1920:1080:0:0:0	Cursor at upper-left corner
fast_cursor=1920:1080:1919:1079:0	Cursor at lower-right corner
fast_cursor=1366:768:682:383:0	Cursor at center of the screen
fast_cursor=0:0:0:0:0	Cursor is hidden

5 Security

5.1 Security Considerations for Implementers

None.

5.2 Index of Security Parameters

None.

6 Appendix A: Product Behavior

The information in this specification is applicable to the following Microsoft products or supplemental software. References to product versions include released service packs.

Windows 10 operating system

Exceptions, if any, are noted below. If a service pack or Quick Fix Engineering (QFE) number appears with the product version, behavior changed in that service pack or QFE. The new behavior also applies to subsequent service packs of the product unless otherwise specified. If a product edition appears with the product version, behavior is different in that product edition.

Unless otherwise specified, any statement of optional behavior in this specification that is prescribed using the terms SHOULD or SHOULD NOT implies product behavior in accordance with the SHOULD or SHOULD NOT prescription. Unless otherwise specified, the term MAY implies that the product does not follow the prescription.

<1> Section 2.2.2: Intel Fast Cursor is not supported in Windows 10 v1507 operating system.

7 Change Tracking

This section identifies changes that were made to this document since the last release. Changes are classified as New, Major, Minor, Editorial, or No change.

The revision class **New** means that a new document is being released.

The revision class **Major** means that the technical content in the document was significantly revised. Major changes affect protocol interoperability or implementation. Examples of major changes are:

- A document revision that incorporates changes to interoperability requirements or functionality.
- The removal of a document from the documentation set.

The revision class **Minor** means that the meaning of the technical content was clarified. Minor changes do not affect protocol interoperability or implementation. Examples of minor changes are updates to clarify ambiguity at the sentence, paragraph, or table level.

The revision class **Editorial** means that the formatting in the technical content was changed. Editorial changes apply to grammatical, formatting, and style issues.

The revision class **No change** means that no new technical changes were introduced. Minor editorial and formatting changes may have been made, but the technical content of the document is identical to the last released version.

Major and minor changes can be described further using the following change types:

- New content added.
- Content updated.
- Content removed.
- New product behavior note added.
- Product behavior note updated.
- Product behavior note removed.
- New protocol syntax added.
- Protocol syntax updated.
- Protocol syntax removed.
- New content added due to protocol revision.
- Content updated due to protocol revision.
- Content removed due to protocol revision.
- New protocol syntax added due to protocol revision.
- Protocol syntax updated due to protocol revision.
- Protocol syntax removed due to protocol revision.
- Obsolete document removed.

Editorial changes are always classified with the change type **Editorially updated**.

Some important terms used in the change type descriptions are defined as follows:

- **Protocol syntax** refers to data elements (such as packets, structures, enumerations, and methods) as well as interfaces.
- **Protocol revision** refers to changes made to a protocol that affect the bits that are sent over the wire.

The changes made to this document are listed in the following table. For more information, please contact dochelp@microsoft.com.

Section	Tracking number (if applicable) and description	Major change (Y or N)	Change type
1.3 Overview	Added information about Intel Fast Cursor support in Windows 10.	Y	Content update.
1.7 Versioning and Capability Negotiation	Added information about Intel Fast Cursor support in Windows 10.	Y	Content update.
2.2.2 Mouse pointer position message	Added information about Intel Fast Cursor support in Windows 10.	Y	Content update.
4 Protocol Examples	Added information about Intel Fast Cursor support in Windows 10.	Y	Content update.

8 Index