[MS-DMCT]:
Device Media Control Protocol

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, email 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
11/06/2009	0.1	Major	First Release.
12/18/2009	0.1.1	Editorial	Revised and edited the technical content.
01/29/2010	0.1.2	Editorial	Revised and edited the technical content.
03/12/2010	0.1.3	Editorial	Revised and edited the technical content.
04/23/2010	0.1.4	Editorial	Revised and edited the technical content.
06/04/2010	1.0	Major	Updated and revised the technical content.
07/16/2010	1.0.1	Editorial	Changed language and formatting in the technical content.
08/27/2010	1.0.1	No change	No changes to the meaning, language, or formatting of the technical content.
10/08/2010	1.0.1	No change	No changes to the meaning, language, or formatting of the technical content.
11/19/2010	1.0.1	No change	No changes to the meaning, language, or formatting of the technical content.
01/07/2011	1.0.1	No change	No changes to the meaning, language, or formatting of the technical content.
02/11/2011	1.0.1	No change	No changes to the meaning, language, or formatting of the technical content.
03/25/2011	1.0.1	No change	No changes to the meaning, language, or formatting of the technical content.
05/06/2011	1.0.1	No change	No changes to the meaning, language, or formatting of the technical content.
06/17/2011	1.1	Minor	Clarified the meaning of the technical content.
09/23/2011	1.1	No change	No changes to the meaning, language, or formatting of the technical content.
12/16/2011	2.0	Major	Significantly changed the technical content.
03/30/2012	2.0	No change	No changes to the meaning, language, or formatting of the technical content.
07/12/2012	2.0	No change	No changes to the meaning, language, or formatting of the technical content.
10/25/2012	2.0	No change	No changes to the meaning, language, or formatting of the technical content.

Date	Revision History	Revision Class	Comments
01/31/2013	2.0	No change	No changes to the meaning, language, or formatting of the technical content.
08/08/2013	3.0	Major	Significantly changed the technical content.
11/14/2013	4.0	Major	Significantly changed the technical content.
02/13/2014	4.0	No change	No changes to the meaning, language, or formatting of the technical content.

Contents

1 Introduction

This document specifies the Device Media Control Protocol. This protocol uses the Device Services Lightweight Remoting Protocol specified in [MS-DSLR] to enable a computer to control media playback in an active device session.

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 RFC 2119. Sections 1.5 and 1.9 are also normative but cannot contain those terms. All other sections and examples in this specification are informative.

1.1 Glossary

The following terms are defined in [MS-GLOS]:

big-endian
little-endian
globally unique identifier (GUID)
session

The following terms are defined in [MS-DSLR]:

proxy
stub
tag

The following terms are defined in [MS-RTSP]:

content
playlist

The following terms are specific to this document:

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

1.2 References

References to Microsoft Open Specifications documentation do not include a publishing year because links are to the latest version of the documents, which are updated frequently. References to other documents include a publishing year when one is available.

A reference marked "(Archived)" means that the reference document was either retired and is no longer being maintained or was replaced with a new document that provides current implementation details. We archive our documents online [Windows Protocol].

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-DSLR] Microsoft Corporation, "Device Services Lightweight Remoting Protocol".

[MS-DTYP] Microsoft Corporation, "Windows Data Types".

[MS-RTSP] Microsoft Corporation, "Real-Time Streaming Protocol (RTSP) Windows Media Extensions".

[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

[RFC3550] Schulzrinne, H., Casner, S., Frederick, R., and Jacobson, V., "RTP: A Transport Protocol for Real-Time Applications", STD 64, RFC 3550, July 2003, http://www.ietf.org/rfc/rfc3550.txt

[RFC3556] Casner, S., "Session Description Protocol (SDP) Bandwidth Modifiers for RTP Control Protocol (RTCP) Bandwidth", RFC 3556, July 2003, http://www.ietf.org/rfc/rfc3556.txt

[RFC4566] Handley, M., Jacobson, V., and Perkins, C., "SDP: Session Description Protocol", RFC 4566, July 2006, http://www.ietf.org/rfc/rfc4566.txt

1.2.2 Informative References

[MS-DRMND] Microsoft Corporation, "Windows Media Digital Rights Management (WMDRM): Network Devices Protocol".

[MS-GLOS] Microsoft Corporation, "Windows Protocols Master Glossary".

1.3 Overview

The Device Media Control Protocol can be viewed as a set of services implemented and offered between the extender device (client) and a computer (host) so that the computer can remotely control a media **session** on an extender device. The computer sends media control operation requests, such as OpenMedia and CloseMedia, to the client to control video playback. The client in return sends asynchronous events, such as end of file, to the computer. This protocol uses the Device Services Lightweight Remoting Protocol specified in [MS-DSLR] to enable the remoting of services between the two devices over a reliable point-to-point channel.

The following block diagram shows the relationship between the host device (that is, the computer) and the client device (the extender device):

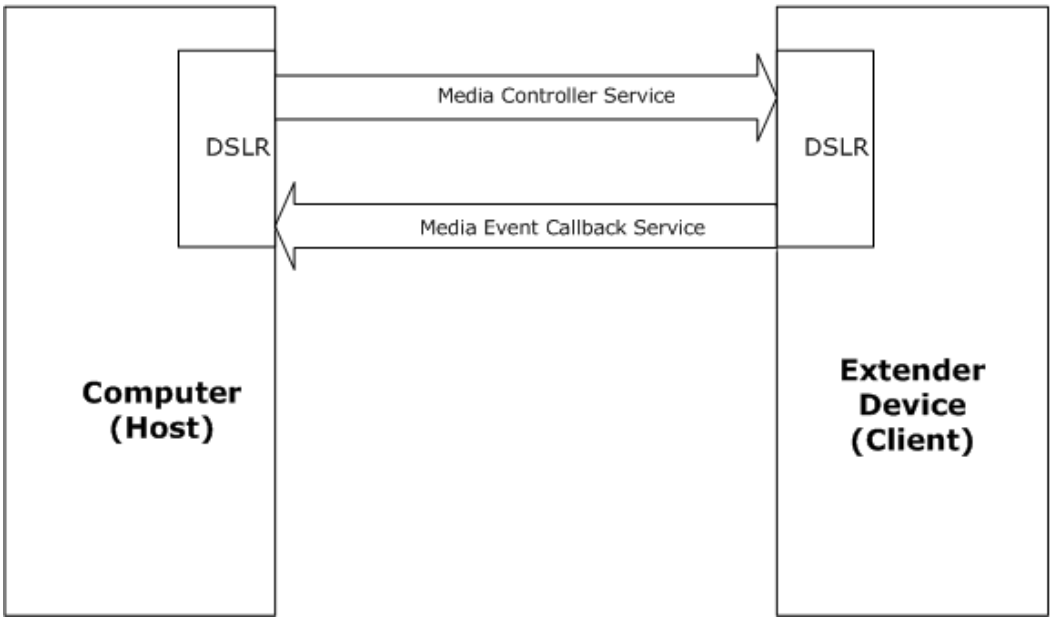


Figure 1: Device Media Control Protocol block diagram

The Media Controller service must be implemented and offered by the extender device (acting in this case as the Device Services Lightweight Remoting (DSLR) **stub**/server while the computer acts as the DSLR **proxy**/client, in DSLR nomenclatures. For a more detailed definition of these roles, see [MS-DSLR]). The Media Controller service contains the following functions:

OpenMedia: This message is used to open the streaming media session on an **extender device**.

CloseMedia: This message is called to close the streaming media session.

Start: This message is used to request the extender device to start streaming **media** samples and playing them.

Pause: This message is called to pause the streaming media session.

Stop: This message is called to stop the streaming media session.

GetDuration: This message is called to get the duration of media.

GetPosition: This message is called to get the current position of **media**.

RegisterMediaEventCallback: This message is used to create and connect the Media Event Callback Service between the extender device and the computer to get events from the extender device to host.

UnRegisterMediaEventCallback: This message is used to disconnect and release the current Media Event Callback Service between the extender device and the computer.

The Media Event Callback service is implemented and offered by the computer (acting in this case as the stub while the extender device acts as the proxy). This service contains following function:

OnMediaEvent: This message is used to send events from an extender device to a computer.

1.4 Relationship to Other Protocols

The Device Media Control Protocol uses the Device Services Lightweight Remoting Protocol to enable the remote control of the **media** session.

1.4.1 Device Services Lightweight Remoting Protocol

The Device Services Lightweight Remoting Protocol (DSLR) specified in [MS-DSLR] is a COM-like protocol that enables remoting of services (for example, function calls, events, and so on) over a reliable point-to-point connection. It enables an application to call functions on and/or send events to a remote device over the established channel. The service itself is implemented on the local/stub side of the connection, and the remote side creates a proxy for that service. The Device Services Lightweight Remoting Protocol is direction agnostic; that is, each side of the connection can act as both a proxy for a remote service and a stub that manages calls into a local service. Both the stub and proxy are implemented by the DSLR consumer; each side has knowledge of the functions and events exposed by the service, as well as the in/out parameters for each. By convention, the request/response calling convention follows COM rules, that is:

- The function returns an HRESULT.
- All [in] parameters are serialized in the request **tag**.
- The returned HRESULT is serialized in the response tag, followed, if successful, by the [out] parameters.
- The caller should expect the returned HRESULT to be either one of the values returned by the function, or one of the DSLR failure values.
- The caller may not evaluate any of the [out] parameters if the call returned a failure.

1.4.2 Real Time Streaming Protocol (RTSP)

The Real Time Streaming Protocol (RTSP), specified in [MS-RTSP], is used for transferring real-time multimedia data (for example, audio and video) between a server and a client. It is a streaming protocol; this means that RTSP attempts to facilitate scenarios in which the multimedia data is being simultaneously transferred and rendered (that is, video is displayed and audio is played).

RTSP typically uses a TCP connection for control of the streaming media session, although it is also possible to use UDP for this purpose. The entity that sends the RTSP request that initiates the session is referred to as the client, and the entity that responds to that request is referred to as the server. Typically, multimedia data flows from the server to the client. RTSP also allows multimedia data to flow in the opposite direction.

Clients can send RTSP requests to the server requesting information on **content** before a session is established. The information that the server returns is formatted by using a syntax called Session Description Protocol (SDP), as specified in [RFC4566]. Clients use RTSP requests to control the session and to request that the server perform actions, such as starting or stopping the flow of multimedia data. Each request has a corresponding RTSP response that is sent in the opposite direction. Servers can also send RTSP requests to clients, for example, to inform them that the session state has changed. If TCP is used to exchange RTSP requests and responses, the multimedia data can also be transferred over the same TCP connection. Otherwise, the multimedia data is transferred over UDP.

The multimedia data is encapsulated in Real-time Transport Protocol (RTP) packets, as specified in [RFC3550]. For each RTP stream, the server and client can also exchange Real-Time Transport Control Protocol (RTCP) packets, as specified in [RFC3556].

1.5 Prerequisites/Preconditions

For the Device Media Control Protocol to function properly, the following assumptions must be met:

- A network connection has been established between the host and the client (extender device).
- The Device Services Lightweight Remoting Protocol modules have been initialized and started on both devices. Once completed, the proxy side must call the CreateService message to instantiate the service on the stub side, and create a proxy for that service (that is, an object that implements the proxied service's interfaces). As part of the CreateService request, it allocates a service handle that is sent to the stub side. This handle will subsequently be used when calling functions on the service and to terminate the service via the DeleteService message.
  - The following class/service GUIDS are passed in the CreateService message (see [MS-DSLR] section 2.2.2.3) and the DeleteService message (see [MS-DSLR] section 2.2.2.4) messages for the Media Controller service:

    ClassID GUID: 18c7c708-c529-4639-a846-5847f31b1e83.

    ServiceID GUID: 601df477-89b6-43b4-95bc-50e8dfef12eb.
  - The following class/service GUIDS are passed in the CreateService and DeleteService messages for the Media Event callback event:

    ClassID GUID: This GUID is newly generated each time.

    ServiceID GUID: 6d72a615-ca26-4420-95ac-4e4695991015.
- RTSP or HTTP server and client engines have been initialized and started on both devices.
- The player and player controller have been initialized and started on both devices.

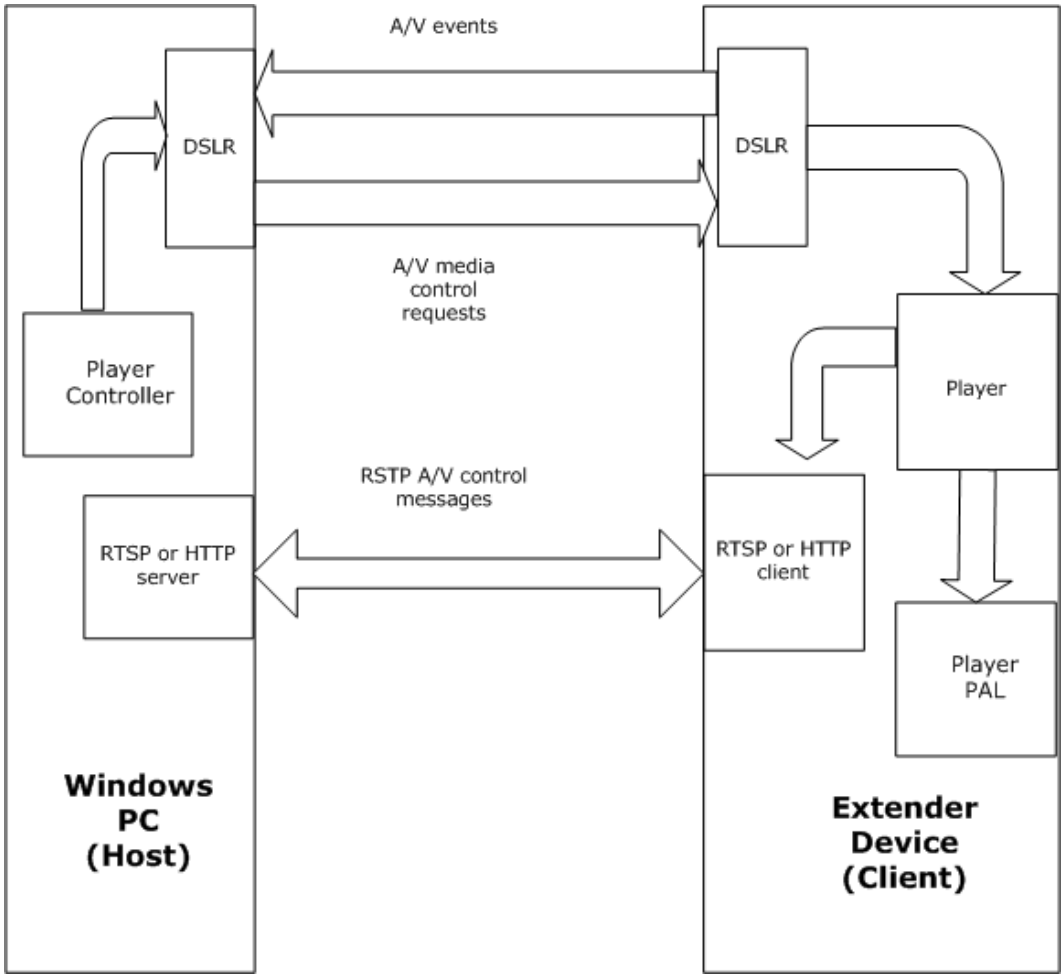


Figure 2: Relationships between host computer and extender device

1.6 Applicability Statement

The Device Media Control Protocol provides a mechanism for two remote devices to control **media** sessions on a remote device over a point-to-point channel. This, of course, requires that the Device Services Lightweight Remoting Protocol, RTSP/HTTP and a player/player controller be available and running on both devices.

1.7 Versioning and Capability Negotiation

None.

1.8 Vendor-Extensible Fields

This protocol uses HRESULT values as defined in [MS-DSLR] section 2.2.2.5, as well as specific HRESULT values defined in section 2.2 of this document.

1.9 Standards Assignments

None.

2 Messages

This protocol references commonly used data types as defined in [MS-DTYP].

2.1 Transport

Messages are transported over the Device Services Lightweight Remoting Protocol, which can be implemented on top of any stream-based or message-based reliable transport.

2.2 Message Syntax

The Device Services Lightweight Remoting Protocol uses a tag-based formatting for its messages; see [MS-DSLR] for details of the tag formats.

The Device Media Control Protocol messages MUST follow the Device Services Lightweight Remoting Protocol message syntax for requests and responses, as specified in [MS-DSLR] section 2.2.2.

The Device Services Lightweight Remoting Protocol payload for a request is defined by the Device Services Lightweight Remoting Protocol Dispatcher Request tag payload, followed by the child payload for a given message (that is, the function parameters for the given message). The Request tag payload includes the service handle for the specific service (see section 1.5 for how this service handle is obtained), the function handle for the specific function being called on that service (defined by the service), the calling convention for that function, and a one-time request handle allocated by the client for each request. See section 2.2.1 for the format of the Device Services Lightweight Remoting Protocol Dispatcher Request tag payload.

The Device Services Lightweight Remoting Protocol payload for a response is defined by the Device Services Lightweight Remoting Protocol Dispatcher Response tag payload, followed by the child payload a given message (that is, the result and return parameters for the given message). The Response tag payload includes the callingConvention and the matching one-time request handle to which this response corresponds. See section 2.2.1 for the format of the Device Services Lightweight Remoting Protocol Dispatcher Response tag payload.

The format of the data types for input and output parameters for the following functions are defined in [MS-DSLR]; see section 2.2.2 for valid input/output parameters and how they are formatted on the wire (**big-endian** or **little-endian**). All numeric data are big-endian byte-oriented.

For more details on the Device Services Lightweight Remoting Protocol message syntax, see [MS-DSLR].

2.2.1 Media Controller Service

The host uses this service to control media sessions on an extender device. The host uses a proxy code to send messages to the extender device while the extender device has a stub to receive messages.

2.2.1.1 OpenMedia

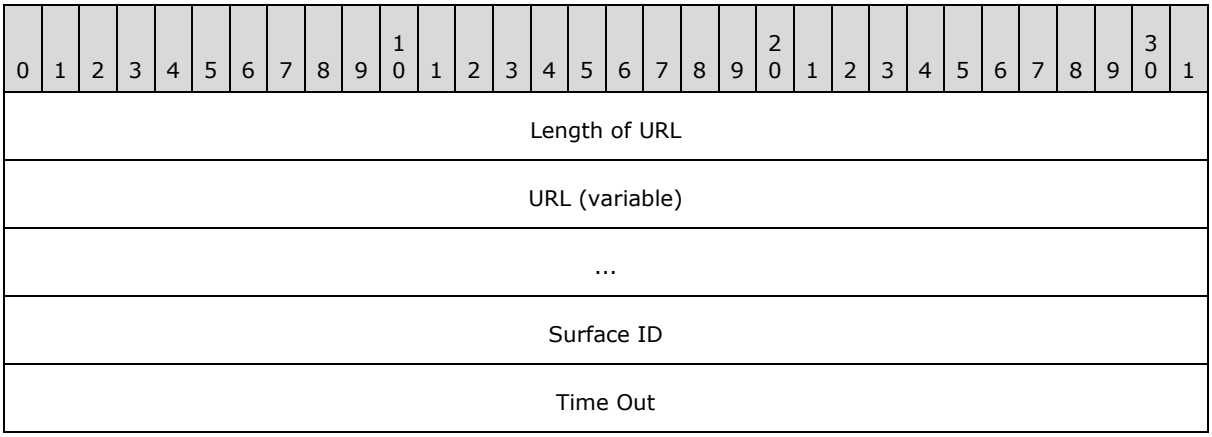
OpenMedia is a two-way request message.

2.2.1.1.1 OpenMedia (request)

The *callingConvention* parameter in the Dispatch Request tag MUST be dslrRequest (0x00000001). The function handle for the Dispatch Request tag for OpenMedia MUST be 0x00000000.

This message is used to open a streaming media session on an **extender device**.

Request payload (input parameters):



Length of URL (4 bytes): An unsigned 32-bit integer. This is length of the URL.

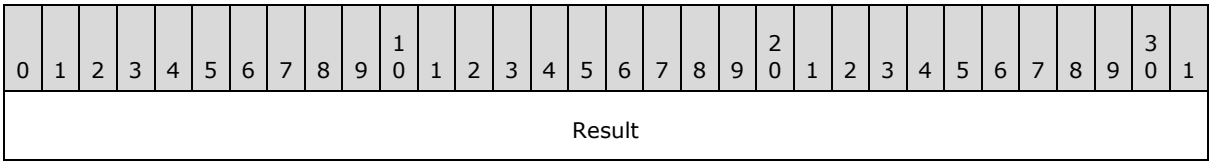
URL (variable): A byte array of UTF-8 string data of size equal to the length of the URL. It is a URL of the streaming media server. It can be either a local **media** file URL or a remote URL. If it is a local media file, then the URL will start with "rtsp:" so the URL will play using RTSP streaming. If the URL is web-based, then it will start with "http:" and will play using HTTP streaming.

Surface ID (4 bytes): An unsigned 32-bit integer. It is used to select a rendering surface if the extender device supports simultaneous streams by maintaining an array of Video Producers with each one tied to a different dynamic surface ID.

Time Out (4 bytes): An unsigned 32-bit integer. This is the time that the extender device waits for an RTSP or HTTP response. It MUST be greater than 5 seconds. It is 30 seconds for local streams and 45 second for online streams.

2.2.1.1.2 OpenMedia (response)

Response payload (result and output parameters):



Result (4 bytes): An unsigned 32-bit integer. An HRESULT returned from the function call (**BE**). See [MS-DSLR] section 2.2.2.5 for error codes. The following errors are specific to the Device Media Control Protocol.<1>

Value	Meaning
E_FILE_NOT_FOUND 0x80070002	Cannot open the URL.
E_INVALID_REQUEST	The extender device is not initialized and not closed.

Value	Meaning
0x80004007	
E_INVALID_STREAM 0x800DFF01	Invalid stream settings.
E_MDM_STREAM_TYPE_NOT_SUPPORTED 0xC0000004	The extender device does not have a decoder for the input media type.
E_UNSUPPORTED_STREAM_TYPE 0x800D0003	The media server does not support the input media format.
E_FIRMWARE_UPDATE_REQUIRED 0x80099702	If the extender device does not have a required codec to decode the content, it sends this error saying a new codec pack needs to be downloaded.
E_H264_CODECPACK_REQUIRED 0x80099703	If the extender device does not have the H.264 codec to decode the content, it sends this error saying the H.264 codec pack needs to be downloaded.
E_RTSP_NO_CONNECTION 0x800B0000	If the extender device takes more than the number of seconds specified in the *timeout* parameter to establish a connection, then it sends this error.

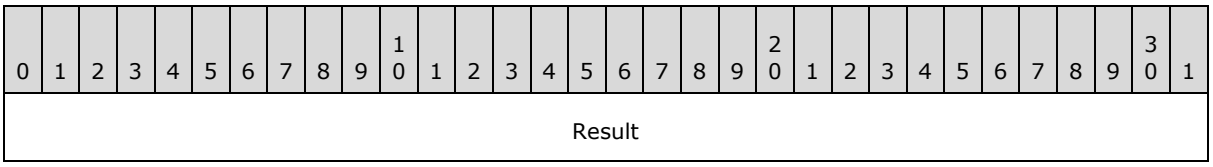
2.2.1.2 CloseMedia

CloseMedia is a two-way request message, so the *callingConvention* parameter in the Dispatch Request tag MUST be dslrRequest (0x00000001). The function handle for the Dispatch Request tag for CloseMedia MUST be 0x00000001.

This message is called to close the streaming media session.

The message does not take any input parameters.

Response payload (result and output parameters):



Result (4 bytes): An unsigned 32-bit integer. An HRESULT that is returned from the function call. See [MS-DSLR] section 2.2.2.5 for error codes.

2.2.1.3 Start

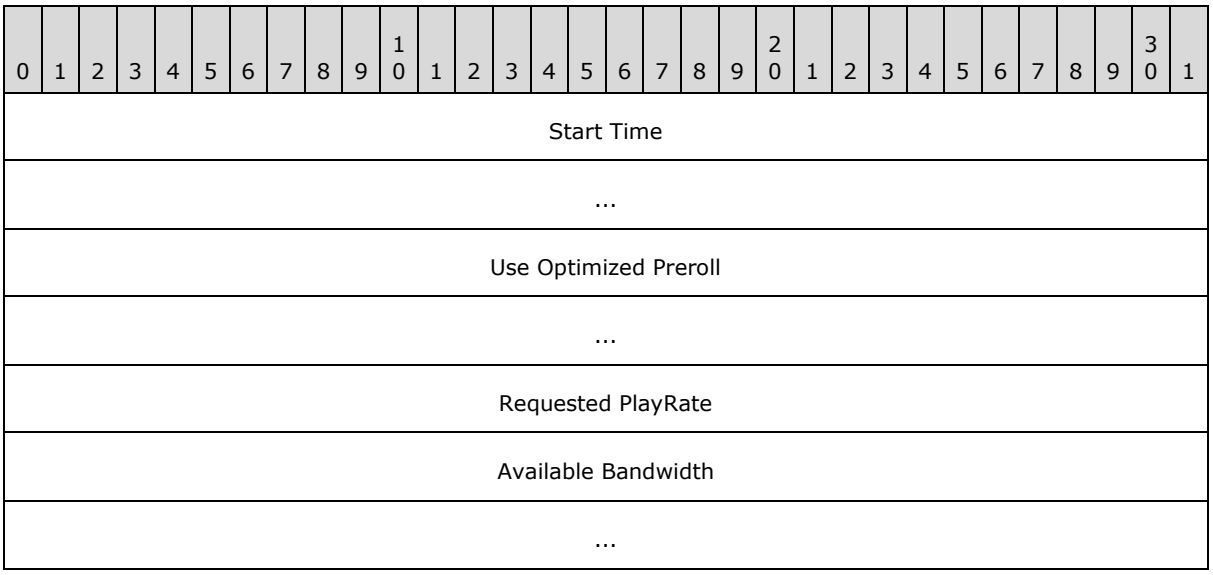
Start is a two-way request message.

2.2.1.3.1 Start (request)

The *CallingConvention* parameter in the Dispatch Request tag MUST be dslrRequest (0x00000001) (see [MS-DSLR] section 2.2.2.1). The function handle for the Dispatch Request tag for Start MUST be 0x00000002.

This message is used to request the server to start streaming media samples and playing them.

Request payload (input parameters):



Start Time (8 bytes): An unsigned 64-bit integer. The value of the field is either a time position in the presentation from which the server starts to stream media samples, specified in milliseconds, or the value 0xFFFFFFFFFFFFFFFF. The field can be set to 0xFFFFFFFFFFFFFFFF when resuming from pause, or if a suitable start time cannot be determined.

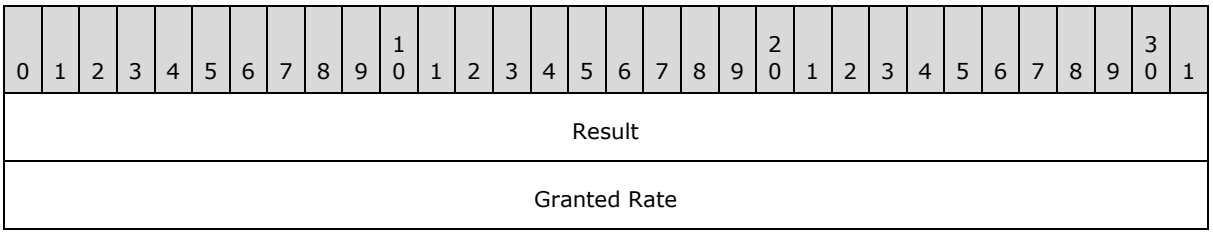
Use Optimized Preroll (8 bytes): An unsigned 64-bit integer. It is used to determine whether to use optimized preroll or not. It can be set to either 1 (to use optimized preroll) or 0 (not to use optimized preroll).

Requested PlayRate (4 bytes): An unsigned 32-bit integer. It is a trick-play rate of the content. This parameter MUST NOT be 0. This field is specified as 1 for normal playback. A value greater than 1 requests fast forward, and a negative value requests the server to rewind the presentation.

Available Bandwidth (8 bytes): An unsigned 64-bit integer. It is used to indicate available bandwidth in bits per second. It can be set to 0 to let the client decide the value. Its default value is 22 megabytes per second.

2.2.1.3.2 Start (response)

Response payload (result and output parameters):



Result (4 bytes): An unsigned 32-bit integer. An HRESULT that is returned from the function call. See [MS-DSLR] section 2.2.2.5 for error codes.

Granted Rate (4 bytes): An unsigned 32-bit integer. It is the play-rate granted by the extender device.

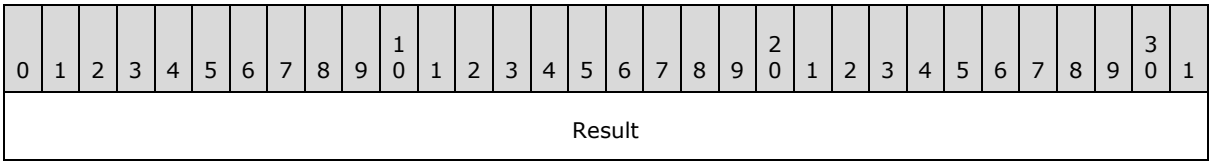
2.2.1.4 Pause

Pause is a two-way request message, so the *CallingConvention* parameter in the Dispatch Request tag MUST be dslrRequest (0x00000001) (see [MS-DSLR] section 2.2.2.1). The function handle for the Dispatch Request tag for Pause MUST be 0x00000003.

This message is called to pause the streaming media session.

The message does not take any input parameters.

The response payload (result and output parameters) is as follows:



Result (4 bytes): An unsigned 32-bit integer. An HRESULT that is returned from the function call. See [MS-DSLR] section 2.2.2.5 for error codes.

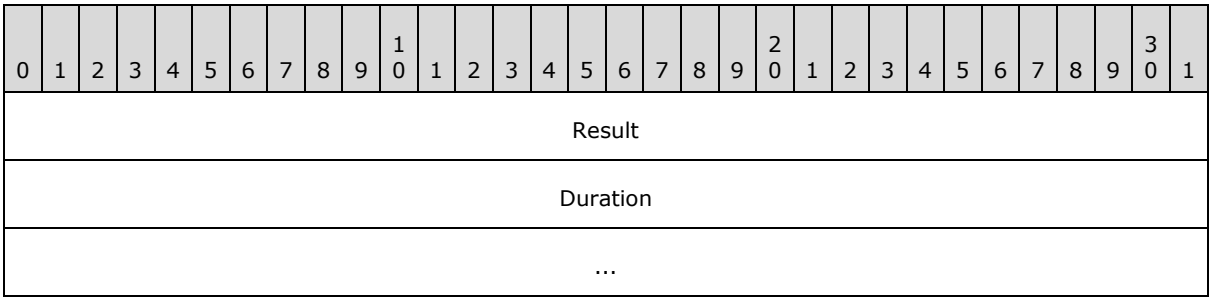
2.2.1.5 GetDuration

GetDuration is a two-way request message, so the *CallingConvention* parameter in the Dispatch Request tag MUST be dslrRequest (0x00000001) (see [MS-DSLR]section 2.2.2.1). The function handle for the Dispatch Request tag for GetDuration MUST be 0x00000005.

This message is called to get the duration of media.

The message does not take any input parameters.

The Response payload (result and output parameters) is as follows:



Result (4 bytes): An unsigned 32-bit integer. An HRESULT that is returned from the function call. See [MS-DSLR] section 2.2.2.5 for error codes.

Duration (8 bytes): An unsigned 64-bit integer. It is used to get the duration of media in units of 10 milliseconds.

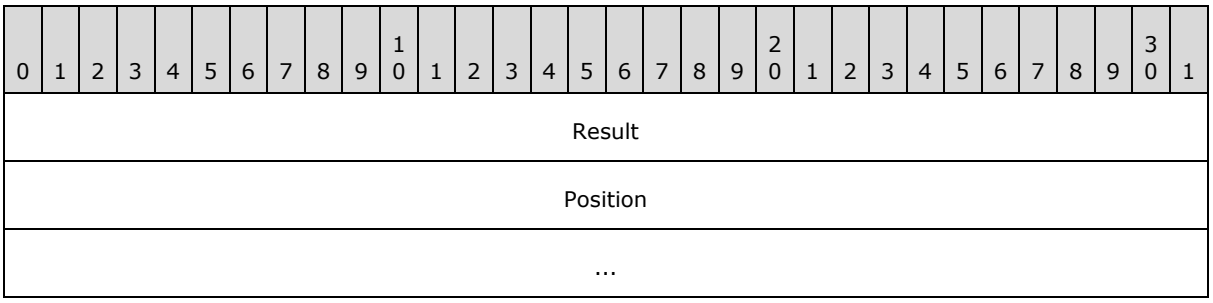
2.2.1.6 GetPosition

GetPosition is a two-way request message, so the *CallingConvention* parameter in the Dispatch Request tag MUST be dslrRequest (0x00000001) (see [MS-DSLR] section 2.2.2.1). The function handle for the Dispatch Request tag for GetPosition MUST be 0x00000006.

This message is called to get the current position of media.

The message does not take any input parameters.

The response payload (result and output parameters) is as follows:



Result (4 bytes): An unsigned 32-bit integer. An HRESULT that is returned from the function call. See [MS-DSLR] section 2.2.2.5 for error codes.

Position (8 bytes): An unsigned 64-bit integer. It is used to get the current position of media in units of 10 milliseconds.

2.2.1.7 RegisterMediaEventCallback

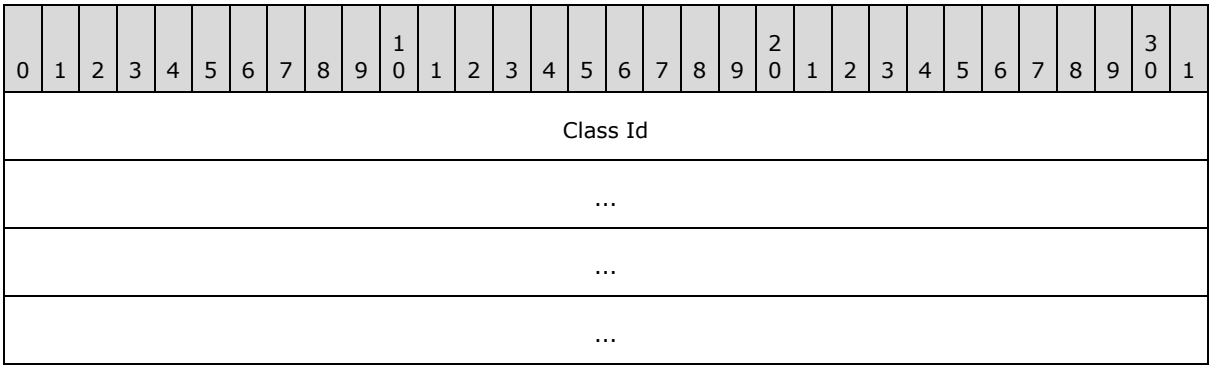
RegisterMediaEventCallback is a two-way request message.

2.2.1.7.1 RegisterMediaEventCallback (request)

The *CallingConvention* parameter in the Dispatch Request tag MUST be dslrRequest (0x00000001). The function handle for the Dispatch Request tag for RegisterMediaEventCallback MUST be 0x00000008.

This message is used to create and connect the Media Event Callback Service between the extender device and the host computer and to get events from the extender device to the host.

The request payload (input parameters) is as follows:



Service ID
...
...
...

Class Id (16 bytes): A 16-byte **GUID** that consists of a **DWORD** (4 bytes, unsigned 32-bit integer) Data1, WORD (2 bytes, unsigned 16-bit integer) Data2, WORD (2 bytes, unsigned 16-bit integer) Data3, Data4 = 8 bytes field. It MUST be a unique ID per session.

Service ID (16 bytes): A 16-byte GUID that consists of **DWORD** (4 bytes, unsigned 32-bit integer) Data1, WORD (2 bytes, unsigned 16-bit integer) Data2, WORD (2 bytes, unsigned 16-bit integer) Data3, Data4 = 8 bytes field. It MUST be defined as 6d72a615-ca26-4420-95ac-4e4695991015.

2.2.1.7.2 RegisterMediaEventCallback (response)

Response payload (result and output parameters):

0	1	2	3	4	5	6	7	8	9	10	1	2	3	4	5	6	7	8	9	20	1	2	3	4	5	6	7	8	9	30	1
Result																															
Cookie																															

Result (4 bytes): An unsigned 32-bit integer. An HRESULT that is returned from the function call. See [MS-DSLR] section 2.2.2.5 for error codes. <2>

Cookie (4 bytes): An unsigned 32-bit integer. The extender device will send a random number as a cookie.

2.2.1.8 UnRegisterMediaEventCallback

UnRegisterMediaEventCallback is a two-way request message.

2.2.1.8.1 UnRegisterMediaEventCallback(request)

The *CallingConvention* parameter in the Dispatch Request tag MUST be dslrRequest (0x00000001). The function handle for the Dispatch Request tag for UnRegisterMediaEventCallback MUST be 0x00000009.

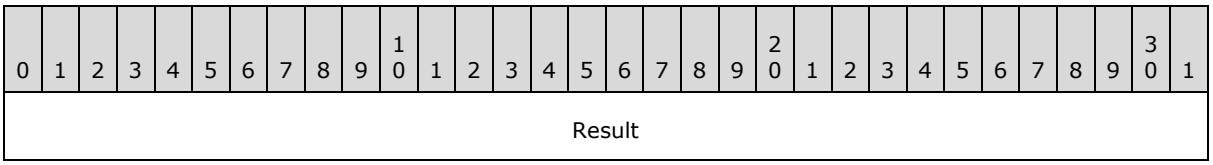
This message is used to disconnect and release the current Media Event Callback Service between the **extender device** and the host.

The request payload (input parameters) is as follows:

Cookie (4 bytes): An unsigned 32-bit integer. It is a value sent by the extender device while registering.

2.2.1.8.2 UnRegisterMediaEventCallback (response)

The response payload (result and output parameters) is as follows:



Result (4 bytes): An unsigned 32-bit integer. An HRESULT that is returned from the function call. See [MS-DSLR] section 2.2.2.5 for error codes.

2.2.2 Media Event Callback

The **extender device** uses this service to respond to out-of-band errors that the device has encountered or for end-of-stream states that aid in support of the three-box model. Errors include device subsystem failures (decoder errors), Digital Rights Management (DRM) errors, and so on. The **extender device** has a proxy code to send messages to the host while the host has a stub to receive messages.

2.2.2.1 OnMediaEvent

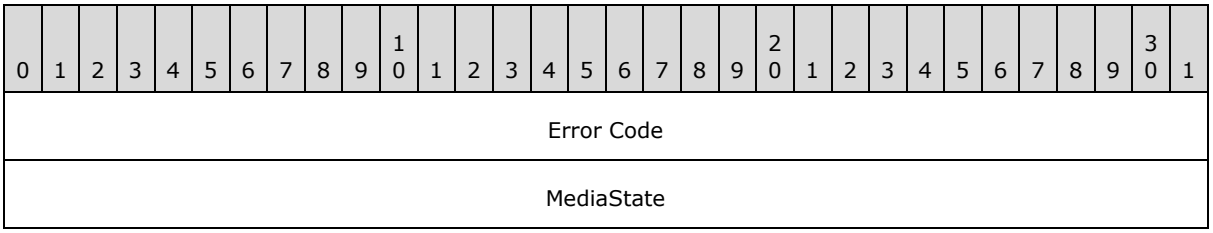
OnMediaEvent is a two-way request message.

2.2.2.1.1 OnMediaEvent (request)

The *callingConvention* parameter in the Dispatch Request tag MUST be dslrRequest (0x00000001). The function handle for the Dispatch Request tag for OnMediaEvent MUST be 0x00000000.

This message is used to send events from an extender device to a host.

Request payload (input parameters):

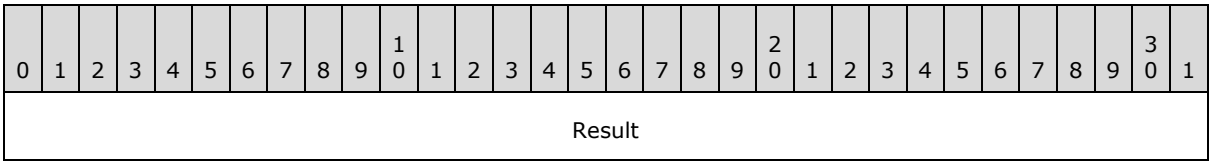


Error Code (4 bytes): An unsigned 32-bit integer. It is an error code value.

MediaState (4 bytes): An unsigned 32-bit integer. It is a value to indicate the media state. It can have any of the values for **Result**, as specified in section 2.2.2.1.2.

2.2.2.1.2 OnMediaEvent (response)

The response payload (result and output parameters) is as follows:



Result (4 bytes): An unsigned 32-bit integer. An HRESULT that is returned from the function call. See [MS-DSLR] section 2.2.2.5 for error codes.

2.2.2.1.2.1 BUFFERING_STOP

It MUST be 0x00000001. If there is no more data in the stream or the buffers are full, the **extender device** sends this event to the host. In this case, the error code value SHOULD be 0x00000000.

2.2.2.1.2.2 END_OF_MEDIA

It MUST be 0x00000002. The **extender device** uses this event to indicate the end of stream. In this case, the error code value SHOULD be 0x00000000.

2.2.2.1.2.3 RTSP_DISCONNECT

It MUST be 0x00000003. The **extender device** will send this event to the host if the **extender device** received an error over RTSP while receiving data, or if it failed to send a response message. In this case, the error code value SHOULD be 0x00000000.

2.2.2.1.2.4 PTS_ERROR

It MUST be 0x00000005. If the current presentation time is behind the previous time by a value greater than 200 ms, or if the current presentation is ahead of the previous time by a value greater than 2000 ms, then the **extender device** sends this event. In this case, the error code value SHOULD be 0x00000000.

2.2.2.1.2.5 UNRECOVERABLE_SKEW

It MUST be 0x00000006. If the decoder takes more than 500 ms to open, or if the difference between audio's presentation time and video's presentation time is greater than 3500 ms for the first sample after seek, then the **extender device** sends this event. In this case, the error code value SHOULD be 0x00000000.

2.2.2.1.2.6 DRM_LICENSE_ERROR

It MUST be 0x0000000B. If the **extender device** doesn't support playback of a given protected content, then it sends this event. In this case, the error code value SHOULD be 0x00000000.

2.2.2.1.2.7 DRM_LICENSE_CLEAR

It MUST be 0x0000000E. The **extender device** sends this event to the host to clear out its license error. In this case, the error code value SHOULD be 0x00000000.

2.2.2.1.2.8 DRM_HDCP_ERROR

It MUST be 0x0000000F. If the **extender device** gets notification for High-bandwidth Digital Content Protection (HDCP) and it does not support HDCP, then the extender device sends this event to the host. See [MS-DRMND] for error code values.

2.2.2.1.2.9 FIRMWARE_UPDATE

It MUST be 0x00000011. If the **extender device** needs a new codec pack to download to play the current **media** file, it SHOULD send this event. The error code value is as follows.

Value	Meaning
E_FIRMWARE_UPDATE_REQUIRED 0x80099702	If the extender device does not have a required codec to play the **media** file, it sends this error saying a new codec pack needs to be downloaded.
E_H264_CODECPACK_REQUIRED 0x80099703	If the extender device does not have the H.264 codec to play the **media** file, it sends this error saying the H.264 codec pack needs to be downloaded.

3 Protocol Details

3.1 Extender Device Details

An **extender device** is a server for the Media Control Service and a client for the Media Event Callback Service.

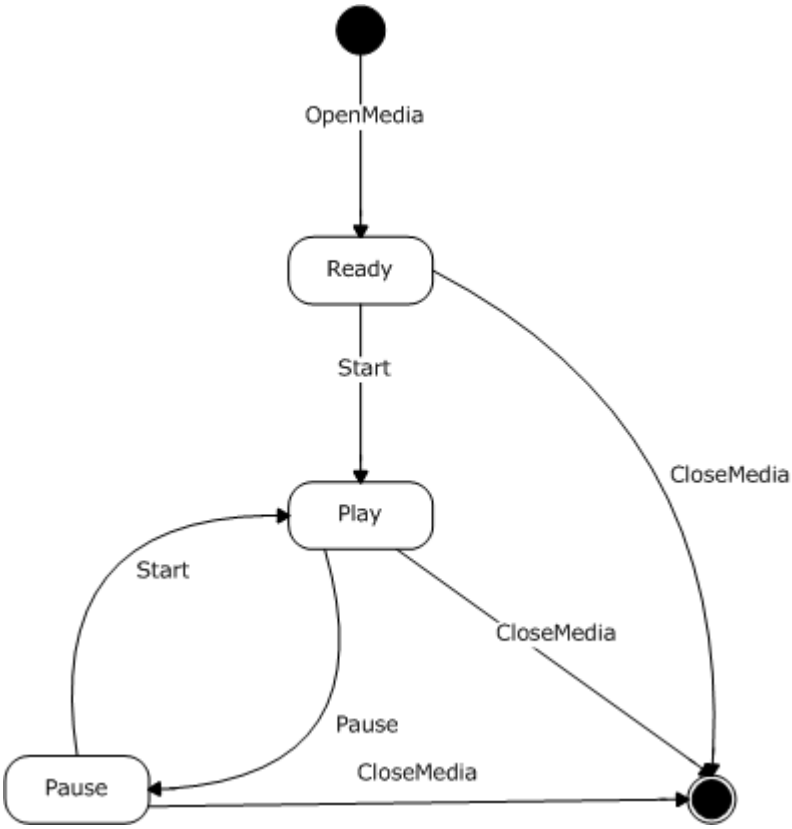


Figure 3: Extender device state machine diagram

Start State: The device is ready to start a new session. The following events are processed in this state:

1. RegisterMediaEventCallback
2. OpenMedia

The Extender Device can send the following events to the host:

- FIRMWARE_UPDATE

Ready state: The device is ready to start playing the content. The following event is processed in this state:

1. Start
2. CloseMedia

It can send the following events to the host:

1. UNRECOVERABLE_SKEW
2. DRM_LICENSE_CLEAR
3. DRM_HDCP_ERROR
4. DRM_HDCP_CLEAR
5. RTSP_DISCONNECT
6. DRM_LICENSE_ERROR

Play State: The device is playing the content. The following event is processed in this state:

1. Pause
2. CloseMedia

It can send the following events to the host:

1. END_OF_MEDIA
2. RTSP_DISCONNECT

Pause State: The device is in a paused state. The following event is processed in this state:

1. Start
2. CloseMedia

Finish State: The device MUST close the current session. No event is processed in this state. However, before closing, the device processes UnRegisterMediaEventCallback.

3.1.1 Abstract Data Model

This section describes a conceptual model of possible data organization that an implementation maintains to participate in this protocol. The described organization is provided to facilitate the explanation of how the protocol behaves. This document does not mandate that implementations adhere to this model as long as their external behavior is consistent with that described in this document.

The Device Media Control Protocol maintains the following data fields for each session:

URL: This variable is used to store the URL string of the current **media**.

State: This variable is used to store the current state of the **extender device**.

Duration: This variable is used to store the duration of the current **media**.

Position: This variable is used to store the current position of the **media**.

SurfaceId: This variable is used to store the current render surface.

PlayRate: This variable is used to store the current playrate of **media** that is supported by the **extender device**.

AvailableBandwidth: This variable is used to store available bandwidth.

OptimizedPrerollFlag: If this variable is true, then do optimized preroll; else normal preroll.

3.1.2 Timer

None.

3.1.3 Initialization

Before the Device Media Control Protocol takes action, the Device Services Lightweight Remoting Protocol MUST be started on both sides, and a CreateService call for the Media Controller Service received by the server.

3.1.4 Higher-Layer Triggered Events

None.

3.1.5 Processing Events and Sequencing Rules

3.1.5.1 OpenMedia

When the **extender device** gets this event, it does the following:

1. Check the input URL. If the file does not exist, send error E_FILE_NOT_FOUND.
2. If the **extender device** has not closed the previous **media** item, call the CloseMedia() function internally to close it.
3. Establish a connection to the **media** server by doing a TCP or UDP connection. If the **extender device** takes more than the number of seconds specified in the *timeout* parameter to establish a connection, then send E_RTSP_NO_CONNECTION error.
4. If the extender device uses RTSP streaming, send an RTSP DESCRIBE command to the server. And then send RTSP SETUP to select streams.
5. Check that the **media** server supports the **media** type of a given **media**. If it doesn't support the media type, send error E_UNSUPPORTED_STREAM_TYPE.
6. Create audio and video decoders of the **media** type for a given **media**. If the **extender device** does not have decoders for that **media** type, send error E_MDM_STREAM_TYPE_NOT_SUPPORTED. If the **extender device** needs new decoders to play the content, send error E_FIRMWARE_UPDATE_REQUIRED or E_H264_CODECPACK_REQUIRED, depending on the **media** type.
7. Initialize the downstream rendering pipeline.

If the **extender device** succeeds in doing all steps, send S_OK to the host.

3.1.5.2 CloseMedia

When the **extender device** receives this event, it closes the **media** session by clearing the downstream rendering pipeline and decoders. It releases the connection to the **media** server. If the extender device is using RTSP streaming, then send an RTSP:TEARDOWN command to the data server.

If the **extender device** successfully closes the **media** session, it returns S_OK; else it returns an error code.

3.1.5.3 Start

The **extender device** receives this event to start streaming **media** samples. The **extender device** does the following:

1. If the **extender device** is in 'Pause' state and the start time is 0xFFFFFFFFFFFFFFFF, it will resume from pause.
2. Ask the **media** server to start playback. If RTSP streaming is used, the extender device sends an RTSP:PLAY command.
3. Check that the **media** server supports the requested playRate, and update the Granted Rate. accordingly
4. If the 'Use Optimized preroll' variable is enabled, then do fast prerolling (prerolling data with a predefined threshold); otherwise do normal pre-rolling (preroll the data up to the level defined in the file).
5. If this event is for seek and the difference between audio's presentation time and video's presentation time is greater than the threshhold for first sample, then the **extender device** sends UNRECOVERABLE_SKEW.
6. Check the DRM license, if the **media** is protected content . If the extender device failed to get a DRM license, then send event DRM_LICENSE_ERROR with a DRM error code.

If the **extender device** successfully starts the **media** session, it returns S_OK; else it returns an error code.

3.1.5.4 Pause

The **extender device** receives this event to pause the streaming **media** session. It pauses the downstream rendering pipeline. If using RTSP streaming, the extender device sends an RTSP:PAUSE command to the server.

If the **extender device** successfully pauses the **media** session, it returns S_OK; else it returns an error code.

3.1.5.5 GetDuration

The **extender device** receives this event to send the duration of **media**.

If the **extender device** successfully pauses the **media** session, it returns S_OK; else it returns an error code.

3.1.5.6 GetPosition

The **extender device** receives this event to send the current position of **media**.

If the **extender device** successfully pauses the **media** session, it returns S_OK; else it returns an error code.

3.1.5.7 RegisterMediaEventCallback

The **extender device** receives this event to create and connect the Media Event Callback Service between the **extender device** and the host and to get events from the **extender device** to the host.

If the **extender device** successfully pauses the **media** session, it returns S_OK; else it returns an error code.

3.1.5.8 UnRegisterMediaEventCallback

The **extender device** receives this event to disconnect and release the current Media Event Callback Service between the **extender device** and the host.

If the **extender device** successfully pauses the **media** session, it returns S_OK; else it returns an error code.

3.1.6 Timer

None.

3.1.7 Other Local Events

None.

3.2 Host Details

A host or a device is a client for the Media Control Service and a server for the Media Event Callback Service.

The host is responsible for controlling the **media** session by sending events using the Media Control Service and taking action on events received using the Media Event Callback Service.

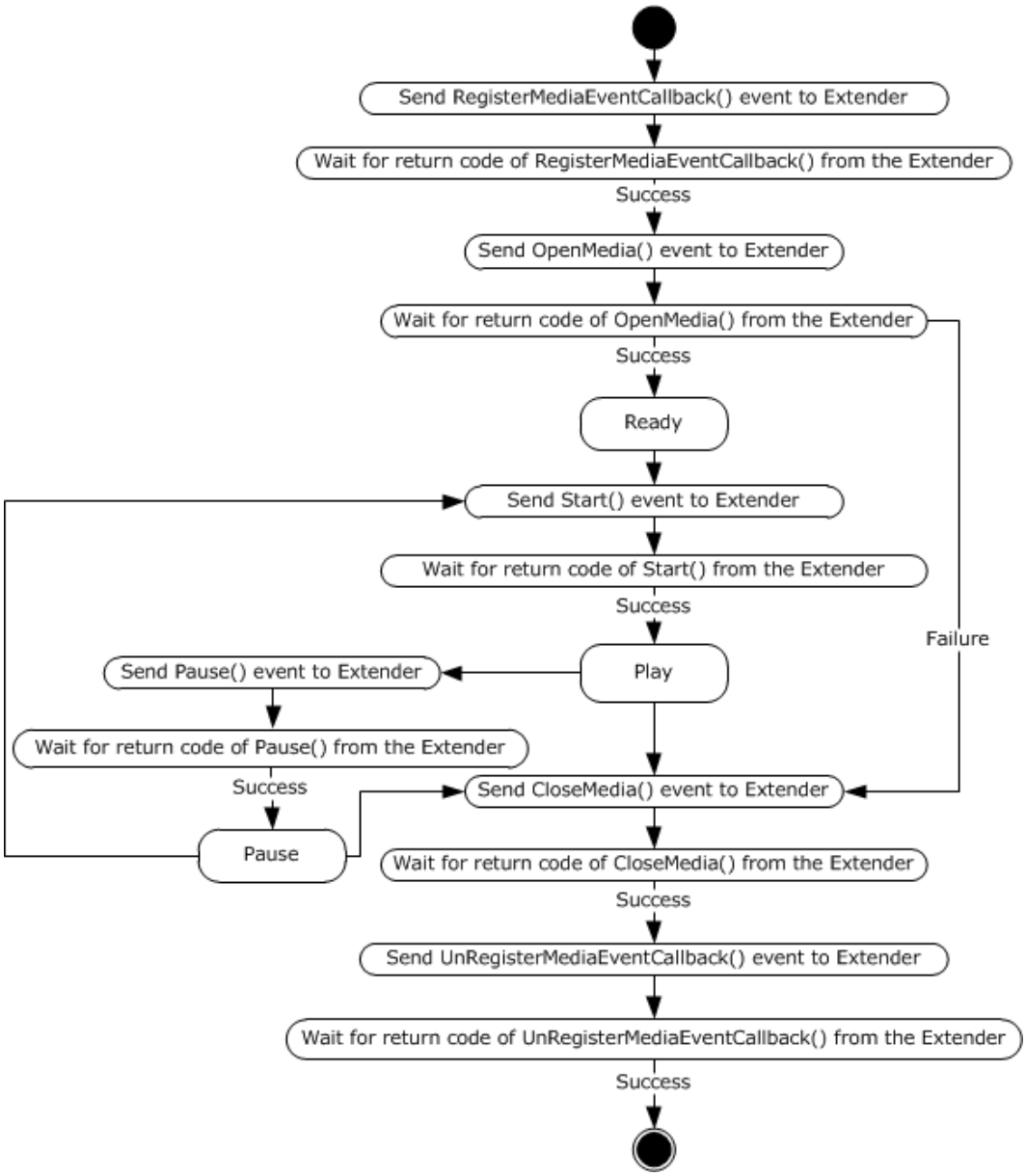


Figure 4: Host machine state diagram

3.2.1 Abstract Data Model

The Device Media Control Protocol maintains the following data fields for each session:

URL: This variable is used to store the URL string of the current **media**.

State: This variable is used to store the current state of the **extender device**.

Duration: This variable is used to store the duration of the current **media**.

Position: This variable is used to store the current position of **media**.

SurfaceId: This variable is used to store the current render surface.

PlayRate: This variable is used to store the current playrate of **media** that is supported by the **extender device**.

AvailableBandwidth: This variable is used to store available bandwidth.

OptimizedPrerollFlag: If this variable is true, then do optimized preroll; else normal preroll.

3.2.2 Timer

None.

3.2.3 Initialization

Before the Device Media Control Protocol takes action, the Device Services Lightweight Remoting Protocol MUST be started on both sides, and the Media Event Callback service is created by calling a RegisterMediaEventCallback message over the Media Control Service.

3.2.4 Higher-Layer Triggered Events

None.

3.2.5 Processing Events and Sequencing Rules

3.2.5.1 END_OF_MEDIA

When the host gets this event, it stops the playback and resets the stream position. Upon a query for the current position, the host returns the full duration of the **media**. It does the following depending on the host's current state:

1. If the host receives this event while closing the player, the host ignores the end-of-stream (EOS).
2. If host receives this event while rewinding to the beginning of the stream, then it suppresses this call and starts playing with regular 1x playback again.
3. If it is end of **media** in the middle of the **playlist**, then it continues to play the next item; else it stops the playback.

3.2.5.2 RTSP_DISCONNECT

The host will close down the session if it is in the 'Ready' or 'Play' state when it receives this event. But the host cannot close the device, because the RTSP session is already torn down. It resets its local state.

3.2.5.3 UNRECOVERABLE_SKEW

When the host receives this event, it tries to recover from skew by a force seek of 10 ms forward, which is unnoticeable.

If a new event occurred less than 1000 ms after the previous one, then the host ignores that event.

If the host is trying to recover from skew for more than 15000 ms, then it treats that file as corrupted. Therefore the host stops recovering and aborts playback and displays an error saying that the **extender device** cannot play the corrputed file.

3.2.5.4 DRM_LICENSE_ERROR

When the host receives this event, it displays an error page saying that the **extender device** does not support playback of this protected content.

3.2.5.5 DRM_LICENSE_CLEAR

The host clears out the license error when it receives this event.

3.2.5.6 DRM_HDCP_ERROR

The host displays an error saying that the **extender device** is connected to a device that does not support HDCP as required, when it receives this event.

3.2.5.7 DRM_HDCP_CLEAR

The host clears out the existing HDCP error states when it receives this event.

3.2.5.8 FIRMWARE_UPDATE

The host displays a message saying that a firmware update is required if it receives the error E_FIRMWARE_UPDATE_REQUIRED; or that a new H.264 codec pack is required if it received the error E_H264_CODECPACK_REQUIRED; and it stops the playback when receives this event.

If the events are unknown, the host ignores those events.

3.2.6 Timer Events

None.

3.2.7 Other Local Events

None.

4 Protocol Examples

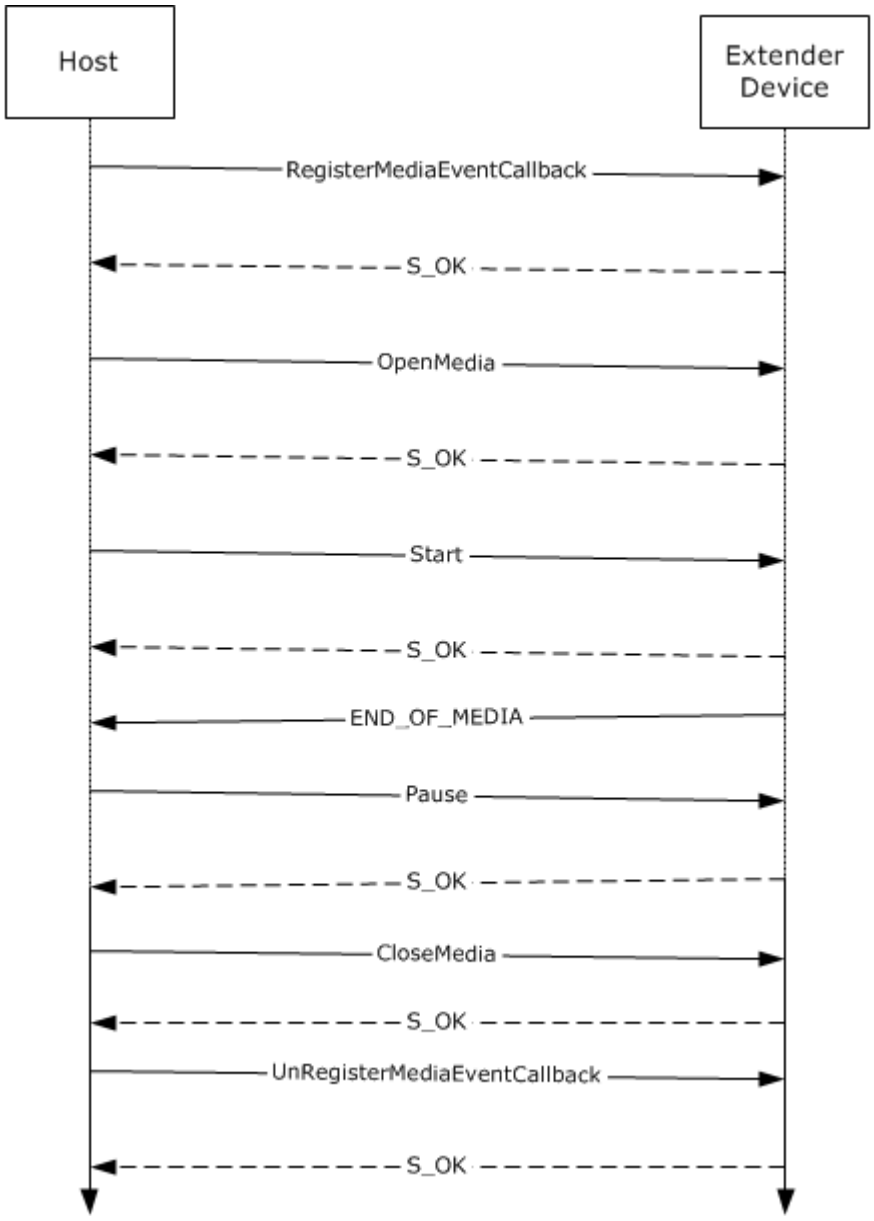


Figure 5: Sequence diagram for the Device Media Control Protocol

1. The host sends a RegisterMediaEventCallback event to the extender device to create the MediaEvent Callback Service. This event is sent while constructing the device and not per media session.

2. The device returns an S_OK code after successfully creating the MediaEvent Callback Service.

3. The host sends an OpenMedia message using the Media Controller Service over Device Services Lightweight Remoting Protocol to open a media session.

4. The device returns an S_OK code after successfully opening the media session.

5. The host sends a Start message using the Media Controller Service over the Device Services Lightweight Remoting Protocol to start a **media** session.

6. The device returns an S_OK code after successfully starting a media session.

7. When the device reaches the end of stream while playing, it sends an END_OF_MEDIA event to the host using the MediaEvent Callback Service over the Device Services Lightweight Remoting Protocol.

8. The host sends a Pause message using the Media Controller Service over the Device Services Lightweight Remoting Protocol to pause the media session.

9. The device returns an S_OK code after successfully pausing the media session.

10. The host sends a CloseMedia message using the Media Controller Service over the Device Services Lightweight Remoting Protocol to close the media session.

11. The device returns an S_OK code after successfully closing the media session.

12. The host sends an UnRegisterMediaEventCallback event to the extender device to release the MediaEvent Callback Service. This event is released while destructing the device and not per media session.

13. The device returns an S_OK code after successfully releasing the MediaEvent Callback Service.

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 Vista operating system
- Windows 7 operating system
- Windows 8 operating system
- Windows 8.1 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 2.2.1.1.2: Media Center on Windows Vista does not handle E_FIRMWARE_UPDATE_REQUIRED and E_H264_CODECPACK_REQUIRED error codes of OpenMedia message.

<2> Section 2.2.1.7.2: Media Center on Windows Vista does not handle a FIRMWARE_UPDATE event sent from the extender device on the MediaEvent Callback Service.

7 Change Tracking

No table of changes is available. The document is either new or has had no changes since its last release.

8 Index