[MS-DSMN]:

Device Session Monitoring Protocol

Revision Summary

Date	Revision History	Revision Class	Comments
11/6/2009	0.1	Major	First Release.
12/18/2009	0.1.1	Editorial	Changed language and formatting in the technical content.
1/29/2010	1.0	Major	Updated and revised the technical content.
3/12/2010	1.0.1	Editorial	Changed language and formatting in the technical content.
4/23/2010	1.0.2	Editorial	Changed language and formatting in the technical content.
6/4/2010	2.0	Major	Updated and revised the technical content.
7/16/2010	3.0	Major	Updated and revised the technical content.
8/27/2010	4.0	Major	Updated and revised the technical content.
10/8/2010	4.0	None	No changes to the meaning, language, or formatting of the technical content.
11/19/2010	4.0	None	No changes to the meaning, language, or formatting of the technical content.
1/7/2011	4.0	None	No changes to the meaning, language, or formatting of the technical content.
2/11/2011	4.0	None	No changes to the meaning, language, or formatting of the technical content.
3/25/2011	4.0	None	No changes to the meaning, language, or formatting of the technical content.
5/6/2011	4.0	None	No changes to the meaning, language, or formatting of the technical content.
6/17/2011	4.1	Minor	Clarified the meaning of the technical content.
9/23/2011	4.1	None	No changes to the meaning, language, or formatting of the technical content.
12/16/2011	5.0	Major	Updated and revised the technical content.
3/30/2012	5.0	None	No changes to the meaning, language, or formatting of the technical content.
7/12/2012	5.0	None	No changes to the meaning, language, or formatting of the technical content.
10/25/2012	5.0	None	No changes to the meaning, language, or formatting of the technical content.
1/31/2013	5.0	None	No changes to the meaning, language, or formatting of the technical content.
8/8/2013	6.0	Major	Updated and revised the technical content.
11/14/2013	7.0	Major	Updated and revised the technical content.
2/13/2014	7.0	None	No changes to the meaning, language, or formatting of the technical content.

Date	Revision History	Revision Class	Comments
5/15/2014	7.0	None	No changes to the meaning, language, or formatting of the technical content.
6/30/2015	7.0	None	No changes to the meaning, language, or formatting of the technical content.
10/16/2015	7.0	None	No changes to the meaning, language, or formatting of the technical content.
7/14/2016	7.0	None	No changes to the meaning, language, or formatting of the technical content.
6/1/2017	7.0	None	No changes to the meaning, language, or formatting of the technical content.

Table of Contents

1 Introduction

The Device Session Monitoring Protocol (DSMN) enables a client device to monitor the status of the host in a remote session. DSMN is built on the Device Services Lightweight Remoting Protocol (DSLR), as specified in [MS-DSLR].

Sections 1.5, 1.8, 1.9, 2, and 3 of this specification are normative. All other sections and examples in this specification are informative.

1.1 Glossary

This document uses the following terms:

big-endian: Multiple-byte values that are byte-ordered with the most significant byte stored in the memory location with the lowest address.

Component Object Model (COM): An object-oriented programming model that defines how objects interact within a single process or between processes. In **COM**, clients have access to an object through interfaces implemented on the object. For more information, see [MS-DCOM].

little-endian: Multiple-byte values that are byte-ordered with the least significant byte stored in the memory location with the lowest address.

Quality Windows Audio/Video Experience (qWAVE): A part of the Link Layer Topology Discovery (LLTD) Quality of Service (QoS) extension for Internet streaming media.

remote procedure call (RPC): A context-dependent term commonly overloaded with three meanings. Note that much of the industry literature concerning RPC technologies uses this term interchangeably for any of the three meanings. Following are the three definitions: (*) The runtime environment providing remote procedure call facilities. The preferred usage for this meaning is "RPC runtime". (*) The pattern of request and response message exchange between two parties (typically, a client and a server). The preferred usage for this meaning is "RPC exchange". (*) A single message from an exchange as defined in the previous definition. The preferred usage for this term is "RPC message". For more information about RPC, see [C706].

shell: Part of the Windows user interface (UI) that organizes and controls user access to a wide variety of objects necessary for running applications and managing the operating system. The most numerous are the folders and files that reside on computer storage media. There are also a number of virtual objects such as network printers and other computers. The **shell** organizes these objects into a hierarchical namespace and provides an API to access them.

terminal services (TS): A service on a server computer that allows delivery of applications, or the desktop itself, to various computing devices. When a user runs an application on a terminal server, the application execution takes place on the server computer and only keyboard, mouse, and display information is transmitted over the network. Each user sees only his or her individual session, which is managed transparently by the server operating system and is independent of any other client session.

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

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

[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

1.2.2 Informative References

[MS-LLTD] Microsoft Corporation, "Link Layer Topology Discovery (LLTD) Protocol".

1.3 Overview

After a network-connected device establishes a remote session with a host PC, the users on the remote device can interact with the **shell** running on the host. The purpose of DSMN is for the remote device to monitor the shell status on the host. For this, DSMN provides a **remote procedure call (RPC)**-like interface via which the host can notify the shell status to the remote host. In addition, the host can retrieve the **Quality Windows Audio/Video Experience (qWAVE)** information of the device via DSMN.

This protocol uses the Device Services Lightweight Remoting Protocol as specified in [MS-DSLR] to enable the remoting of services between the host and the client over a reliable point-to-point channel.

DSMN is implemented and offered by the remote device (acting in this case as the stub) while the host acts as the proxy, in DSLR nomenclatures. For a more detailed definition of these roles, see [MS-DSLR]. DSMN contains the following messages/functions:

- ShellDisconnect
- ShellIsActive
- Heartbeat
- GetQwaveSinkInfo

The following block diagram shows the relationship between the host and the device.<1>

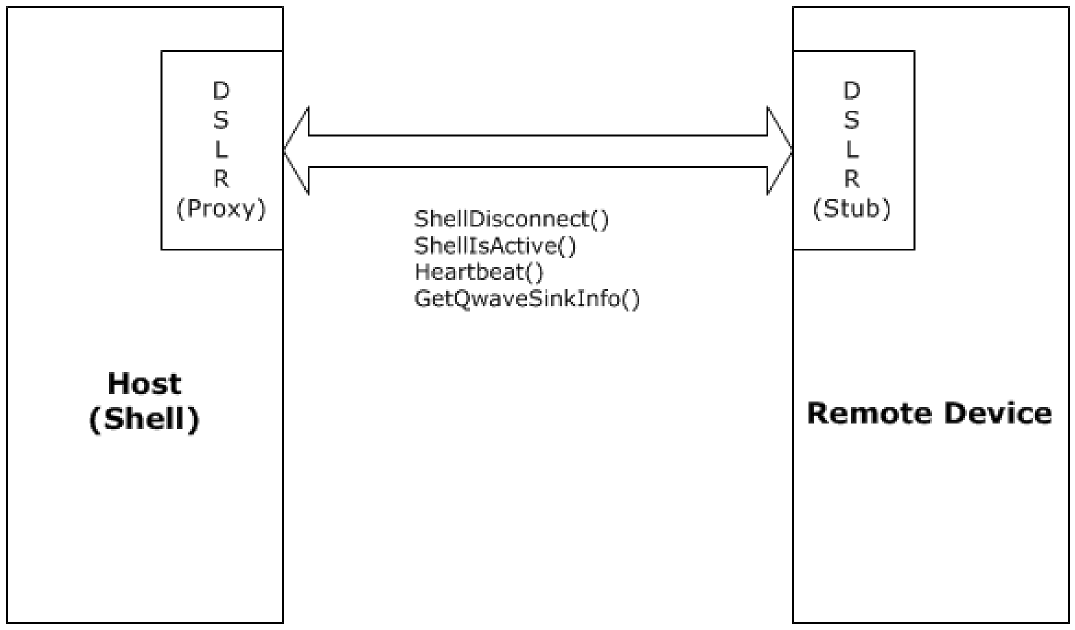

Figure 1: Architecture and operation of the DSMN Protocol

1.4 Relationship to Other Protocols

DSMN uses the Device Services Lightweight Remoting Protocol (DSLR) for transport. DSMN can be also used to retrieve **qWAVE** information from the device.

1.4.1 Device Services Lightweight Remoting Protocol (DSLR)

Device Services Lightweight Remoting Protocol (DSLR) is a **Component Object Model (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. DSLR 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/events exposed by the service, as well as the in/out parameters for each. By convention, the request/response calling convention follows COM rules:

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

For more information about this protocol, see [MS-DSLR].

1.4.2 Quality Windows Audio/Video Experience (qWAVE)

qWAVE provides the functionality for socket-based applications to gather in-depth, real-time information of a variable bandwidth network allowing it to dynamically adapt to changing network conditions. It also allows applications to prioritize packets in order to make better use of the available bandwidth.

The functionality provided by qWAVE is mainly targeted for use by multimedia applications that require network QoS for streams on a home network.

A PC or device must implement the Link Layer Topology Discovery (LLTD) Responder Protocol [MS-LLTD] with QoS extensions for qWAVE in order to expose advanced functionality, such as bandwidth measurements and congestion notifications.<2>

1.5 Prerequisites/Preconditions

For DSMN to function properly, the following conditions must be met:

- A network connection has been established between the host and the remote device.
- The DSLR modules have been initialized and started on both devices. Once completed, the proxy side calls the **CreateService** request to instantiate the service on the stub side, and creates 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 would subsequently be used when calling functions on the service and to terminate the service via **DeleteService**. See [MS-DSLR] section 3.1.5.1 and section 3.2.5.1 for more information on this process. The following class/service GUIDS are passed in the CreateService ([MS-DSLR] section 2.2.2.3) message for DSMN:
    - **ClassID GUID**: a30dc60e-1e2c-44f2-bfd1-17e51c0cdf19.
    - **ServiceID GUID**: 73e8f48c-033c-4590-a59f-fb844eb24681.

1.6 Applicability Statement

DSMN is suitable for a device running a remote session which needs to be informed of the shell status. Additionally, DSMN can be used where it is necessary for the host to query **qWAVE** information from the remote device, although the operation of DSMN does not require qWAVE/LLTD.

1.7 Versioning and Capability Negotiation

DSMN does not specify versioning and capability negotiation beyond what is specified by [MS-DSLR].

1.8 Vendor-Extensible Fields

None.

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 DSLR, which can be implemented on top of any stream-based or message-based reliable transport.

2.2 Message Syntax

The DSMN messages MUST follow the DSLR message syntax for requests and responses, as specified in [MS-DSLR] section 2.2. DSLR uses a tag-based formatting for its messages, see [MS-DSLR] section 2.2 for details of the tag formats.

The DSLR payload for a request is defined by the DSLR Dispatcher Request tag payload, followed by the child payload of a given message (that is, the function parameters for the given message). The Dispatcher 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 [MS-DSLR] section 2.2.2.1 for the format of the DSLR Dispatcher Request tag payload.

The DSLR payload for a response is defined by the DSLR Dispatcher Response tag payload, followed by the child payload of a given message (that is, the result and return parameters for the given message). The Dispatcher Response tag payload includes the callingConvention parameter and the matching one-time request handle to which this response corresponds. See [MS-DSLR] section 2.2 for the format of the DSLR 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 [MS-DSLR] section 2.2 for valid input/output parameters and how they are formatted on the wire (**big-endian** or **little-endian**).

For more details on the DSLR message syntax, see [MS-DSLR].

2.2.1 ShellDisconnect

The ShellDisconnect message is a two-way request message

2.2.1.1 ShellDisconnect (request)

The *CallingConvention* parameter in the Dispatcher Request tag MUST be dslrRequest (0x00000001). The function handle for the Dispatcher Request tag for ShellDisconnect MUST be 0x00000000.

Request payload (input parameter)

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
Disconnect Reason																															

Disconnect Reason (4 bytes): An unsigned 32-bit integer. This parameter MUST be the correct value from the following table.

Disconnect reason	Description
0	Shell exited unexpectedly.
1	Error is unknown. (This is deprecated.)
2	Initialization error.
3	Shell is not responding.
4	Unauthorized UI in the session.
5	User is not allowed - the remote device was disabled on the host.
6	Certificate is invalid.
7	Shell cannot be started.
8	Shell monitor thread cannot be started.
9	Message window cannot be created.
10	**Terminal Services** (TS) session cannot be started.
11	Plug-and-Play (PNP) failed.
12	Certificate is not trusted.
13	Product registration is expired.
14	PC goes to sleep or shuts down.
15	User closed the session.

2.2.1.2 ShellDisconnect (response)

The *CallingConvention* parameter in the Dispatch Response tag MUST be dslrResponse (0x00000002).

Response payload (result parameter)

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																															

Result (4 bytes): An unsigned 32-bit integer. HRESULT is returned from the function call.

2.2.2 ShellIsActive

The ShellIsActive message is a two-way request message.

2.2.2.1 ShellIsActive (request)

The *CallingConvention* parameter in the Dispatcher Request tag MUST be dslrRequest (0x00000001). The function handle for the Dispatcher Request tag for ShellIsActive MUST be 0x00000001. There is no input parameter for this message.

2.2.2.2 ShellIsActive (response)

The *CallingConvention* parameter in the Dispatch Response tag MUST be dslrResponse (0x00000002).

Response payload (result parameter)

										1										2										3	
0	1	2	3	4	5	6	7	8	9	0	1	2	3	4	5	6	7	8	9	0	1	2	3	4	5	6	7	8	9	0	1
Result																															

Result (4 bytes): An unsigned 32-bit integer. HRESULT is returned from the function call.

2.2.3 Heartbeat

The Heartbeat message is a two-way request message.

2.2.3.1 Heartbeat (request)

The *CallingConvention* parameter in the Dispatcher Request tag MUST be dslrRequest (0x00000001). The function handle for the Dispatcher Request tag for Heartbeat message MUST be 0x00000002.

Request payload (input parameter)

										1										2										3	
0	1	2	3	4	5	6	7	8	9	0	1	2	3	4	5	6	7	8	9	0	1	2	3	4	5	6	7	8	9	0	1
Screensaver Flag																															

Screensaver Flag (4 bytes): An unsigned 32-bit integer. The device MAY have its own screensaver. If so, the device's native screensaver MUST be controlled as specified in section 3.1.5.3 based on the value of the *Screensaver Flag* parameter whenever the device receives HeartBeat message.

2.2.3.2 Heartbeat (response)

The *CallingConvention* parameter in the Dispatch Response tag MUST be dslrResponse (0x00000002).

Response payload (result parameter)

										1										2										3	
0	1	2	3	4	5	6	7	8	9	0	1	2	3	4	5	6	7	8	9	0	1	2	3	4	5	6	7	8	9	0	1
Result																															

Result (4 bytes): An unsigned 32-bit integer. HRESULT is returned from the function call.

2.2.4 GetQWaveSinkInfo

The GetQWaveSinkInfo message is a two-way request message.

2.2.4.1 GetQWaveSinkInfo (request)

The *CallingConvention* parameter in the Dispatcher Request tag MUST be dslrRequest (0x00000001). The function handle for the Dispatcher Request tag for GetQWaveSinkInfo MUST be 0x00000003. There is no input parameter for this message.

2.2.4.2 GetQWaveSinkInfo (response)

The *CallingConvention* parameter in the Dispatch Response tag MUST be dslrResponse (0x00000002).

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																															
Is Sink Running																															
Port Number																															

Result (4 bytes): An unsigned 32-bit integer. HRESULT is returned from the function call.

Is Sink Running (4 bytes): An unsigned 32-bit integer. A nonzero value is returned if **qWAVE** is running on the device. Otherwise, 0 is returned.

Port Number (4 bytes): An unsigned 32-bit integer. The port number on which qWAVE runs is returned.<3>

3 Protocol Details

3.1 Server (Stub) Details

Built on DSLR, DSMN service (stub) resides in the remote device and processes the messages issued by the shell monitoring logic at the host (acting as DSMN client/proxy). The following figure illustrates the state transition diagram for DSMN.

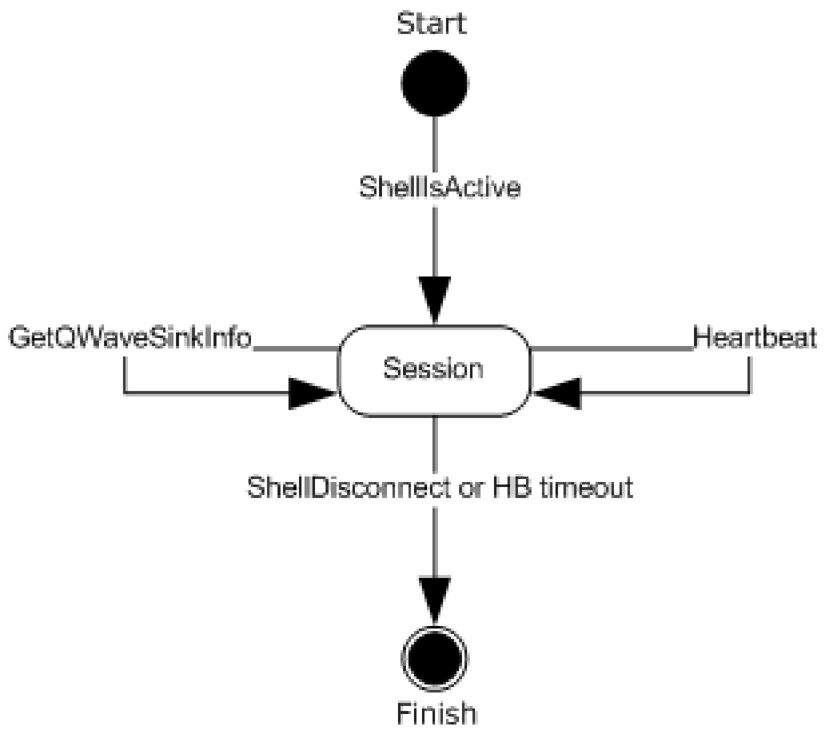

Figure 2: State diagram for DSMN

The states for DSMN can be summarized as follows:

Start: The device established a remote session and the shell is ready to start on the host. The following event is processed in this state:

- ShellIsActive

ShellRunning: The shell is up and running on the remote session. The following events are processed in this state:

- Heartbeat
- GetQWaveSinkInfo
- ShellDisconnect
- HB (heartbeat) timeout

Finish: The shell is closed. No event is processed in this state.

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.

IsNativeScreensaverOn: This flag is TRUE if the device has its own native screensaver and it is turned on. If the device does not have its own screensaver or it is not turned on, this flag is FALSE.

IsQWaveSinkRunning: If the device is currently running qWAVE, this flag is TRUE. Otherwise, this flag is FALSE.

QWaveSinkPortNumber: This is the port number on which qWAVE runs.

3.1.2 Timers

DSMN MUST maintain the following timer for each session:

HB timeout timer: The elapsed time from which the device received the last heartbeat from the host.

3.1.3 Initialization

Before DSMN takes action, DSLR MUST be started on both sides, and the CreateService message received by server.

3.1.4 Higher-Layer Triggered Events

None.

3.1.5 Processing Events and Sequencing Rules

3.1.5.1 ShellDisconnect

If this event occurs while the device is in the ShellRunning state, the device MUST move to the Finish state and return S_OK. Otherwise, the device MAY ignore this message.<4>

3.1.5.2 ShellIsActive

If this event occurs while the device is in the Start state, the device moves into the ShellRunning state and returns S_OK. Otherwise, the device returns an appropriate error code.<5> See [MS-DSLR] section 2.2.2.5 for the possible error codes returned in the DSLR Dispatcher Response Tag Payload.

3.1.5.3 Heartbeat

If this event occurs while the device is in the ShellRunning state, the device MUST reset the HB timeout timer (section 3.1.2). The device MUST return S_OK if the message is processed successfully. Otherwise, the device MUST return an appropriate error code.

The device MAY have its own native screensaver, of which state is represented by IsNativeScreensaverOn flag. If IsNativeScreensaverOn flag is TRUE and the value of the Screensaver Flag parameter is nonzero, the device MUST suppress the screensaver at the moment by disengaging any active native screensaver. Otherwise, the device MUST let the native screensaver run following the local settings of the device.

3.1.5.4 GetQWaveSinkInfo

If this event occurs while the device is in the ShellRunning state, the device MUST return IsQWaveSinkRunning and QWaveSinkPortNumber information in the Is Sink Running and Port Number parameters, respectively, as specified in section 2.2.4.

The device MUST return S_OK if the message is processed successfully. Otherwise, the device MUST return an appropriate error code. See [MS-DSLR] section 2.2.2.5 for the possible error codes returned in the DSLR Dispatcher Response Tag Payload.

3.1.6 Timer Events

3.1.6.1 HB Timeout Event

This represents the abnormal shell inactivity, which is originated from, not limited to, network disconnection or host shutdown. The device triggers this event if it cannot receive a Heartbeat message for 60 seconds after it received the last Heartbeat message from the host. When this event occurs, the device MUST move to the Finish state.<6>

3.1.7 Other Local Events

None.

4 Protocol Examples

The following diagram and accompanying list of steps show the sequence of DSMN messages that pass over the wire after the host and client establish a connection.

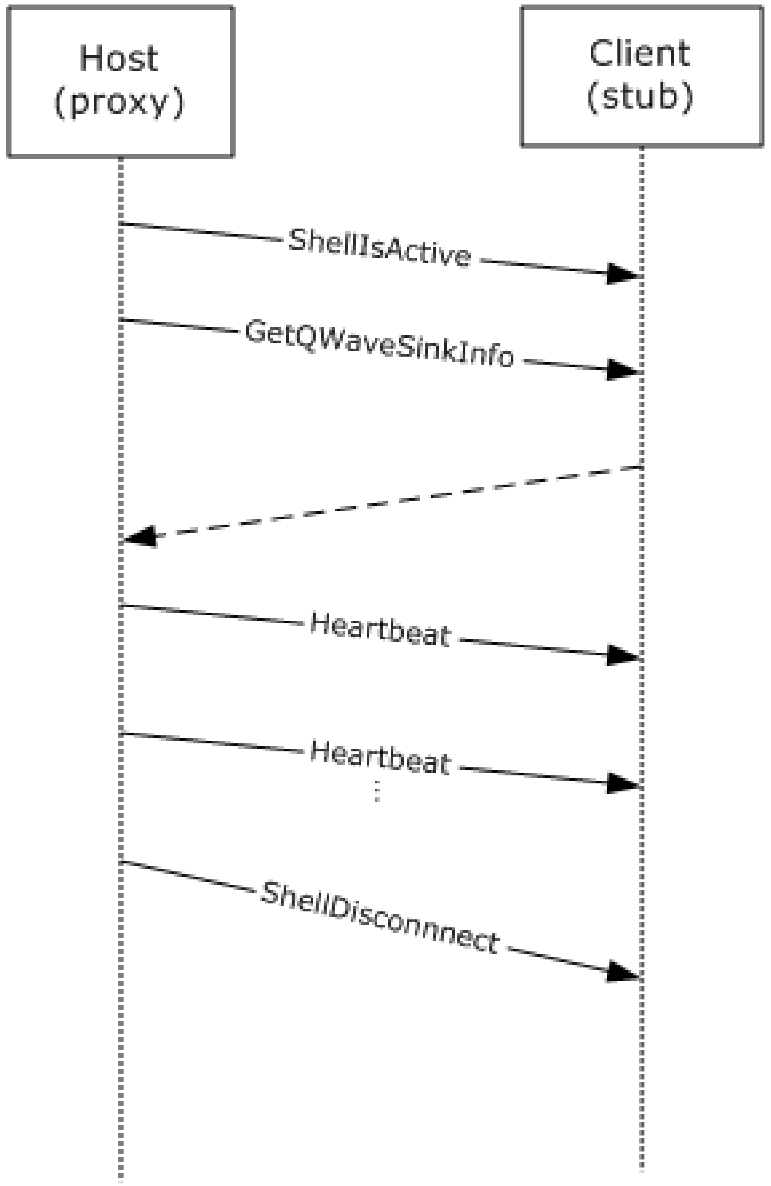

Figure 3: Typical DSMN message sequence during the lifetime of DSMN

1. The host notifies that the shell is now activated.
2. The host queries qWAVE sink information on the client.
3. The client returns qWAVE sink information in response.
4. The host starts sending Heartbeat messages, with the appropriate screensaver flag.
5. The host keeps sending Heartbeat messages before HB timeout timer is expired, continuing to control native screensaver behavior with the screensaver flag.

6. The host notifies the client of the shell closedown and the reason.

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 1.3: Windows Media Center is the shell, Windows is the host which runs the shell, and Extenders for Windows Media Center is the client device.

<2> Section 1.4.2: Windows supports LLTD by default.

<3> Section 2.2.4.2: Media Center uses port 2177 for qWAVE/LLTD with Extenders for Windows Media Center.

<4> Section 3.1.5.1: Media Center on Windows Vista, Windows 7, Windows 8, and Windows 8.1 sends ShellDisconnect when Media Center shell exits.

<5> Section 3.1.5.2: Media Center on Windows Vista, Windows 7, Windows 8, and Windows 8.1 sends ShellIsActive when Media Center shell is initialized and ready to start UI.

<6> Section 3.1.6.1: Media Center on Windows Vista sends Heartbeat to the device every 5 seconds. Media Center on Windows 7, Windows 8, and Windows 8.1 sends Heartbeat to the device every 5 seconds and whenever Screensaver flag is set to 0x00000001.

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