[MS-DRMRI]:

Windows Media Digital Rights Management for Network Devices (WMDRM-ND): Registrar Initiation 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	1.0.3	Editorial	Changed language and formatting in the technical content.
7/16/2010	1.0.3	None	No changes to the meaning, language, or formatting of the technical content.
8/27/2010	1.0.3	None	No changes to the meaning, language, or formatting of the technical content.
10/8/2010	1.0.3	None	No changes to the meaning, language, or formatting of the technical content.
11/19/2010	1.0.3	None	No changes to the meaning, language, or formatting of the technical content.
1/7/2011	1.0.3	None	No changes to the meaning, language, or formatting of the technical content.
2/11/2011	1.0.3	None	No changes to the meaning, language, or formatting of the technical content.
3/25/2011	1.0.3	None	No changes to the meaning, language, or formatting of the technical content.
5/6/2011	1.0.3	None	No changes to the meaning, language, or formatting of the technical content.
6/17/2011	1.1	Minor	Clarified the meaning of the technical content.
9/23/2011	2.0	Major	Updated and revised the technical content.
12/16/2011	3.0	Major	Updated and revised the technical content.
3/30/2012	3.0	None	No changes to the meaning, language, or formatting of the technical content.
7/12/2012	3.0	None	No changes to the meaning, language, or formatting of the technical content.
10/25/2012	3.0	None	No changes to the meaning, language, or formatting of the technical content.
1/31/2013	3.0	None	No changes to the meaning, language, or formatting of the technical content.
8/8/2013	4.0	Major	Updated and revised the technical content.
11/14/2013	4.0	None	No changes to the meaning, language, or formatting of the technical content.
2/13/2014	4.0	None	No changes to the meaning, language, or formatting of the

Date	Revision History	Revision Class	Comments
			technical content.
5/15/2014	4.0	None	No changes to the meaning, language, or formatting of the technical content.
6/30/2015	5.0	Major	Significantly changed the technical content.
10/16/2015	5.0	None	No changes to the meaning, language, or formatting of the technical content.
7/14/2016	6.0	Major	Significantly changed the technical content.
6/1/2017	6.0	None	No changes to the meaning, language, or formatting of the technical content.

Table of Contents

1 Introduction

This document describes the Windows Media Digital Rights Management for Network Devices (WMDRM-ND): Registrar Initiation Protocol, also known as DRMRI. This protocol is a set of services provided by a host (for example, a personal computer) and a client (for example, an extender device). These services allow a WMDRM-ND registration and authentication process to be remotely initiated and completed between the host and the client. The end result of this process is that DRM-**protected content** stored on the personal computer can ultimately be shared securely with the remote extender device. This protocol uses the Device Services Lightweight Remoting Protocol (DSLR) [MS-DSLR] to enable the remote initiation of the WMDRM-ND registrar process.

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.

binary large object (BLOB): A discrete packet of data that is stored in a database and is treated as a sequence of uninterpreted bytes.

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].

DSLR: Device Services Lightweight Remoting Protocol, as specified in [MS-DSLR]. A COM-like protocol that enables remoting of services, such as function calls and events, over a reliable point-to-point connection.

globally unique identifier (GUID): A term used interchangeably with universally unique identifier (UUID) in Microsoft protocol technical documents (TDs). Interchanging the usage of these terms does not imply or require a specific algorithm or mechanism to generate the value. Specifically, the use of this term does not imply or require that the algorithms described in [RFC4122] or [C706] must be used for generating the **GUID**. See also universally unique identifier (UUID).

host: A general-purpose computer that is networking capable.

HRESULT: An integer value that indicates the result or status of an operation. A particular HRESULT can have different meanings depending on the protocol using it. See [MS-ERREF] section 2.1 and specific protocol documents for further details.

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

message: A data structure representing a unit of data transfer between distributed applications. A message has message properties, which may include message header properties, a message body property, and message trailer properties.

payload: Tag-specific data sent as part of each DSLR message ([MS-DSLR]). Each DSLR **tag** contains one payload. Examples include Dispatcher Request tag payload ([MS-DSLR] section 2.2.2.1) (data identifying the type of request being made on the remote service), dispenser CreateService message payload ([MS-DSLR] section 2.2.2.3) (the parameters for the CreateService function), service-specific function payloads (the parameters for the service-specific functions), and so on.

protected content: Any content or information, such as a file, Internet message, or other object type, to which a rights-management usage policy is assigned and is encrypted according to that policy. See also Information Rights Management (IRM).

proximity detection: The procedure in which a transmitter determines if a receiver is near.

proxy: Part of the Remoting Data Model. A **Proxy** forwards the invocations of Remote Methods from the client to the Server Object for execution. The **Proxy** contains the Request URI of the Server Object. For more information, see [MS-NRTP] section 3.1.1.

receiver: The node that is the receiver of the protocol stream.

server: An entity that transfers content to a client through streaming. A server might be able to do streaming on behalf of another server; thus, a server can also be a proxy. See [MS-WMLOG]

service: A SIP method defined by Session Initiation Protocol Extensions used by the client to request a service from the **server**.

stub: Used as specified in [C706] section 2.1.2.2. A **stub** that is used on the client is called a "client **stub**", and a **stub** that is used on the server is called a "server **stub**".

tag: The format of all Device Services Lightweight Remoting Protocol ([MS-DSLR]) messages includes the size of the payload, number of children, and the tag payload itself.

transmitter: A device that issues policy and transfers content to a receiver. An example of a transmitter is a digital media server.

WMDRM-ND: Windows Media Digital Rights Management for Network Devices. A protocol in the digital rights management (DRM) system that extends the reach of **protected content** to consumer electronic devices (such as digital media receivers) that are connected to transmitting devices (such as personal computers) over home Internet protocol (IP) networks.

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

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

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

1.3 Overview

The DRMRI protocol can be viewed as a set of services that are implemented on and offered by an extender device and a host computer, so that the host computer can remotely initiate a registration and authentication process between itself and the extender device. The result of this process is that **protected content** stored on the host computer can be shared with the remote extender device. This protocol uses the Device Services Lightweight Remoting Protocol (DSLR) [MS-DSLR] to enable the use of remote services between the two devices over a reliable point-to-point channel.

The DRMRI protocol consists of two services: the DRM **receiver** and the DRM transmitter.

The DRM receiver **service** is implemented on and offered by the extender device. In this case, in **DSLR** nomenclatures, the extender device acts as the DSLR stub/server, and the host computer acts as the DSLR proxy/client. See [MS-DSLR] for a more detailed definition of these roles. The DRM receiver service contains the following functions:

- **RegisterTransmitterService**: Creates and connects a new DSLR remote service between the extender device and the host computer. This function would in turn invoke the DSLR **CreateService** message to instantiate the DRM **transmitter** service and create a proxy for that service. For more information on the **CreateService** message, see [MS-DSLR] section 2.2.2.3.
- **UnregisterTransmitterService**: Disconnects and releases the current DSLR service between the extender device and the host computer. This function in turn invokes the DSLR message **DeleteService** to clean up and disengage the DRM transmitter service. For more information on the **DeleteService** message, see [MS-DSLR] section 2.2.2.4.
- **InitiateRegistration**: Initiates the **WMDRM-ND** registration process between the extender device and the host computer.
- **RegistrationResponseMessage**: Handles the registration response messages received from the host computer.

The DRM transmitter service is implemented and offered by the host computer, in this case acting as the **stub** while the extender device acts as the proxy. This service contains the following functions:

- **RegistrationRequestMessage**: Handles the registration request message received from the extender device.
- **RegistrationResponseResult**: Handles the registration response Result received from the extender device.

At any given time, the host computer or the extender device can act as a DSLR client (or "proxy" in DSLR terminology) that invokes the service remotely or as a DSLR **server** (or "stub") that performs the request. This document refers to either device as proxy or stub as appropriate, whereas the term "host" always refers to the host computer and the term "device" always refers to the extender device.

The following block diagram shows the relationship between the host computer and the extender device.

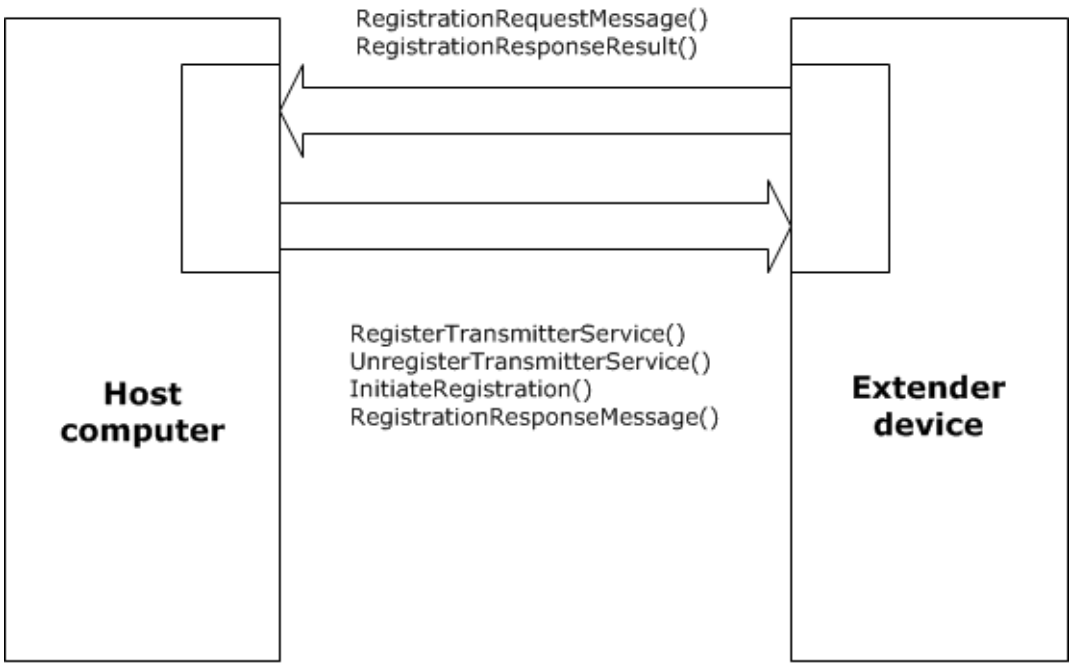


Figure 1: Relationship between the host and the device

1.4 Relationship to Other Protocols

The DRMRI protocol uses DSLR [MS-DSLR] to enable the remote initiation of the WMDRM-ND [MS-DRMND] registration process.

1.4.1 Device Services Lightweight Remoting Protocol (DSLR)

DSLR is a **COM**-like protocol that enables remoting of services, such as function calls and events, 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 as 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 and 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, and if successful, is followed by the [out] parameters.
- The caller can expect 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 Windows Media DRM for Network Devices (WMDRM-ND)

DRMRI depends on the WMDRM-ND protocol [MS-DRMND]. WMDRM-ND is a protocol in the Media digital rights management (DRM) system that extends the reach of **protected content** to consumer electronic devices (such as digital media receivers) that are connected to transmitting devices (such as personal computers) over home Internet protocol (IP) networks. WMDRM-ND enables these receivers to render protected content while enforcing the rights specified by the content owner.

The following diagram shows the relationship between the DRMRI protocol and other protocols and services.

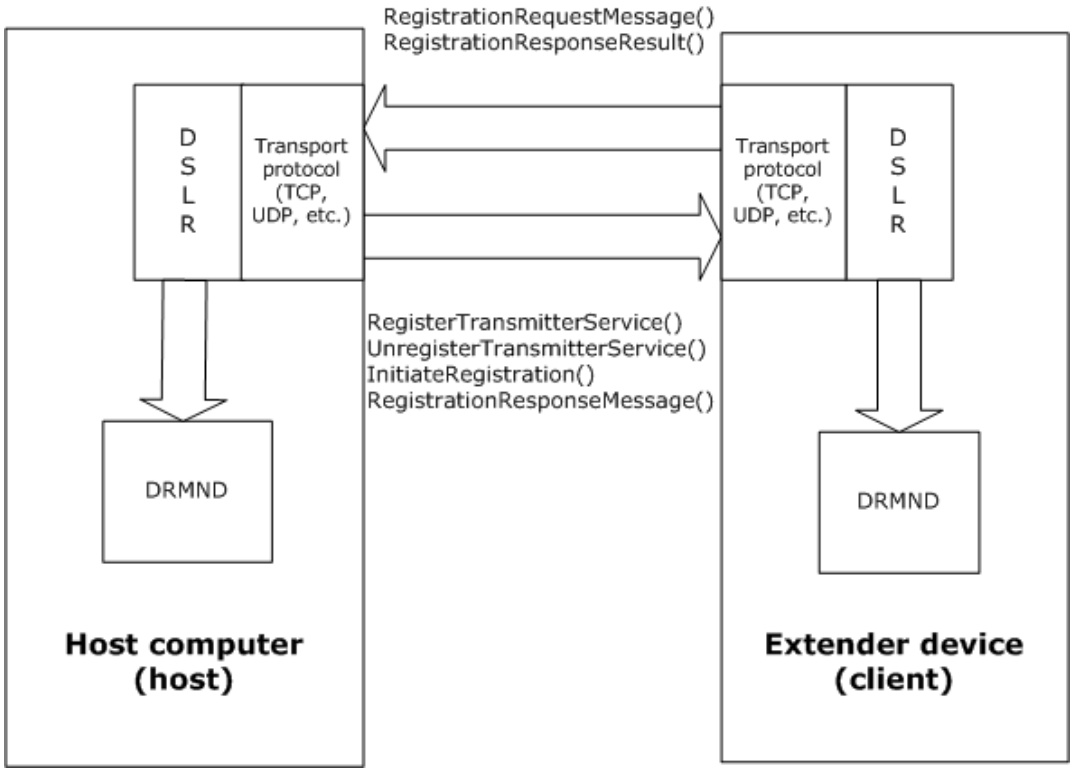


Figure 2: Relationship between DRMRI and other protocols

1.5 Prerequisites/Preconditions

For the DRMRI protocol to function properly, the following is assumed:

- A network connection has been established between the host and the device.
- The WMDRM-ND engines have been initialized and started on both devices. See [MS-DRMND] for more information on this process.
- The DSLR modules have been initialized and started on both the host and the device. Once completed, the proxy side calls **CreateService** to instantiate the service on the **stub** side and creates a proxy for that service. This proxy is an object that implements the interfaces of the service. As part of the **CreateService** request, it allocates a service handle that is sent to the stub side. This handle is subsequently used when calling functions on the service. For more information on module initialization, see [MS-DSLR].

1.6 Applicability Statement

The DRMRI protocol provides a mechanism for 2 remote devices to register, authenticate, and ultimately connect to share DRM-**protected content** over a point-to-point channel using DSLR and WMDRM-ND.

1.7 Versioning and Capability Negotiation

There are no versioning requirements beyond those specified in [MS-DSLR] section 1.7.

1.8 Vendor-Extensible Fields

The DRMRI protocol uses **HRESULT** values as described in [MS-DSLR] and [MS-DRMND], as well as error code values described in [MS-ERREF] section 2.1.

1.9 Standards Assignments

None.

2 Messages

2.1 Transport

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

2.2 Message Syntax

The DSLR protocol uses a **tag**-based format for its messages. See [MS-DSLR] for the tag formats.

The DRMRI messages MUST follow the DSLR message syntax specified in [MS-DSLR] section 2.2.

The DSLR **payload** sent for a request is defined by the DSLR dispatcher request tag payload followed by the child payload, comprising the function parameters for the given message. The request tag payload includes the following:

- The service handle for the specific service. See section 2.3 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 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 that is the result and return parameters for the given message. The response tag payload includes the *callingConvention* parameter and the matching one-time request handle to which this response corresponds. See section 2.2.1 for the format of the DSLR dispatcher response tag payload.

The format of the data types used as input and output parameters in the following functions are defined in [MS-DSLR]. See section 2.2.1.6 for valid input/output parameters and how they are formatted on the wire, including **big-endian** or **little-endian** byte order.

For more details on the DSLR message syntax, see [MS-DSLR] section 2.2.

The child payloads (that is, the input/output parameters) for specific services are described in the following sections.

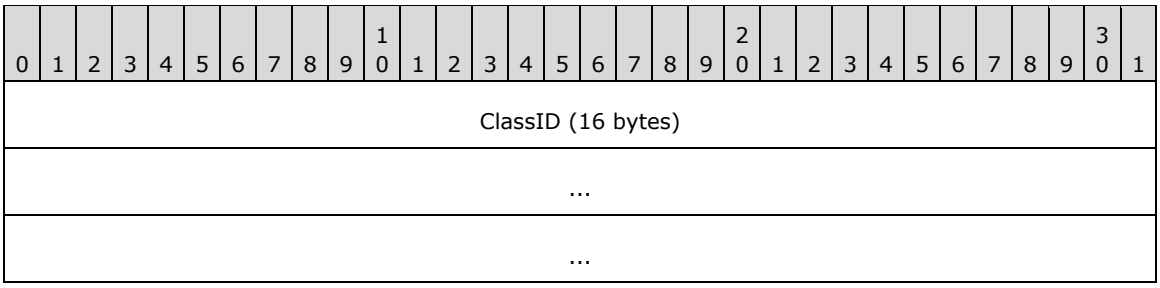
2.2.1 DRM Receiver Service

For this service, the proxy is defined in the host, and the **stub** is in the device. The host invokes the following functions using the proxy, and the device carries out the actions.

2.2.1.1 RegisterTransmitterService Request

The **RegisterTransmitterService** request **message** contains the input parameters of a 2-way request message. The *callingConvention* parameter in the dispatch request tag MUST be set to dslrRequest (0x00000001). The function handle for the dispatch request tag MUST be set to 0x00000000.

The request payload has the following format.

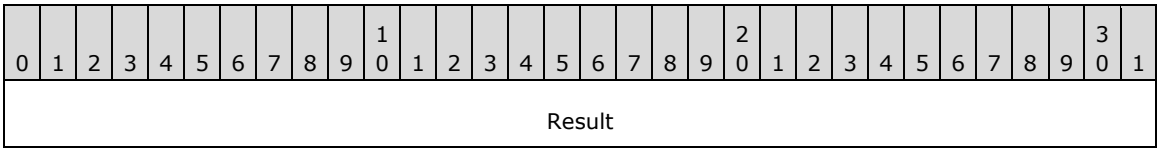


ClassID (16 bytes): 128-bit **GUID** value, as specified in [MS-DSLR] section 2.2.2.6. For this service, the value MUST be **b707af79-ca99-42d1-8c60-469fe112001e**.

2.2.1.2 RegisterTransmitterService Response

The **RegisterTransmitterService** response packet contains the return value for the **RegisterTransmitterService** function. The *callingConvention* parameter in the dispatch response tag MUST be set to dslrResponse (0x00000002).

The response payload has the following format.

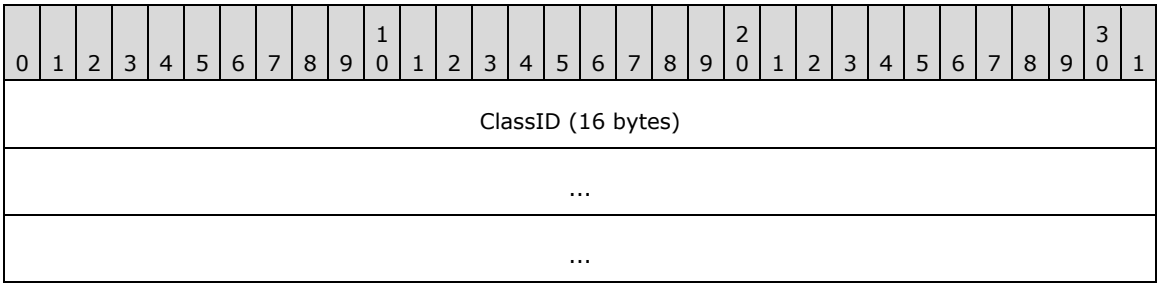


Result (4 bytes): An unsigned 32-bit integer, in big-endian byte order. This field contains the **HRESULT** returned by the function call. For error codes, see [MS-DSLR] and [MS-ERREF].

2.2.1.3 UnregisterTransmitterService Request

The **UnregisterTransmitterService** request message contains the input parameters of a 2-way request message. The *callingConvention* parameter in the dispatch request tag MUST be set to dslrRequest (0x00000001). The function handle for the dispatch request tag MUST be set to 0x00000001.

The request payload has the following format.

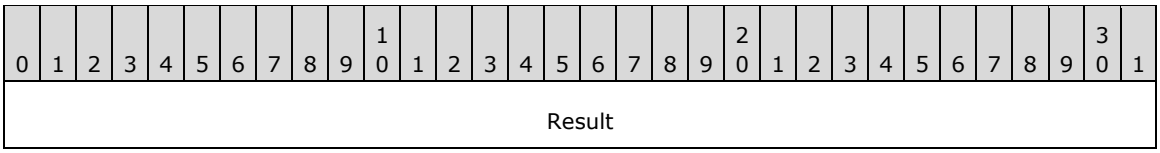


ClassID (16 bytes): 128-bit **GUID** value, as specified in [MS-DSLR] section 2.2.2.6. For this service, the value MUST be **b707af79-ca99-42d1-8c60-469fe112001e**.

2.2.1.4 UnregisterTransmitterService Response

The **UnregisterTransmitterService** response message contains the return value for the **UnregisterTransmitterService** function. The *callingConvention* parameter in the dispatch response tag MUST be set to dslrResponse (0x00000002).

The response payload has the following format.



Result (4 bytes): An unsigned 32-bit integer, in **big-endian** byte order. This field contains the **HRESULT** returned by the function call. For error codes, see [MS-DSLR] and [MS-ERREF].

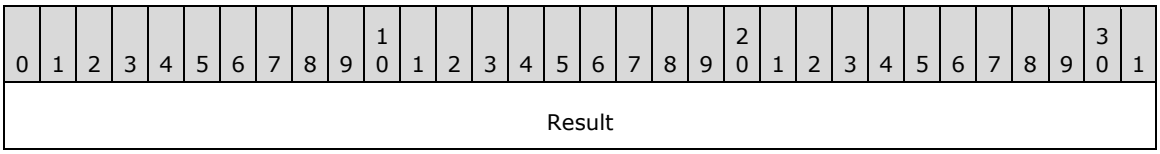
2.2.1.5 InitiateRegistration Request

The **InitiateRegistration** request is a dispatch request that initiates the 2-way request **message** for the **InitiateRegistration** function. There are no input parameters for this function. The *callingConvention* parameter in the dispatch request tag MUST be set to dslrRequest (0x00000001). The function handle for the dispatch request tag MUST be set to 0x00000002.

2.2.1.6 InitiateRegistration Response

The **InitiateRegistration** response message contains the return value for the **InitiateRegistration** function. The *callingConvention* parameter in the dispatch response tag MUST be set to dslrResponse (0x00000002).

The response payload has the following format.

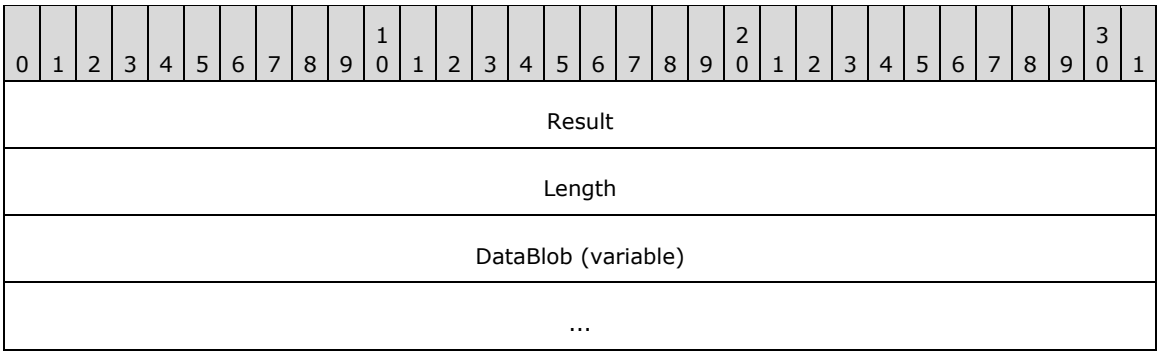


Result (4 bytes): An unsigned 32-bit integer, in **big-endian** byte order. This field contains the **HRESULT** returned by the function call. For error codes, see [MS-DSLR] and [MS-ERREF].

2.2.1.7 RegistrationResponseMessage Request

The **RegistrationResponseMessage** request **message** contains the input parameters for a 2-way request message. The *callingConvention* parameter in the dispatch request tag MUST be set to dslrRequest (0x00000001). The function handle for the dispatch request tag MUST be set to 0x00000003.

The request payload has the following format.



Result (4 bytes): An unsigned 32-bit integer, in **big-endian** byte order. HRESULT is returned from the previous operation. See [MS-ERREF] for the definitions of error codes.

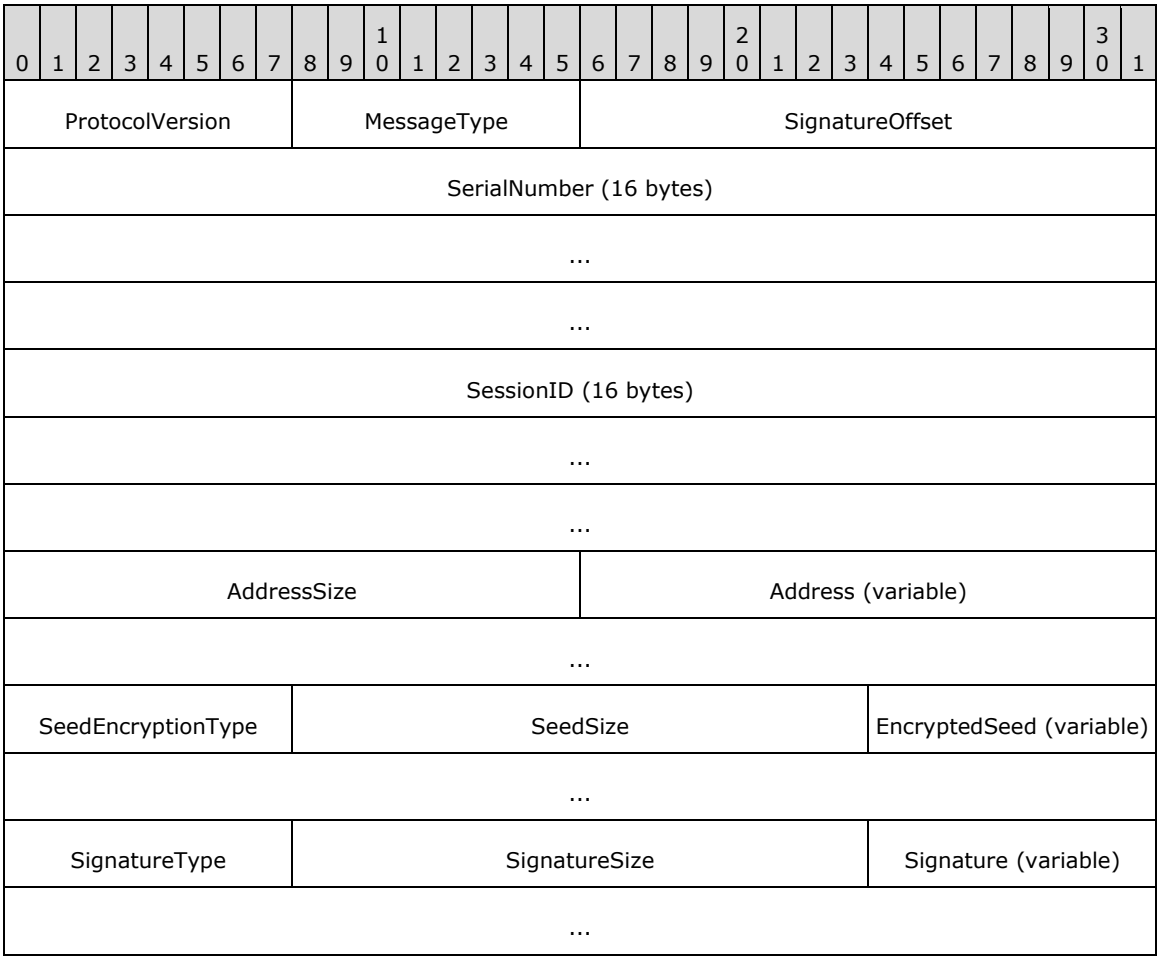
Length (4 bytes): An unsigned 32-bit integer, in big-endian byte order. Size of the following data blob.

DataBlob (variable): A byte array with length equal to the value of the Length field. This field contains a WMDRM-ND registration response message **blob**, as specified in section 2.2.1.7.1.

2.2.1.7.1 WMDRM-ND RegistrationResponseMessage Blob

This section defines the **RegistrationResponseMessage** data **blob**.

For more details on message formats of the WMDRM-ND protocol, refer to [MS-DRMND].



ProtocolVersion (1 byte): An unsigned 8-bit integer. Version number of the WMDRM-ND protocol. This field MUST be set to 0x02.

MessageType (1 byte): An unsigned 8-bit integer. WMDRM-ND message type. This field MUST be set to 0x02.

SignatureOffset (2 bytes): An unsigned 16-bit integer, in **little-endian** byte order. Offset (in bytes), from the first byte of this packet to the **Signature** section of the blob.

SerialNumber (16 bytes): An unsigned 128-bit integer, in little-endian byte order.

SessionID (16 bytes): An unsigned 128-bit integer, in little-endian byte order.

AddressSize (2 bytes): An unsigned 16-bit integer, in little-endian byte order. This is the size of the Address field.

Address (variable): Consists of "AddressSize" bytes of data, in little-endian byte order.

SeedEncryptionType (1 byte): An unsigned 8-bit integer. WMDRM-ND seed encryption type. This field MUST be set to 0x01 to indicate RSAES-OAEP encryption (for more details, see [MS-DRMND]).

SeedSize (2 bytes): An unsigned 16-bit integer, in little-endian byte order. This is the size of the **EncryptedSeed** field.

EncryptedSeed (variable): Consists of "SeedSize" bytes of data, in little-endian byte order.

SignatureType (1 byte): An unsigned 8-bit integer. WMDRM-ND signature type. This field MUST be set to 0x01 to indicate the AES OMAC1 signature type (for more details, see [MS-DRMND]).

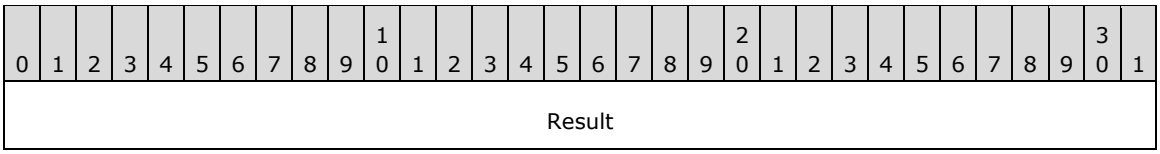
SignatureSize (2 bytes): An unsigned 16-bit integer, in little-endian byte order. This is the size of the **Signature** field.

Signature (variable): Consists of "SignatureSize" bytes of data, in little-endian byte order.

2.2.1.8 RegistrationResponseMessage Response

The **RegistrationResponseMessage** message contains the return value for the **RegistrationResponseMessage** function. The *callingConvention* parameter in the dispatch response tag MUST be set todslrResponse (0x00000002).

The response payload contains the return value and has the following format.



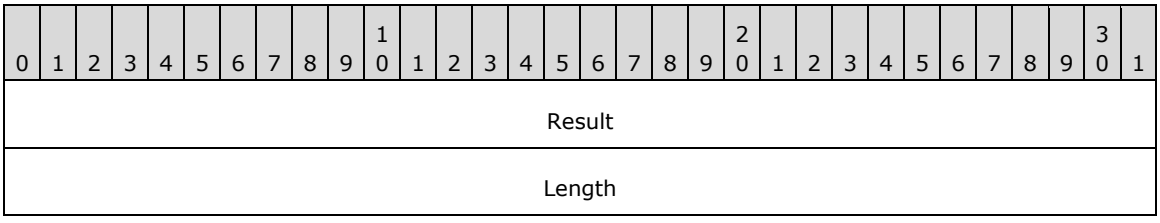
Result (4 bytes): An unsigned 32-bit integer, in **big-endian** byte order. This field contains the **HRESULT** returned by the function call. For error codes, see [MS-DSLR] and [MS-ERREF].

2.2.2 DRM Transmitter Service

2.2.2.1 RegistrationRequestMessage Request

The **RegistrationRequestMessage** request message contains the input parameters for a 2-way request message, so the *callingConvention* parameter in the dispatch request tag MUST be dslrRequest (0x00000001). The function handle for the dispatch request tag MUST be 0x00000000.

The request payload has the following format.



DataBlob (variable)
...

Result (4 bytes): An unsigned 32-bit integer, in **big-endian** byte order. **HRESULT** is returned from the previous operation. See [MS-DRMND] and [MS-ERREF] for error codes.

Length (4 bytes): An unsigned 32-bit integer, in big-endian byte order. Size of the following data blob.

DataBlob (variable): A byte array with length equal to the value of the **Length** field. This field contains a WMDRM-ND registration request message blob, as specified in section 2.2.2.1.1.

2.2.2.1.1 WMDRM-ND RegistrationRequestMessage Blob

This section defines the **RegistrationRequestMessage**.

For more detailed information on message formats of the WMDRM-ND protocol, refer to [MS-DRMND].

The format of this data **blob** is as follows.

<table>
<tr><td>0</td><td>1</td><td>2</td><td>3</td><td>4</td><td>5</td><td>6</td><td>7</td><td>8</td><td>9</td><td>10</td><td>1</td><td>2</td><td>3</td><td>4</td><td>5</td><td>6</td><td>7</td><td>8</td><td>9</td><td>20</td><td>1</td><td>2</td><td>3</td><td>4</td><td>5</td><td>6</td><td>7</td><td>8</td><td>9</td><td>30</td><td>1</td></tr>
<tr><td colspan="8">ProtocolVersion</td><td colspan="8">MessageType</td><td colspan="16">SerialNumber (16 bytes)</td></tr>
<tr><td colspan="32">...</td></tr>
<tr><td colspan="32">...</td></tr>
<tr><td colspan="16">...</td><td colspan="16">DeviceCertificateSize</td></tr>
<tr><td colspan="32">DeviceCertificate (variable)</td></tr>
<tr><td colspan="32">...</td></tr>
</table>

ProtocolVersion (1 byte): An unsigned 8-bit integer. Version number of the WMDRM-ND protocol. This field MUST be set to 0x02.

MessageType (1 byte): An unsigned 8-bit integer. WMDRM-ND message type. This field MUST be set to 0x01.

SerialNumber (16 bytes): An unsigned 128-bit integer, in **little-endian** byte order.

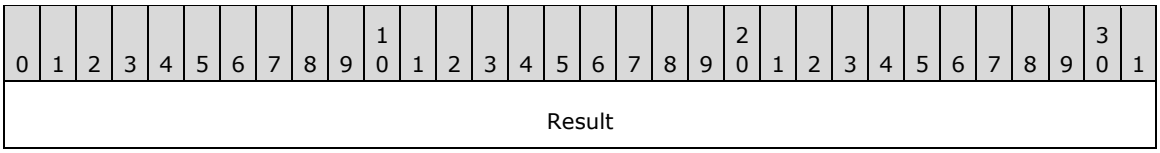
DeviceCertificateSize (2 bytes): An unsigned 16-bit integer, in little-endian byte order. This is the size of the following device certificate.

DeviceCertificate (variable): A little-endian byte array with length equal to the value of the **DeviceCertificateSize** field.

2.2.2.2 RegistrationRequestMessage Response

The **RegistrationRequestMessage** response payload contains the return value for the **RegistrationRequestMessage**.

The response payload has the following format.

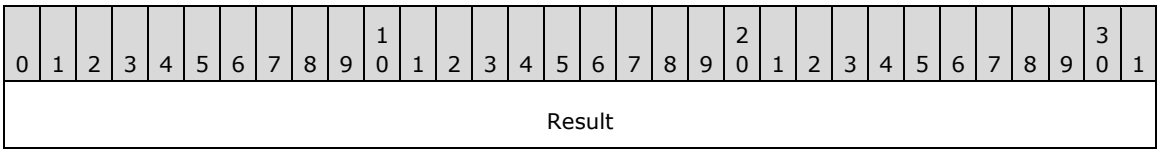


Result (4 bytes): An unsigned 32-bit integer, in **big-endian** byte order. This field contains the **HRESULT** returned by the function call. For error codes, see [MS-DSLR] and [MS-ERREF].

2.2.2.3 RegistrationResponseResult Request

The **RegistrationResponseResult** request message contains the result of the registration process. There are no input parameters for this function. The *callingConvention* parameter in the dispatch request tag MUST be dslrRequest (0x00000001). The function handle for the dispatch request tag MUST be 0x00000001

The request payload has the following format.

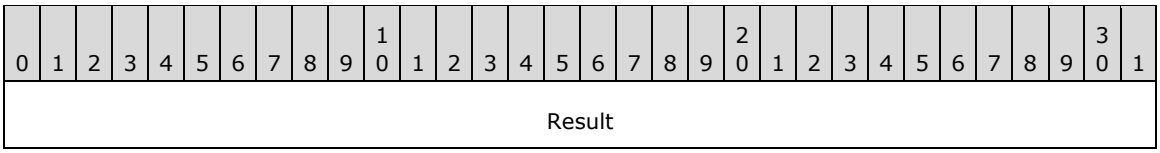


Result (4 bytes): An unsigned 32-bit integer, in **big-endian** byte order. **HRESULT** returned from the **Proximity Detection** operation. See [MS-DRMND] and [MS-DSLR] for error codes.

2.2.2.4 RegistrationResponseResult Response

The **RegistrationResponseResult** response payload contains the return value for the **RegistrationResponseResult** function.

The response payload has the following format.



Result (4 bytes): An unsigned 32-bit integer, in **big-endian** byte order. This field contains the **HRESULT** returned by the function call. This HRESULT represents the status of the function that returns the status of the registration response operation. For error codes, see [MS-DSLR] and [MS-ERREF].

2.3 CreateService/DeleteService

This section describes the use of the **CreateService** and **DeleteService** messages in the WMDRM-ND protocol. The **CreateService** and **DeleteService** messages are defined in [MS-DSLR] sections 2.2.2.3 and 2.2.2.4 respectively.

Before the DRM **transmitter** or DRM **receiver** service calls can be initiated, the DSLR client MUST invoke the **CreateService** message, which indicates to the DSLR **server** to create the respective service. At this time, the DSLR client allocates a service handle for the new service and passes it to the DSLR server via this call. This service handle is used in the dispatch request tag for all calls made on that service.

Consequently, the **DeleteService** message MUST be invoked by the DSLR client at cleanup time.

Note that with this DRMRI protocol, the device (the extender device) does not invoke **CreateService** (or **DeleteService**) directly by itself. Instead it will wait for the host computer to invoke this operation remotely (via the proxy function **RegisterTransmitterService** or **UnregisterTransmitterService**) as described in section 3.1.5.

2.3.1 DRM Receiver Service

The following class/service GUIDS are passed in the **CreateService** and **DeleteService** messages for this service:

ClassID GUID MUST be set to: **b707af79-ca99-42d1-8c60-469fe112001e**

ServiceID GUID MUST be set to: **8ef82607-9129-42f6-951c-9365ad68bdf7**

2.3.2 DRM Transmitter Service

The following class/service GUIDS are passed in the **CreateService** and **DeleteService** messages for this service:

ClassID GUID MUST be set to: **b707af79-ca99-42d1-8c60-469fe112001e**

ServiceID GUID MUST be set to: **acb96f70-e61f-45cb-9745-86c47dcbb156**

3 Protocol Details

At any given time, primary users of the DRMRI services, the host computer, or the extender device can act as a proxy and invoke the service remotely or act as a **stub** and perform the request, depending on the specific service implementation. The DRMRI services are symmetrical; all services MAY be implemented and offered on both the proxy side and the stub side.<1>

3.1 Device Details

The device MUST call **CreateService** or **DeleteService** routines to establish or disestablish the DRM **transmitter** service with the host. The device does not invoke these routines by itself and instead waits for the host to initiate the calls as described in section 3.1.5. Once the remote service has been established, the device is ready to start the registration process. This process is described in detail in section 3.1.5.

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 maintains the following state:

ServiceHandle: This variable stores the service handle of the DRM **transmitter** service. This handle is allocated by the device and sent to the host via the **CreateService** routine. It is then used by both the host and the device to uniquely identify the DRM transmitter service in subsequent remote function calls.

DRMTransmitterRemoteService: This variable contains the pointers to the proxy functions offered by the DRM transmitter service (**RegistrationRequestMessage** and **RegistrationResponseResult**). These proxies are allocated by the device after the DRM transmitter service has been successfully instantiated using the **CreateService** routine. The device uses these proxies to invoke functions remotely in the WMDRM-ND registration process.

The following information is not maintained by the device. This information is made available by the system, using implementation-specific methods, for the WMDRM-ND registration process to succeed.

Device Certificate: The unique signed device certificate that includes a 1024-bit RSA public key. This value is maintained by WMDRM-ND and stored in the non-volatile memory in the device hardware. See [MS-DRMND] for more detail.

Device ID: The unique serial number that has been assigned to the device. This value is stored in non-volatile memory on the device.

3.1.2 Timers

In the **Proximity Detection** operation, the **host** and the device exchange UDP messages in an attempt to determine the latency between them. If the latency is more than 7 milliseconds, the device is considered too far away from the host, causing this operation to fail. See [MS-DRMND] sections 2.2.1.3, 3.1.1.4, 3.2.5.2, and 3.3.5.2 for more details.

3.1.3 Initialization

The DSLR and WMDRM-ND services MUST be initialized and then started on the device. The device waits for the host to invoke **CreateService** or **DeleteService** remotely via the service **RegisterTransmitterService** or **UnregisterTransmitterService** as described in section 3.1.5.

3.1.4 Higher-Layer Triggered Events

None.

3.1.5 Processing Events and Sequencing Rules

3.1.5.1 RegisterTransmitterService

This function is invoked remotely by the host to establish the **transmitter** service between the host and the device. This function MUST be implemented on the device.

This operation will call **CreateService** to instantiate the DRM transmitter service provided by the host and create a proxy for that service. The DRM transmitter service includes the **RegistrationRequestMessage** and **RegistrationResponseResult** routines as specified in section 2.2.2. As part of the **CreateService** process, a service handle will be allocated and passed to the host. This handle will subsequently be used when calling these remote functions in the WMDRM-ND registration initiation process. For more details on **CreateService**, as well as the message syntax, refer to [MS-DSLR] sections 2.2.2.3 and 3.1.5.1.

The following figure depicts the state transition for the **RegisterTransmitterService** operation.

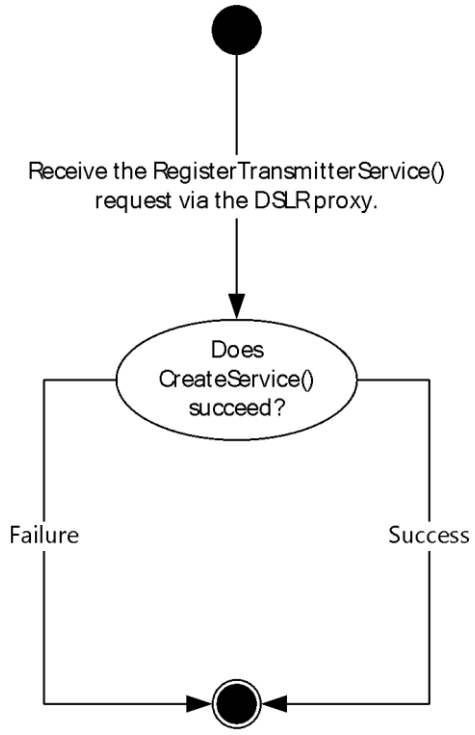


Figure 3: RegisterTransmitterService operation

3.1.5.2 UnregisterTransmitterService

The **UnregisterTransmitterService** function is invoked remotely by the host to disestablish the **transmitter** service between the host and the device. This function MUST be implemented on the device.

This function calls **DeleteService** to clean up DRM transmitter service provided by the host. For more details on **DeleteService**, as well as on the message syntax, refer to [MS-DSLR] section 2.2.2.4.

The following figure depicts the state transition for the **UnregisterTransmitterService** operation.

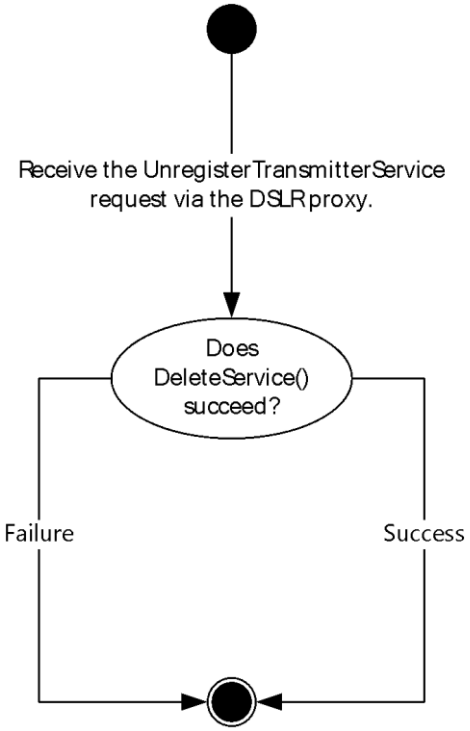


Figure 4: UnregisterTransmitterService operation

3.1.5.3 InitiateRegistration and RegistrationResponseMessage

Once the DRM transmitter service has been established between the **host** and device, the device is ready for the WMDRM-ND registration process. The registration process begins as soon as the device receives the InitiateRegistration request from the host. The flow of events is outlined in the following figure.

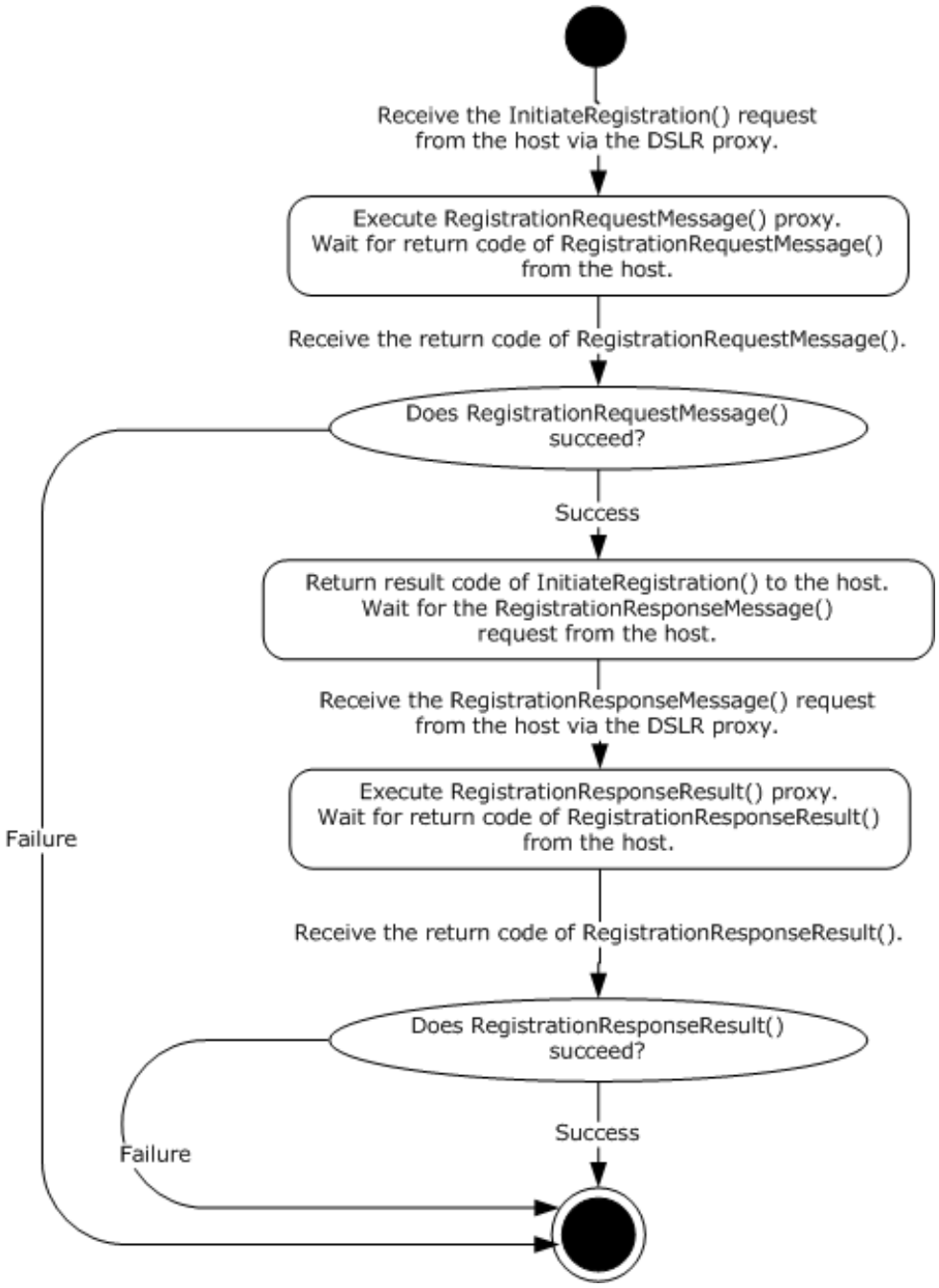


Figure 5: WMDRM-ND registrar process on the client side

3.1.5.3.1 InitiateRegistration Function

The device MUST perform the following steps to complete the **InitiateRegistration** function:

1. Collect the WMDRM-ND device certificate and the root device ID stored in the extender device.
2. Pack the certificate and device ID information into a DSLR request message **blob** (refer to [MS-DSLR] section 2.2.2.1 for the message syntax).

3. Invoke the **RegistrationRequestMessage** request using the remote function proxy on the host, passing in the message blob as an input parameter.

4. Wait for the **RegistrationRequestMessage** response with the return code from the host.

5. Return the final result code by sending the **InitiateRegistration** request to the host.

Once the **InitiateRegistration** function returns, the device waits for the next request from the host.

3.1.5.3.2 RegistrationResponseMessage function

After the function **InitiateRegistration** has been completed successfully, the device MUST perform the following steps to complete the **RegistrationResponseMessage** function:

1. Parse and verify the registration response message blob sent from the host as an input parameter.

2. Retrieve the address, session ID, and DRM data-encryption key for use in the proximity detection operation.

3. Open a UDP socket to the given address, generate a proximity start message based on the session ID, and send it to the UDP socket.

4. Wait for the proximity challenge message from the UDP socket.

5. Process the proximity challenge message, generate a corresponding proximity response message, and sends it back to the socket.

6. Wait for the proximity result message and process the return result.

7. Invoke the **RegistrationResponseResult** request using the remote function proxy on the host, passing in the final result as an input parameter.

See [MS-DRMND] for details on the WMDRM-ND process and message syntax.

3.1.6 Timer Events

None.

3.1.7 Other Local Events

None.

3.2 Host Details

The **host** initiates the WMDRM-ND registration process. The host first calls the DSLR **CreateService** or **DeleteService** routines on itself and on the device. This establishes or disestablishes all DRMRI services between the host and the device. Once the remote services are established, the host is ready to initiate the registration process. This process is described in detail in section 3.1.5.

3.2.1 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 host maintains the following state for each managed device:

ServiceHandle: Stores the service handle of the DRM **Receiver** service. The **ServiceHandle** is allocated by the host and sent to the device using the **CreateService** routine. It is then used by both the host and the device to uniquely identify the DRM receiver service for that device in subsequent remote function calls.

DRMReceiverRemoteService: This variable contains the pointers to the proxy functions offered by the DRM receiver service: **RegisterTransmitterService, UnregisterTransmitterService, InitiateRegistration,** and **RegistrationResponseMessage**. These proxies are allocated by the host after the DRM receiver service has been successfully instantiated by using the **CreateService** routine. The host then uses these proxies to invoke the functions remotely in the WMDRM-ND registration process.

RegisteredDevice: The variable to store the registered device interface for the remote device being managed. If the device is not registered, this variable is NULL.

3.2.2 Timers

There is a timer present in the **proximity detection** operation specified in section 3.1.2.

3.2.3 Initialization

The DSLR and WMDRM-ND services MUST be initialized and then started on the host. The **host** MUST call **CreateService** on itself as well as on the device in order to instantiate all DRMRI services between the host and the device. The host MUST call **DeleteService** on itself as well as on the device in order to disestablish all DRMRI services between the host and the device cleanup.

3.2.4 Higher-Layer Triggered Events

None.

3.2.5 Processing Events and Sequencing Rules

The following figure depicts message processing events and sequencing rules for the DRMRI protocol.

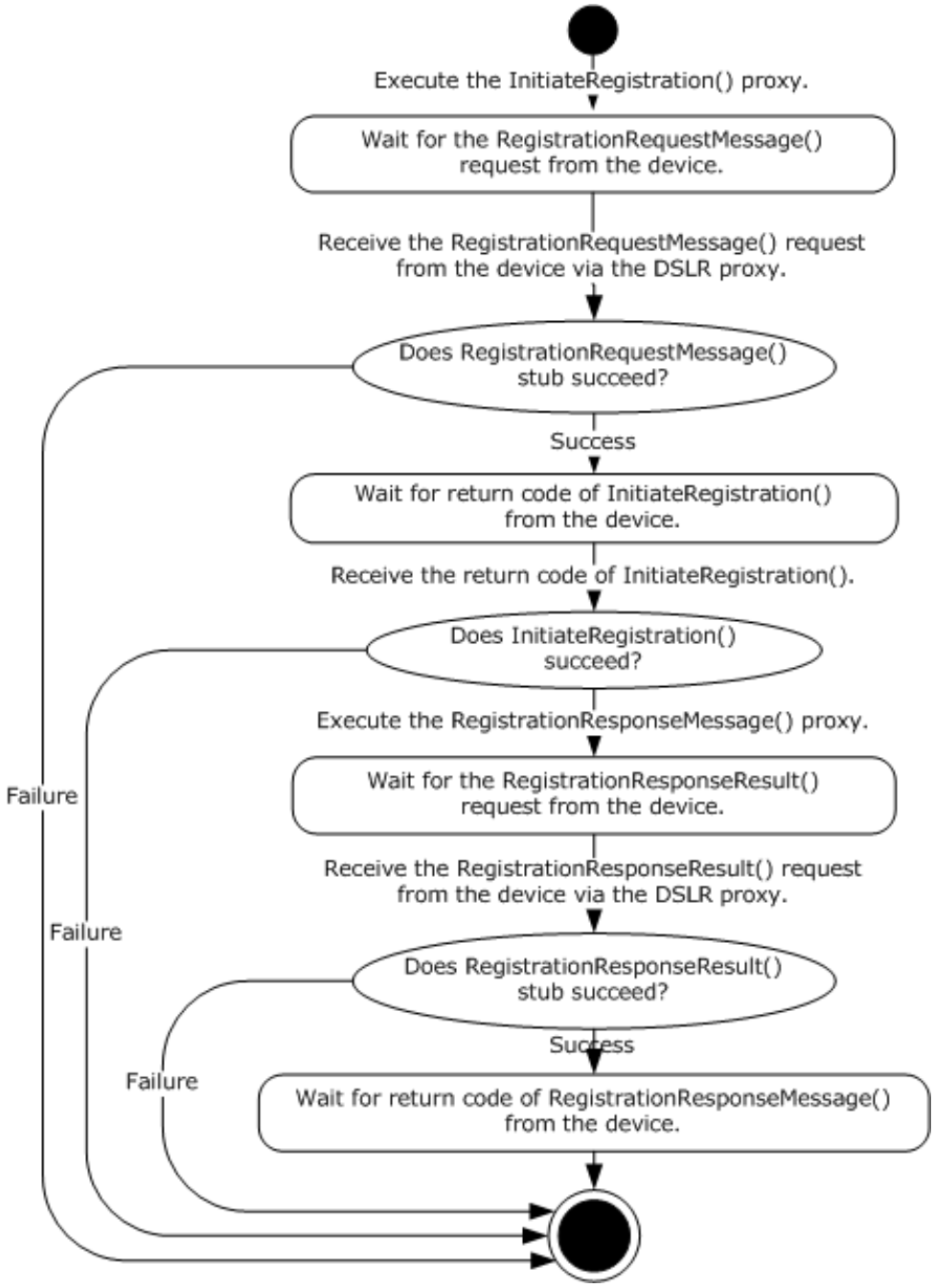


Figure 6: WMDRM-ND registrar process on the host side

3.2.5.1 RegistrationRequestMessage

Once the **RegistrationRequestMessage** request has been received, the **host** MUST perform the following actions to successfully complete the **RegistrationRequestMessage** remote function:

1. Parse and verify the registration request message blob.

2. Retrieve the device certificate and the root device ID, and register the device with these values.

3. Approve and open the device.

4. Once the device has been successfully opened, start the proximity detection process.

5. Invoke the **RegistrationResponseMessage** response using the remote **proxy**, passing in the result code as an input parameter.

3.2.5.2 RegistrationResponseResult

Once the **RegistrationResponseResult** request has been received, the host MUST perform the following actions to successfully complete the **RegistrationResponseResult** function:

1. Parse the message to get the result of the registration response request. This result indicates the status of the **Proximity Detection** process.

2. If the status indicates success, the WMDRM-ND registration is marked as complete. Otherwise, registration is left as pending, and another registration request message is required to kick off another Proximity Detection operation by invoking another RegistrationRequestMessage request to the host.

For details on the WMDRM-ND process and message syntax, refer to [MS-DRMND].

3.2.6 Timer Events

None.

3.2.7 Other Local Events

None.

4 Protocol Examples

In a typical registration scenario, events will be passed back and forth between the host and the device as illustrated in the following sequence diagram. In this diagram, note that the device that initiates the function calls becomes the DSLR proxy, and the device that performs the function calls acts as the DSLR stub.

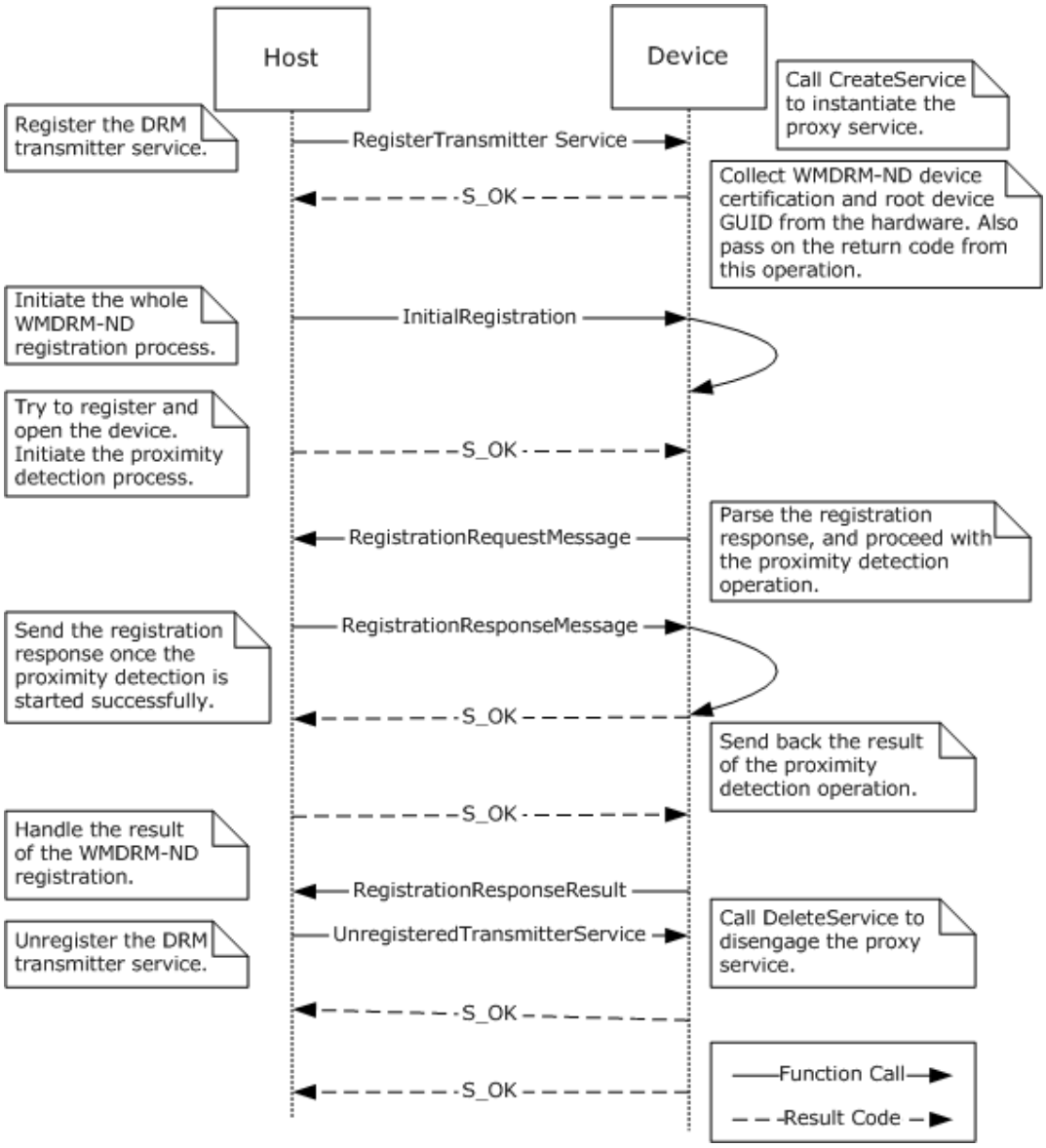


Figure 7: Typical registration procedure

1. At initialization the host calls **CreateService** on itself and the proxy, and calls **RegisterTransmitterService** on the device to instantiate all DRMRI services required for this protocol to function.

2. The device executes the **RegisterTransmitterService** **stub** and returns S_OK by using the DSLR proxy mechanism.

3. When both sides are ready for the WMDRM-ND operation, the host issues a call to the **InitiateRegistration** proxy.

4. The device collects the device certificate and device GUID, packages them into a **RequestMessage**, and calls the **RegistrationRequestMessage** proxy to the host.

5. The host executes the **RegistrationRequestMessage** stub and returns S_OK to the device.

6. The device returns S_OK as the status of the **InitiateRegistration** call to the host.

7. The host starts the Proximity Detection process and invokes the **RegistrationResponseMessage** proxy.

8. The device executes the **RegistrationResponseMessage** stub to parse the response and proceeds with the Proximity Detection operation.

9. The device sends the result of the Proximity Detection operation to the host using a call to the **RegistrationResponseResult** proxy.

10. The host executes the **RegistrationResponseResult** stub on its side, processes the result, and returns S_OK to the device.

11. The device returns S_OK to the host as the result of the **RegistrationResponseMessage** stub.

12. The WMDRM-ND registration process is now completed successfully. The host and the device can now share the **protected content**.

13. At cleanup time, the host issues a call to the **UnregisterTransmitterService** proxy.

14. The device calls **DeleteService** to end the DSLR service and return S_OK to the host.

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.

- Extenders for Windows Media Center
- 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 3: The Windows 7 Extenders for Windows Media Center, and Windows 8, and Windows 8.1 device implementations, implement some of the services asymmetrically; that is, some of the services are provided on one side but not on the other.

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

R

S

T

U

V

W