[MS-DRM]:

Digital Rights Management License Protocol

Intellectual Property Rights Notice for Open Specifications Documentation

- **Technical Documentation.** Microsoft publishes Open Specifications documentation ("this documentation") for protocols, file formats, data portability, computer languages, and standards support. Additionally, overview documents cover inter-protocol relationships and interactions.
- **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 can make copies of it in order to develop implementations of the technologies that are described in this documentation and can distribute portions of it in your implementations that use these technologies or in your documentation as necessary to properly document the implementation. You can also distribute in your implementation, with or without modification, any schemas, IDLs, or code samples that are included in the documentation. This permission also applies to any documents that are referenced in the Open Specifications documentation.
- **No Trade Secrets**. Microsoft does not claim any trade secret rights in this documentation.
- **Patents**. Microsoft has patents that might cover your implementations of the technologies described in the Open Specifications documentation. Neither this notice nor Microsoft's delivery of this documentation grants any licenses under those patents or any other Microsoft patents. However, a given Open Specifications document might be covered by the Microsoft Open Specifications Promise or the Microsoft Community Promise. If you would prefer a written license, or if the technologies described in this documentation are not covered by the Open Specifications Promise or Community Promise, as applicable, patent licenses are available by contacting iplg@microsoft.com.
- **License Programs**. To see all of the protocols in scope under a specific license program and the associated patents, visit the Patent Map.
- **Trademarks**. The names of companies and products contained in this documentation might 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 that are 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 as specifically described above, whether by implication, estoppel, or otherwise.

Tools. The Open Specifications documentation does 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 documents are intended for use in conjunction with publicly available standards specifications and network programming art and, as such, assume that the reader either is familiar with the aforementioned material or has immediate access to it.

Support. For questions and support, please contact dochelp@microsoft.com.

Revision Summary

Date	Revision History	Revision Class	Comments
5/11/2007	0.1	New	Version 0.1 release
8/10/2007	1.0	Major	Updated and revised the technical content.
9/28/2007	1.0.1	Editorial	Changed language and formatting in the technical content.
10/23/2007	1.0.2	Editorial	Changed language and formatting in the technical content.
11/30/2007	1.0.3	Editorial	Changed language and formatting in the technical content.
1/25/2008	1.0.4	Editorial	Changed language and formatting in the technical content.
3/14/2008	2.0	Major	Updated and revised the technical content.
5/16/2008	2.0.1	Editorial	Changed language and formatting in the technical content.
6/20/2008	2.0.2	Editorial	Changed language and formatting in the technical content.
7/25/2008	2.0.3	Editorial	Changed language and formatting in the technical content.
8/29/2008	2.0.4	Editorial	Changed language and formatting in the technical content.
10/24/2008	2.1	Minor	Clarified the meaning of the technical content.
12/5/2008	2.1.1	Editorial	Editorial Update.
1/16/2009	2.1.2	Editorial	Changed language and formatting in the technical content.
2/27/2009	2.2	Minor	Clarified the meaning of the technical content.
4/10/2009	3.0	Major	Updated and revised the technical content.
5/22/2009	4.0	Major	Updated and revised the technical content.
7/2/2009	5.0	Major	Updated and revised the technical content.
8/14/2009	5.0.1	Editorial	Changed language and formatting in the technical content.
9/25/2009	5.1	Minor	Clarified the meaning of the technical content.
11/6/2009	5.1.1	Editorial	Changed language and formatting in the technical content.
12/18/2009	6.0	Major	Updated and revised the technical content.
1/29/2010	6.1	Minor	Clarified the meaning of the technical content.
3/12/2010	6.1.1	Editorial	Changed language and formatting in the technical content.
4/23/2010	7.0	Major	Updated and revised the technical content.
6/4/2010	8.0	Major	Updated and revised the technical content.
7/16/2010	9.0	Major	Updated and revised the technical content.
8/27/2010	9.0	None	No changes to the meaning, language, or formatting of the technical content.
10/8/2010	9.0	None	No changes to the meaning, language, or formatting of the technical content.

Date	Revision History	Revision Class	Comments
11/19/2010	9.0	None	No changes to the meaning, language, or formatting of the technical content.
1/7/2011	9.0	None	No changes to the meaning, language, or formatting of the technical content.
2/11/2011	9.0	None	No changes to the meaning, language, or formatting of the technical content.
3/25/2011	10.0	Major	Updated and revised the technical content.
5/6/2011	11.0	Major	Updated and revised the technical content.
6/17/2011	11.1	Minor	Clarified the meaning of the technical content.
9/23/2011	11.1	None	No changes to the meaning, language, or formatting of the technical content.
12/16/2011	12.0	Major	Updated and revised the technical content.
3/30/2012	12.0	None	No changes to the meaning, language, or formatting of the technical content.
7/12/2012	12.0	None	No changes to the meaning, language, or formatting of the technical content.
10/25/2012	12.0	None	No changes to the meaning, language, or formatting of the technical content.
1/31/2013	13.0	Major	Updated and revised the technical content.
8/8/2013	14.0	Major	Updated and revised the technical content.
11/14/2013	15.0	Major	Updated and revised the technical content.
2/13/2014	15.0	None	No changes to the meaning, language, or formatting of the technical content.
5/15/2014	15.0	None	No changes to the meaning, language, or formatting of the technical content.
6/30/2015	16.0	Major	Significantly changed the technical content.
10/16/2015	16.0	None	No changes to the meaning, language, or formatting of the technical content.
7/14/2016	17.0	Major	Significantly changed the technical content.
6/1/2017	17.0	None	No changes to the meaning, language, or formatting of the technical content.

Table of Contents

1 Introduction

The Windows Media Digital Rights Management (WMDRM): License Protocol provides secure distribution, promotion, and sale of digital media content. The protocol is used to acquire licenses for Windows Media content protected using Digital Rights Management Version 1, Digital Rights Management Version 7, or Digital Rights Management Version 11 technologies.

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:

ASCII: The American Standard Code for Information Interchange (ASCII) is an 8-bit character-encoding scheme based on the English alphabet. ASCII codes represent text in computers, communications equipment, and other devices that work with text. ASCII refers to a single 8-bit ASCII character or an array of 8-bit ASCII characters with the high bit of each character set to zero.

base64 encoding: A binary-to-text encoding scheme whereby an arbitrary sequence of bytes is converted to a sequence of printable **ASCII** characters, as described in [RFC4648].

certificate: A certificate is a collection of attributes and extensions that can be stored persistently. The set of attributes in a certificate can vary depending on the intended usage of the certificate. A certificate securely binds a public key to the entity that holds the corresponding private key. A certificate is commonly used for authentication and secure exchange of information on open networks, such as the Internet, extranets, and intranets. Certificates are digitally signed by the issuing certification authority (CA) and can be issued for a user, a computer, or a service. The most widely accepted format for certificates is defined by the ITU-T X.509 version 3 international standards. For more information about attributes and extensions, see [RFC3280] and [X509] sections 7 and 8.

certificate revocation: The process of invalidating a **certificate**. For more information, see [RFC3280] section 3.3.

certificate revocation list (CRL): A list of **certificates** that have been revoked by the certification authority (CA) that issued them (that have not yet expired of their own accord). The list must be cryptographically signed by the CA that issues it. Typically, the certificates are identified by serial number. In addition to the serial number for the revoked certificates, the CRL contains the revocation reason for each certificate and the time the certificate was revoked. As described in [RFC3280], two types of CRLs commonly exist in the industry. Base CRLs keep a complete list of revoked certificates, while delta CRLs maintain only those certificates that have been revoked since the last issuance of a base CRL. For more information, see [X509] section 7.3, [MSFT-CRL], and [RFC3280] section 5.

curly braced GUID string: The string representation of a 128-bit globally unique identifier (**GUID**) using the form {XXXXXXXX-XXXX-XXXX-XXXX-XXXXXXXXXXXX}, where X denotes a hexadecimal digit. The string representation between the enclosing braces is the standard representation of a GUID as described in [RFC4122] section 3. Unlike a GUIDString, a curly braced GUID string includes enclosing braces.

Digital Rights Management (DRM): A set of technologies that provides control over how a given piece of protected content can be used.

elliptic curve cryptography (ECC): A public-key cryptosystem that is based on high-order elliptic curves over finite fields. For more information, see [IEEE1363].

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).

Hypertext Transfer Protocol (HTTP): An application-level protocol for distributed, collaborative, hypermedia information systems (text, graphic images, sound, video, and other multimedia files) on the World Wide Web.

Hypertext Transfer Protocol Secure (HTTPS): An extension of HTTP that securely encrypts and decrypts web page requests. In some older protocols, "Hypertext Transfer Protocol over Secure Sockets Layer" is still used (Secure Sockets Layer has been deprecated). For more information, see [SSL3] and [RFC5246].

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

RC4: A variable key-length symmetric encryption algorithm. For more information, see [SCHNEIER] section 17.1.

revocation: The process of invalidating a certificate. For more details, see [RFC3280] section 3.3.

Secure Digital Music Initiative (SDMI): An initiative to establish technology specifications that would protect the playing, storing. and distributing of digital music. These specifications are currently obsolete.

SHA-1 hash: A hashing algorithm as specified in [FIPS180-2] that was developed by the National Institute of Standards and Technology (NIST) and the National Security Agency (NSA).

transport layer: The fourth layer in the Open Systems Interconnection (OSI) architectural model as defined by the International Organization for Standardization (ISO). The transport layer provides for transfer correctness, data recovery, and flow control. The transport layer responds to service requests from the session layer and issues service requests to the network layer.

Uniform Resource Identifier (URI): A string that identifies a resource. The URI is an addressing mechanism defined in Internet Engineering Task Force (IETF) Uniform Resource Identifier (URI): Generic Syntax [RFC3986].

XML: The Extensible Markup Language, as described in [XML1.0].

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

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

[RC4-ENCRYPT] Schneier, B., "Applied Cryptography: Protocols, Algorithms, and Source Code in C", 2nd edition, Wiley, 1996, ISBN-10: 041117099 and ISBN-13: 978-0471117094.

[RFC2045] Freed, N., and Borenstein, N., "Multipurpose Internet Mail Extensions (MIME) Part One: Format of Internet Message Bodies", RFC 2045, November 1996, http://www.rfc-editor.org/rfc/rfc2045.txt

[RFC2109] Kristol, D., and Montulli, L., "HTTP State Management Mechanism", RFC 2109, February 1997, http://www.rfc-editor.org/rfc/rfc2109.txt

[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

[RFC2616] Fielding, R., Gettys, J., Mogul, J., et al., "Hypertext Transfer Protocol -- HTTP/1.1", RFC 2616, June 1999, http://www.rfc-editor.org/rfc/rfc2616.txt

[RFC2818] Rescorla, E., "HTTP Over TLS", RFC 2818, May 2000, http://www.rfc-editor.org/rfc/rfc2818.txt

[RFC2821] Klensin, J., "Simple Mail Transfer Protocol", RFC 2821, April 2001, http://www.ietf.org/rfc/rfc2821.txt

[RFC3275] Eastlake III, D., Reagle, J., and Solo, D., "(Extensible Markup Language) XML-Signature Syntax and Processing", RFC 3275, March 2002, http://www.ietf.org/rfc/rfc3275.txt

[RFC4648] Josefsson, S., "The Base16, Base32, and Base64 Data Encodings", RFC 4648, October 2006, http://www.rfc-editor.org/rfc/rfc4648.txt

[RSAFAQ] RSA Laboratories, "Frequently Asked Questions About Today's Cryptography, Version 4.1", May 2000, http://www.nordugrid.org/documents/rsalabs_faq41.pdf

[XMLSCHEMA1/2] Thompson, H., Beech, D., Maloney, M., and Mendelsohn, N., Eds., "XML Schema Part 1: Structures Second Edition", W3C Recommendation, October 2004, http://www.w3.org/TR/2004/REC-xmlschema-1-20041028/

[XMLSCHEMA2/2] Biron, P., and Malhotra, A., Eds., "XML Schema Part 2: Datatypes Second Edition", W3C Recommendation, October 2004, http://www.w3.org/TR/2004/REC-xmlschema-2-20041028/

[XML] World Wide Web Consortium, "Extensible Markup Language (XML) 1.0 (Fourth Edition)", W3C Recommendation 16 August 2006, edited in place 29 September 2006, http://www.w3.org/TR/2006/REC-xml-20060816/

1.2.2 Informative References

[CAECCRYPT] Barbosa, M., Moss, A., and Page, D., "Compiler Assisted Elliptic Curve Cryptography", http://eprint.iacr.org/2007/053.pdf

[ELLIPTICCURVE-DSA] Farkas, S., "Elliptic Curve DSA", January 2007, http://blogs.msdn.com/shawnfa/archive/2007/01/18/elliptic-curve-dsa.aspx

[ELLIPTICCURVE] RSA Laboratories, "Overview of Elliptic Curve Cryptosystems", June 1997, http://www.rsa.com/rsalabs/node.asp?id=2013

[MSDN-WMRMHEADOBJ] Microsoft Corporation, "WMRMHeader Object", http://msdn.microsoft.com/en-us/library/ms984909.aspx

[NSPCPW] Perlman, R., Speciner, M., and Kaufman, C., "Network Security: Private Communication in a Public World", New York, 1980, ASIN: B000N7EJQQ.

[SCHNEIER] Schneier, B., "Applied Cryptography, Second Edition", John Wiley and Sons, 1996, ISBN: 0471117099, http://www.wiley.com/WileyCDA/WileyTitle/productCd-0471117099.html

[X9.62] American National Standards Institute, "Public Key Cryptography for the Financial Services Industry, The Elliptic Curve Digital Signature Algorithm (ECDSA)", ANSI X9.62:2005, 2005, http://webstore.ansi.org/ansidocstore/product.asp?sku=ANSI+X9%2E62%3A2005

Note There is a charge to download the specification.

1.3 Overview

Digital Rights Management (DRM) version 1, version 7, and version 11 provide a means of acquiring a license for Windows Media content.

When using Digital Rights Management Version 1, the client generates a license request and sends it to a license server as an HTTP GET request. The server receives the GET request and returns the license to the client embedded within an HTML page.

Digital Rights Management Version 7 uses a packet containing a license request in extensible markup language (XML) format and is sent using an HTTP POST request. The server responds with an XML packet containing any number and combination of version 1 and version 7 licenses.

Digital Rights Management Version 11 is functionally equivalent to the version 7 protocol, with the addition of a few XML fields in the license request challenge body.

In all versions of the license protocol, the intent is to document the protocol for acquisition of licenses in which the license details themselves are not technically relevant to the protocol. License formats are described in detail for completeness, however.

The following table describes cryptographic and mathematical operators. For more information, see [NSPCPW].

Operator	Description
cryptographic operator "K{text}"	Text encrypted with symmetric key K.
cryptographic operator "[text] $_K$"	Text signed with private portion of asymmetric key K, K_{priv}.
cryptographic operator "$\{text\}_K$"	Text encrypted with public portion of asymmetric key K, K_{pub}.
mathematical operator "□"	A bitwise exclusive OR.
mathematical operator "~"	A bitwise negation.
mathematical operator "|"	A concatenation.

1.3.1 Digital Rights Management Version 1

Digital Rights Management Version 1 provides the means of acquiring a license for Windows Media content. Its packets include a client request for a license and a server response that contains the license.

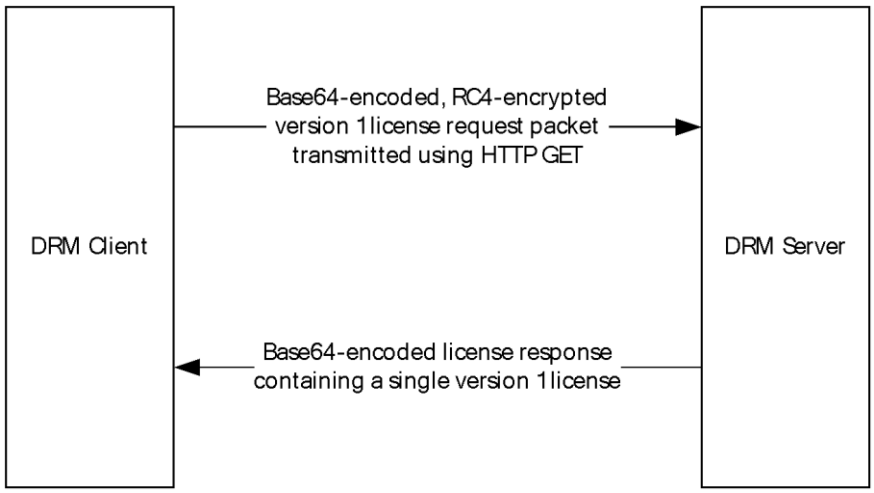


Figure 1: DRM version 1 license request and response

The **Digital Rights Management** client application generates a license request and sends it to a license server. The request is a binary string that is partially encrypted using the **Rivest Cipher 4 (RC4)** (as specified in [RC4-ENCRYPT]) and then encoded using the Base64 Encoding algorithm, as specified in section 2.2.1.1.

The response is a single version 1 license, formatted as a binary string, and encoded with the base64 encoding algorithm, as specified in section 2.2.1.1. It is returned to the client embedded within an HTML page.

A Digital Rights Management Version 1 license is represented as specified in section 2.2.2.3.

The structures that are used by version 1, version 7 and version 11 of the WMDRM: License Protocol are specified in section 2.2.1.

This protocol uses the following packets.

Packet	Description
DRM Version 1 License Request	Contains the client's request for a license.
DRM Version 1 License Response	Contains the server's response to the client's request for a license.

RC4 is a proprietary encryption algorithm available under license from RSA Security, as specified in [RSAFAQ].

1.3.2 Digital Rights Management Version 7

Digital Rights Management Version 7 provides the means of acquiring a license for Windows Media content. Its packets include a client request for a license and a server response that contains the license.

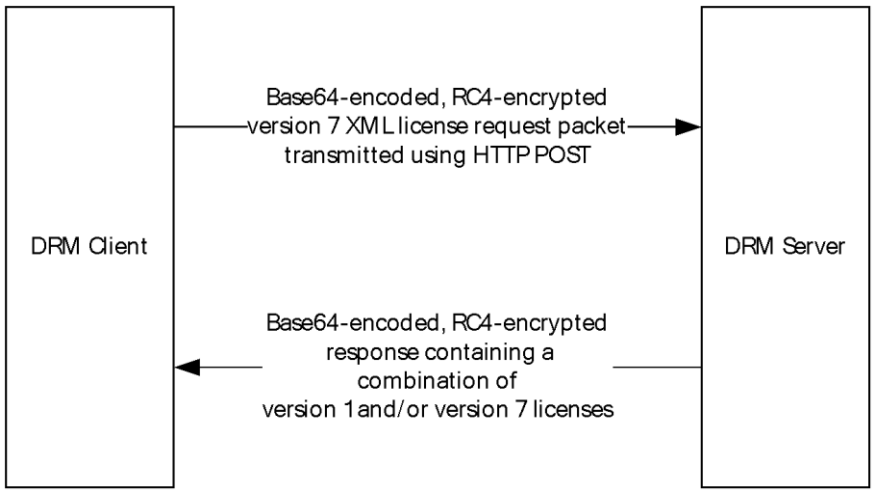


Figure 2: DRM version 7 license request and response

The **Digital Rights Management** client generates a license request and sends it to a license server. The request is in extensible markup language (XML) format, partially **RC4**-encrypted, and then encoded using the base64 algorithm, as specified in section 2.2.1.1. It is sent to the server by means of an HTTP POST request. For more information about RC4, see Remarks at the end of this topic.

The response is an RC4-encrypted XML packet. The first 80 bytes of the license response packet are an **ECC**-encrypted RC4 key. The RC4 key is generated by the server using the EncRandNum member of the **CLIENTID structure**, section **2.2.3.1.3.4**, sent by the client within the license request. The remainder of the packet (the license data itself) is encrypted with the generated RC4 key. The packet is then encoded with the base64 algorithm, as specified in section 2.2.1.1. It can contain any number and combination of version 1 and version 7 licenses. Each version 7 license is itself RC4-encrypted using the mechanism described in this topic.

A WMDRM: License Protocol version 7 license is represented, as specified in section 2.2.3.2.4.2.

The structures that are used by both version 1 and version 7 of the WMDRM: License Protocol are specified in section 2.2.1.

This protocol uses the following packets.

Packet	Description
DRM Version 7 License Request	Contains the client's request for a license.
DRM Version 7 License Response	Contains the server's response to the client's request for a license.
DRM Version 7 License Format	Contains an XML-formatted license.

RC4 is a proprietary encryption algorithm available under license from RSA Security, as specified in [RSAFAQ].

1.3.3 Digital Rights Management Version 11

Digital Rights Management Version 11 is almost identical to the version 7 protocol, with the addition of a few fields in the license request packet.

This protocol uses the following packets.

Packet	Description
DRM Version 11 License Request	Contains the client's request for a license.
DRM Version 11 License Response	Contains the server's response to the client's request for a license.
DRM Version 11 License Format	Contains an XML-formatted license.

1.4 Relationship to Other Protocols

Protocol versions 1, 7, and 11 can be implemented over **Hypertext Transfer Protocol (HTTP)**, **Hypertext Transfer Protocol over Secure Sockets Layer (HTTPS)**, or any other appropriate transport protocol. Selection of a specific transport protocol is at the discretion of the content encoder (the license acquisition URL is embedded within the content).

1.5 Prerequisites/Preconditions

The following data is licensed from Microsoft for the license acquisition server prior to implementing any of these protocols:

- Private server cryptographic key (KS_{priv}).
- Server certificate chain (CS).

The following data is unique for every license server and is generated by the implementer of the server side of the protocol:

- Server public/private key pair (KL).

The following data is licensed from Microsoft for the client application prior to implementing the client portion of this protocol:

- Client application certificate (CA) (leaf certificate only).
- Client machine certificate (CM).

The following keys and certificates are used by the client application and referenced in this document:

- Private client cryptographic key (KC_{priv}).
- Public server cryptographic key (KS_{pub}).
- Public key representing the root certificate authority key used to sign the root certificate in CS (KI_{pub}). KI_{pub} is given by the following byte sequence:

  0x4D, 0xBF, 0xD9, 0x0D, 0xD9, 0x6E, 0x8C, 0x9E,

  0x32, 0x5F, 0x4F, 0x3D, 0xEC, 0xA9, 0x84, 0x59,

  0x6B, 0x5E, 0x06, 0x86, 0xE7, 0xE2, 0xC2, 0x8B,

  0xDE, 0x14, 0x4B, 0x29, 0x2C, 0xEC, 0x4D, 0x1D,

  0x76, 0xFD, 0x5A, 0x14, 0x90, 0x3A, 0x10, 0x77

1.6 Applicability Statement

None.

1.7 Versioning and Capability Negotiation

In the WMDRM: License Protocol, there is no facility for version or capability negotiation. The client submits requests to a server that understands the maximum protocol version used by the client. In practice, content providers embed license acquisition specifics within the content file headers. This information indicates to the client which license version and license acquisition protocol will be used.<1>

This protocol can be implemented on top of the following:

- TCP
- HTTP
- HTTPS

1.8 Vendor-Extensible Fields

Within the version 7 and version 11 license response packet, vendors are free to add any well-formed XML data to the <META> element. The contents of this element are not used by the **Digital Rights Management** client application.

This protocol uses Win32 error codes. These values are taken from the Windows error number space defined in [MS-ERREF] section 2.2. Vendors SHOULD reuse those values with their indicated meaning. Choosing any other value runs the risk of a collision in the future.

1.9 Standards Assignments

None.

2 Messages

This protocol references commonly used data types as defined in [MS-DTYP] such as GUID--Curly Braced String Representation (section 2.3.4.3).

2.1 Transport

The WMDRM: License Protocol uses **HTTP** (as specified in [RFC2616]) or HTTP over TLS (as specified in [RFC2818]) as the **transport layer**.<2> The use of HTTP over TLS is triggered by the specification of an "https" URL rather than an "http" URL within the WMRMHEADER (section 2.2.3.1.3.12). Messages and data are sent via URI query strings, HTTP POST headers, and HTTP responses.

Some client applications can also use the HTTP cookie mechanism (as specified in [RFC2109]) as a transport and state management mechanism outside the purview of license acquisition. The HTTP cookie mechanism allows named data items to be sent from one party to another as part of an HTTP message, stored by the receiving party, and returned automatically to the original party as part of all subsequent HTTP messages to that party.

2.2 Message Syntax

2.2.1 Common Data Types and Algorithms

The following structures and algorithms are common to version 1, version 7, and version 11 of the WMDRM: License Protocol.

Unless otherwise noted, all multi-octet integral values are stored in **little-endian** format.

Unless otherwise noted, all data structures are packed to 4-octet alignment.

For more information about encryption algorithms within this document, see [CAECCRYPT], [ELLIPTICCURVE], [ELLIPTICCURVE-DSA], [SCHNEIER] section 19.6, and [X9.62].

This protocol uses the following types specified in [MS-DTYP].

Type	Reference
BYTE	[MS-DTYP] section 2.2.6

2.2.1.1 Base64 Encoding

The standard base64 encoding algorithm (as specified in [RFC4648]) is used to transmit binary data. Base64 processes data as 24-bit groups, mapping it to four encoded characters of 6 bits each. It is sometimes referred to as 3-to-4 encoding. Each 6-bit group in the 24-bit group is used as an index into a mapping table (see section 2.2.1.1.1) to obtain a character for the encoded data. By convention, line lengths in the encoded data are limited to 76 characters, but this is not strictly enforced in this protocol.

Note The characters used in base64 encoding do not include any of the special characters of the Simple Mail Transfer Protocol (SMTP) (as specified in [RFC2821]), or the hyphen used with Multipurpose Internet Mail Exchange (MIME) boundary strings, as specified in [RFC2045].

2.2.1.1.1 Base64 Mapping Table

This is the base64 mapping table.

```
 0 A     17 R     34 i     51 z
 1 B     18 S     35 j     52 0
 2 C     19 T     36 k     53 1
 3 D     20 U     37 l     54 2
 4 E     21 V     38 m     55 3
 5 F     22 W     39 n     56 4
 6 G     23 X     40 o     57 5
 7 H     24 Y     41 p     58 6
 8 I     25 Z     42 q     59 7
 9 J     26 a     43 r     60 8
10 K     27 b     44 s     61 9
11 L     28 c     45 t     62 +
12 M     29 d     46 u     63 /
13 N     30 e     47 v
14 O     31 f     48 w
15 P     32 g     49 x
16 Q     33 h     50 y
```

2.2.1.1.2 Example: Base64 Encoding of 3 Bytes

This is an example of base64 encoding of 3 Bytes: "XYZ".

```
Input data          X        Y        Z
Input bits   01011000-01011001-01011010
Bit groups   010110-000101-100101-011010
Mapping           W      F      l      a
```

2.2.1.1.3 Base64 and DRM

In the WMDRM: License Protocol, base64 encoding refers to a slightly modified version of the standard base64 algorithm. **Digital Rights Management** base64 encoding is identical to standard base64 encoding, with the exception of the last two characters in the following mapping table.

```
 0 A     17 R     34 i     51 z
 1 B     18 S     35 j     52 0
 2 C     19 T     36 k     53 1
 3 D     20 U     37 l     54 2
 4 E     21 V     38 m     55 3
 5 F     22 W     39 n     56 4
 6 G     23 X     40 o     57 5
 7 H     24 Y     41 p     58 6
 8 I     25 Z     42 q     59 7
 9 J     26 a     43 r     60 8
10 K     27 b     44 s     61 9
11 L     28 c     45 t     62 !
12 M     29 d     46 u     63 *
13 N     30 e     47 v
14 O     31 f     48 w
15 P     32 g     49 x
16 Q     33 h     50 y
```

2.2.1.2 Cryptographic Parameters

The following 160-bit **elliptic curve cryptography (ECC)** curve is used in this document.

ECC_1

Parameter	Value
p(modulus)	0x89abcdef012345672718281831415926141424f7
a	0x37a5abccd277bce87632ff3d4780c009ebe41497
b	0x0dd8dabf725e2f3228e85f1ad78fdedf9328239e
generator x	0x8723947fd6a3a1e53510c07dba38daf0109fa120
generator y	0x445744911075522d8c3c5856d4ed7acda379936f
curve order	0x89abcdef012345672716b26eec14904428c2a675

Prior to encryption, the plaintext (length 1 – 16 bytes) is prepared with the following sequence of operations:

1. Copy the plaintext into a buffer, "x", comprising five DWORDs.
2. The fifth DWORD of x is set to zero.
3. If there is a solution for y in the following equation, x|y is now ready for encryption.

```
(y^2) mod p = (x^3 + ax + b) mod p
```

4. If there is no solution to this equation, increment the fifth DWORD of x and repeat the preceding step.

2.2.1.3 Cryptographic Keys

The client and server use a set of cryptographic keys as follows:

KC: An **ECC**$_1$ key that represents the client application. The client knows KC_{priv} and the server knows KC_{pub}.

KS: A well-known ECC_1 key used to protect the privacy of packets sent between client and server. The client knows KS_{pub} and the server knows KS_{priv}.

KL: An ECC_1 key that represents the license server. The server knows KL_{pub} and KL_{priv}.

KM: An ECC_1 key that represents a specific instance of a machine running the client application. The key pair is either created by or issued to the DRM system during a one-time initialization process. The details are implementation-specific.

KM$_{pub}$ is transmitted from the client to server during a license request.

2.2.1.4 PK

The **PK** structure contains the **PUBKEY** structure and its version information.

```
typedef struct {
  PUBKEY pubkey;
  BYTE version[4];
} PK;
```

pubkey: A **PUBKEY** structure that contains a public key.

version: A 4-byte buffer that contains version information for the public key. MUST be {0x00, 0x01, 0x00, 0x00}.

2.2.1.5 PKCERT

The **PKCERT** structure contains a signed **PK** structure.

```
typedef struct {
  PK pk;
  BYTE sign[40];
} PKCERT;
```

pk: A **PK** structure that contains a public key and its version information.

sign: A 40-byte buffer that contains the signature of the pk member. This signature is created using ECDSA over curve **ECC**$_1$. For more information about ECDSA, see [ELLIPTICCURVE-DSA].

$[pk]_K$

where K is an ECC_1 key.

2.2.1.6 PUBKEY

The **PUBKEY** structure contains a public key.

```
typedef struct {
  BYTE y[40];
} PUBKEY;
```

y: A 40-byte buffer that contains a public key. This is the public portion of a public/private key pair in **ECC**$_1$. The x-coordinate is stored in bytes 0 - 19; the y-coordinate in bytes 20 - 39. The two coordinates are base 0x100000000 integers stored in little-endian order.

2.2.1.7 LicenseToSend

The **LicenseToSend** structure is a container for an arbitrary number of variable-length licenses. It is passed to the Digital Rights Management License Protocol by the higher layer. This structure is used in the **TransmitLicensesToClient** abstract interface (section 3.2.1.1).

```
typedef struct _LicenseToSend {
  int LicenseVersion;
  int LicenseLength;
  byte License[];
} LicenseToSend;
```

LicenseVersion: The version of the license included in this structure.

Value	Meaning
1	License is constructed for DRM version 1.
7	License is constructed for DRM version 7.

LicenseLength: The length, in bytes, of the license included in this structure.

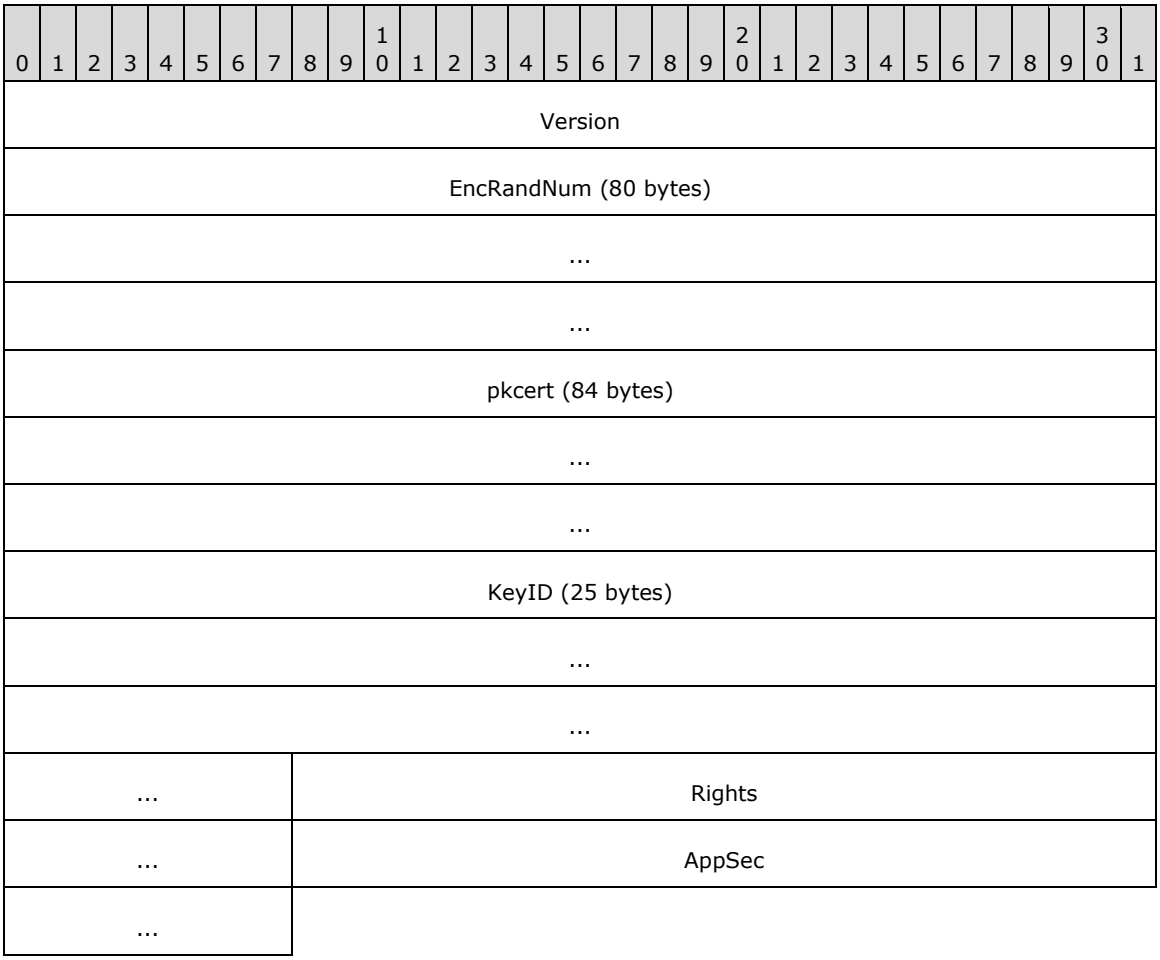
License: A byte array containing the license constructed by the higher layer. This field is **LicenseLength** bytes in length. The format of the license is implementation-dependent.

2.2.2 DRM Version 1 Data Types

The following structures and algorithm are specific to version 1 of the WMDRM: License Protocol.

2.2.2.1 DRM Version 1 License Request

The DRM Version 1 License Request packet is used by the client to request a license for content. This packet is transmitted to the server via a **URI** parameter "challenge" as a **Digital Rights Management (DRM) base64**-encoded value. The URI parameter *DRMVer* is also sent to the server with this license request and MUST appear after the "challenge" URI parameter. For a version 1 client, the value of *DRMVer* MUST be 1.3. For a client that supports version 7 and higher, this value MUST be 1.4. This value is ignored by the server.



Version (4 bytes): The request version. MUST be {0x00, 0x01, 0x00, 0x01}.

EncRandNum (80 bytes): A one-time used, previously 20-byte random number that is encrypted using **ECC**$_1$ with the public server cryptographic key (KS). Before encryption, this buffer contains the following byte values:

- bytes 0 – 6: Used as the initialization vector (IV) to create an **RC4** key (KR)

- bytes 7 – 19: Not used

pkcert (84 bytes): An RC4-encrypted PKCERT that contains a signed copy of KM_{pub}.

KeyID (25 bytes): An RC4-encrypted content key identifier. The content key ID is generated by the server and stored in the header of a protected content stream. Only the first 25 bytes of this field are used. The **KeyID** can come from any source available to the client, but is typically extracted from a content header.

Rights (4 bytes): An RC4-encrypted request for playback rights, which can be any combination of the values in the following table. The values used in the challenge are typically provided by the DRM-enabled application, but could be any combination implemented by the client.

Byte Array	Meaning
RIGHT_PLAY_ON_PC 0x01000000	The right to play back content. This is also known as RIGHT_PLAY_ON_PC.
RIGHT_COPY_TO_NONSDMI_DEVICE 0x02000000	The right to copy licensed content to a device that is not compliant with the **Secure Digital Music Initiative (SDMI)**. This is also known as RIGHT_COPY_TO_NONSDMI_DEVICE.
RIGHT_BURN_TO_CD 0x08000000	The right to copy licensed content to a CD. This is also known as RIGHT_BURN_TO_CD.
RIGHT_COPY_TO_SDMI_DEVICE 0x10000000	The right to copy licensed content to an SDMI device. This is also known as RIGHT_COPY_TO_SDMI_DEVICE.

AppSec (4 bytes): An RC4-encrypted security level of the application that makes the request. The security level MUST be equal to the security level in the client application certificate (CA).

Cryptographic sequence:

1. pkcert.pk = KM_{pub}
2. pkcert.sign = $[pkcert.pk]_{KC}$
3. $\{EncRandNum\}_{KS}$
4. KR {pkcert}
5. KR {KeyID}
6. KR {Rights}
7. KR {AppSec}

2.2.2.2 DRM Version 1 License Response

The license response is returned to the client as an HTML page containing a **base64**-encoded **CERTIFIED_LICENSE** structure. The response is formatted as follows. Both the text enclosed in braces ("{{" and "}}") and the braces MUST be replaced or removed as appropriate.<3>

```
<HTML><HEAD><TITLE>{{optional page title}}</TITLE>
<Script Language="VBScript">Sub Window_OnLoad()
DrmStore.StoreLicense("{{base64-encoded CERTIFIED_LICENSE}}")
End Sub</Script></HEAD>
<BODY>{{optional descriptive text}}
<OBJECT classid=CLSID:760C4B83-E211-11D2-BF3E-00805FBE84A6 id=DrmStore>
<EMBED MAYSCRIPT TYPE="application/x-drm" HIDDEN="true"
LICENSE="{{base64-encoded CERTIFIED_LICENSE}}"></OBJECT>
```

```
{{optional descriptive text}}</BODY></HTML>
```

2.2.2.3 DRM Version 1 License Format

A **Digital Rights Management (DRM)** version 1 license response is a base64-encoded **CERTIFIED_LICENSE** structure.

The **CERTIFIED_LICENSE** structure consists of two **certificates** and a license. The first certificate represents the Microsoft signing certificate. The second certificate represents the signing certificate of the server issuing the content license.

The Digital Rights Management version 1 license format contains the following top-level structures.

Structure	Description
CERT	Defines the certificate component of a DRM version 1 certified license.
CERTDATA	Defines the data block of a certificate, including the public key, serial number, and certificate issuer.
CERTIFIED_LICENSE	Defines a version 1 certified license before it is encoded with base64 encoding.
LICENSE	Defines the license portion of a version 1 certified license.
LICENSEDATA	Defines the data portion of a version 1 license, including the rights and security settings.

2.2.2.3.1 CERT

The **CERT** structure defines the **certificate** component of a **Digital Rights Management (DRM)** version 1 certified license. A CERT structure is obtained by a DRM version 1 license server through an enrollment process that issues valid license server CERT structures. This certificate is not processed on the server, but only included in the license response.

```
typedef struct {
  BYTE certVersion[4];
  BYTE dataLen[4];
  BYTE sign[40];
  CERTDATA cd;
} CERT;
```

certVersion: A 4-byte buffer that contains the certificate version. Valid values for certificate version are {0x00, 0x01, 0x00, 0x00}.

dataLen: A 4-byte buffer that contains the size of the **cd** field, in bytes, as a sequence of four hexadecimal values (this is a DWORD stored in **little-endian** order). For example, if **cd** is 300 bytes (0x12c bytes), this field contains {0x2C, 0x01, 0x00, 0x00}.

sign: A 40-byte buffer that contains the signature of the **cd** member. This signature is created using ECDSA over curve **ECC_1**. The key used to sign this data is the private key of the certificate authority that issued this certificate.

$[cd]_K$

cd: A **CERTDATA** structure that contains the certificate data, including its public key, issuer, and expiration date.

2.2.2.3.2 CERTDATA

The **CERTDATA** structure defines the data block of a certificate, including the public key, serial number, and **certificate** issuer.

```
typedef struct {
  BYTE pk[40];
  BYTE expiryDate[4];
  DWORD serialNumber;
  DWORD issuer;
  DWORD subject;
} CERTDATA;
```

pk: A 40-byte buffer that contains a public key. This is the public portion of a public/private key pair in **ECC**$_1$. The x-coordinate is stored in bytes 0 – 19; the y-coordinate in bytes 20 – 39.

expiryDate: A 4-byte buffer that contains the date on which the certificate expires. All values are encoded as hexadecimal. The first byte contains the value of the first two digits of the year, the second contains the value of the latter two digits of the year, the third contains the value of the month, and the fourth contains the value of the day. For example, the date 12/30/2002 is represented as {0x14, 0x02, 0x0C, 0x1E}.

serialNumber: A serial number that identifies the certificate.

issuer: A certificate server identifier that is provided by Microsoft.

subject: A number that identifies the subject of the certificate. The subject is provided by Microsoft.

2.2.2.3.3 CERTIFIED_LICENSE

The **CERTIFIED_LICENSE** structure defines a version 1 certified license.

```
typedef struct {
  LICENSE license;
  CERT cert1;
  CERT cert2;
} CERTIFIED_LICENSE;
```

license: A **LICENSE** structure that contains the license component of a version 1 certified license.

cert1: A **CERT** structure that contains the Microsoft-signed **certificate** representing the license server. This certificate is supplied in CS.

cert2: A **CERT** structure that contains the root certificate representing the Microsoft certificate authority. This certificate is supplied in CS.

2.2.2.3.4 LICENSE

The **LICENSE** structure defines the license portion of a version 1 certified license.

```
typedef struct {
  BYTE licVersion[4];
  BYTE dataLen[4];
  BYTE sign[40];
  LICENSEDATA ld;
```

```
} LICENSE;
```

licVersion: A 4-byte buffer that contains the license version. This value MUST contain {0x00, 0x01, 0x00, 0x00}.

dataLen: A 4-byte buffer that contains the size of the **ld** field, in bytes, as a sequence of four hexadecimal values (this is a DWORD that is stored in **little-endian** order). For example, if **ld** is 300 bytes (0x12c bytes), this field contains {0x2C, 0x01, 0x00, 0x00}.

sign: A 40-byte buffer that contains the signature of the **ld** member. This signature is created by using ECDSA over curve **ECC_1**. The key that is used to sign this data is the private key of **cert1** in the enclosing **CERTIFIED_LICENSE** structure (KL).

ld: A **LICENSEDATA** structure that contains the license data, including the digital rights and security data.

Cryptographic Sequence:

sign = $[ld]_{KL}$

2.2.2.3.5 LICENSEDATA

The **LICENSEDATA** structure defines the data portion of a version 1 license, including the rights and security settings.

```
typedef struct {
  char KID[25];
  BYTE key[80];
  BYTE rights[4];
  DWORD appSec;
  BYTE expiryDate[4];
} LICENSEDATA;
```

KID: A 25-character array that contains the content key ID. The **KID** MUST be a value that uniquely identifies content for which the license is issued. Use of a **base64**-encoded **GUID** is recommended. This value is usually a copy of the KID value sent in the license challenge, but that is not technically required.

key: An 80-byte buffer that contains the encrypted **RC4** content key ($K_{content}$) and a copy of its bitwise negation ($P_{content} = \sim K_{content}$). This field is encrypted using **ECC_1** with KM. Prior to encryption and after decryption, bytes 0 through 6 of the plaintext represent $K_{content}$ and bytes 7 through 13 of the plaintext represent $P_{content}$. These values can be compared to ensure that they were stored and transmitted properly by calculating

$\sim(K_{content} \oplus P_{content})$

If this value is not 0, $K_{content}$ and/or $P_{content}$ are suspect and cannot be used.

The key is secret and known only to the service. How keys are stored and referenced is service implementation-dependent and not relevant to the protocol.

rights: A 4-byte buffer that contains the client rights for the licensed content. These values are logically combined in byte order. The value used for **rights** is entirely dependent on the service-business-logic implementation.

Byte Array	Meaning
{0x01,0x00,0x00,0x00}	The client is authorized to play back the content. This is known as

Byte Array	Meaning
0x01000000	RIGHT_PLAY_ON_PC.
{0x02,0x00,0x00,0x00} 0x02000000	The client is authorized to copy the licensed content to a device that is not compliant with the **Secure Digital Music Initiative (SDMI)**. This is known as RIGHT_COPY_TO_NONSDMI_DEVICE.
{0x04,0x00,0x00,0x00} 0x04000000	The client is not authorized to restore the license content. This is known as RIGHT_NO_RESTORE.
{0x08,0x00,0x00,0x00} 0x08000000	The client is authorized to burn the licensed content to a CD. This is known as RIGHT_BURN_TO_CD.
{0x10,0x00,0x00,0x00} 0x10000000	The client is authorized to copy the licensed content to an SDMI device. This is known as RIGHT_COPY_TO_SDMI_DEVICE.
{0x20,0x00,0x00,0x00} 0x20000000	The client can perform any of the authorized actions one time. This is known as RIGHT_ONE_TIME.
{0x00,0x00,0x01,0x00} 0x00000100	The client is authorized to handle SDMI-generated events. This is known as RIGHT_SDMI_TRIGGER.
{0x00,0x00,0x02,0x00} 0x00000200	The client is not authorized to make any further SDMI copies of the licensed content. This is known as RIGHT_SDMI_NOMORECOPIES.

appSec: The minimum application security level required to play content associated with this license. The application security level is embedded within CA. Valid values range from 0 to 2000. The value used for **appSec** is entirely dependent on the service-business-logic implementation.

expiryDate: A 4-byte buffer that contains the date on which the license expires. All values are encoded as hexadecimal. The first byte contains the value of the first two digits of the year, the second contains the value of the last two digits of the year, the third contains the value of the month, and the fourth contains the value of the day. For example, the date 12/30/2002 is represented as {0x14, 0x02, 0x0C, 0x1E}. A value of { 0xFF, 0xFF, 0xFF, 0xFF } indicates that there is no expiration date for the license. The value used for **expiryDate** is entirely dependent on the service-business-logic implementation.

Cryptographic Sequence:

key = { $K_{content}$ | $P_{content}$ }$_{KM}$

When content is encrypted, the packager generates a content key identifier (KID) and a content key as a pair. The key is used to encrypt the content, and the KID is placed in the content header of a license request.

The **Digital Rights Management (DRM)** component on the client computer can use this key to decrypt the content.

2.2.3 DRM Version 7 Data Types

The following structures and algorithm are specific to version 7 of the WMDRM: License Protocol.

2.2.3.1 DRM Version 7 License Request

The DRM Version 7 License Request packet is used by the client to request a license for Windows Media content.

2.2.3.1.1 Silent and Nonsilent Requests

The **DRM** version 7 client can generate either a silent or a nonsilent license request. By contrast, version 1 clients always generate a nonsilent license request.

Silent license acquisition means that the client application SHOULD NOT display a license acquisition user interface, which requires active user input, to the end user. A client application MAY display some form of progress indicator.

Conversely, nonsilent license acquisition means that the client application MAY display a license acquisition user interface, which requires active user input, to the end user.

2.2.3.1.1.1 Silent Requests

Silent acquisition is transparent to the user. The client prefixes the string "nonsilent=0&challenge=", or alternatively, just "challenge=", to the head of the encoded data before obtaining the final POST data. **Digital Rights Management** sends the request directly and receives the response directly. The license is delivered and stored without any other action being required.

2.2.3.1.1.2 Nonsilent Requests

Nonsilent acquisition means that the request came from within a user-visible web browser, as opposed to some other means that would be invisible to the user. This decision is made by the higher layer based on implementation-specific application logic.

This decision can be made due to a variety of reasons not relevant to the protocol itself. For example, a webpage is displayed in a browser application that requires the user to enter information. In this case, the POST data is handed back to the higher layer, which prefixes the string "nonsilent=1&challenge=" to the head of the encoded packet and then makes the HTTP POST request. The server displays custom HTML responses because the request data indicates that the request is coming from a user-visible browser.

2.2.3.1.2 HTTP POST Headers

nonsilent: Optional specification for silent versus nonsilent license acquisition. If not present, silent license is assumed. Allowable values are "0" to indicate silent license acquisition and "1" to indicate nonsilent license acquisition.

challenge: Required value containing the version 7 license request body.

2.2.3.1.3 XML Schema for Version 7 License Request

The following is an XML schema for the version 7 license request packet. Where required, elements, attributes, and values are described in greater detail after the schema.

```
<xs:schema xmlns:xs="http://www.w3.org/2001/XMLSchema">
  <xs:element name="LICENSEREQUEST">
    <xs:complexType>
      <xs:sequence>
        <xs:element name="V1CHALLENGE">
          <xs:simpleType>
            <xs:restriction base="xs:base64Binary" />
          </xs:simpleType>
        </xs:element>
        <xs:element name="ACTIONLIST" minOccurs="0">
          <xs:complexType>
            <xs:sequence>
              <xs:element name="ACTION" type="ActionNameType"
                  minOccurs="1" maxOccurs="5" />
            </xs:sequence>
          </xs:complexType>
        </xs:element>
```

```
        <xs:element name="CLIENTINFO" minOccurs="0">
          <xs:complexType>
            <xs:sequence>
              <xs:element name="CLIENTID" type="xs:base64Binary" />
              <xs:element name="CLIENTVERSION" type="xs:string" />
              <xs:element name="SECURITYVERSION" type="xs:string" />
              <xs:element name="APPSECURITY" type="xs:string" />
              <xs:element name="SUBJECTID1" type="xs:integer" />
              <xs:element name="SUBJECTID2" type="xs:integer" />
              <!-- SUBJECTID2 tag must be present; content is optional. -->
              <xs:element name="DRMKVERSION" type="xs:string"
minOccurs="0" />
            </xs:sequence>
          </xs:complexType>
        </xs:element>
        <xs:element name="REVOCATIONINFO" minOccurs="0">
          <xs:complexType>
            <xs:simpleContent>
              <xs:extension base="xs:base64Binary"/>
            </xs:simpleContent>
          </xs:complexType>
        </xs:element>
        <xs:element name="WRMHEADER" minOccurs="0" >
          <xs:complexType>
            <xs:sequence>
              <!-- content varies, depending on media file header
information. -->
              <xs:any />
            </xs:sequence>
          </xs:complexType>
        </xs:element>
      </xs:sequence>
      <xs:attribute name="version" use="required" fixed="2.0.0.0" />
    </xs:complexType>
  </xs:element>
  <xs:simpleType name="ActionNameType">
    <xs:restriction base="xs:string">
      <xs:enumeration value="Play" />
      <xs:enumeration value="Print.redbook" />
      <xs:enumeration value="CREATE PM LICENSE" />
      <xs:enumeration value="Backup" />
      <xs:enumeration value="Restore" />
    </xs:restriction>
  </xs:simpleType>
</xs:schema>
```

2.2.3.1.3.1 ACTION

The ACTION element contains the action rights that the client is requesting based on parameters the client application has provided to the DRM implementation. The meaning of the element contents is described in the following table. The <ACTION> element is filled with appropriate data based on the action the client application will perform with the content for which the license is being requested.

Predefined string	Meaning
Play	Play content on the client computer.
Print.redbook	Burn content to a CD.
CREATE_PM_LICENSE	Transfer content to a portable device.
Backup	Permit backup of the license.
Restore	Allow the license to be restored from another location.

2.2.3.1.3.2 APPSECURITY

The APPSECURITY element contains the security level of the application making the license request. This value is not limited to a specific range and can be used by the service provider to limit license distribution.

2.2.3.1.3.3 CLIENTID (Element)

The CLIENTID element contains base64-encoded **CLIENTID** structure.

2.2.3.1.3.4 CLIENTID (Structure)

The **CLIENTID** structure contains the **Digital Rights Management** version and security **certificate** of the client computer. A conforming DRM client implementation has a valid **CLIENTID** obtained through one of multiple means not relevant to license acquisition. The existence of a **CLIENTID** within the DRM client is a requirement of a valid client. No processing is done with this element, as it is only included in the payload of the protocol.

```
typedef struct {
  BYTE Version[4];
  BYTE EncRandNum[80];
  PKCERT pkcert;
} CLIENTID;
```

Version: The Digital Rights Management version. MUST be {0x02, 0x00, 0x00, 0x00}.

EncRandNum: One-time use random number encrypted using **ECC**$_1$ with KS. The first 7 bytes (unencrypted) of EncRandNum are used as the initialization vector (IV) to create an **RC4** key (KR).

pkcert: A **PKCERT** structure that contains the machine certificate.

Cryptographic Sequence:

1. pkcert.pk = KM_{pub}
2. pkcert.sign = $[pkcert.pk]_{KC}$
3. $\{EncRandNum\}_{KS}$
4. KR {pkcert}

2.2.3.1.3.5 CLIENTVERSION

The CLIENTVERSION element contains the version of the **Digital Rights Management** client making the request. Generally, this will be of the form "2.a.0.b", where "a" is the minor version and "b" is the client build number.

2.2.3.1.3.6 DRMKVERSION

The DRMKVERSION element contains the version of the kernel mode **Digital Rights Management** file (Drmk.sys) on the client computer. Generally, this takes the form "a.b.c.d" where a, b, c, and d are whole numbers.

2.2.3.1.3.7 REVOCATIONINFO

The <REVOCATIONINFO> element contains the **base64**-encoded REV_INFO known to the client application. The REV_INFO is stored by the client in a local data store and retrieved from the store and parsed to extract the value used for <REVOCATIONINFO>.

2.2.3.1.3.8 SECURITYVERSION

The SECURITYVERSION element contains the security version of the **Digital Rights Management** root of trust on the client computer.

Each license server has a list of client verification keys that enable it to ensure the validity of license requests. Each verification key string is a base64-encoded **PUBKEY** structure. The list of possible security versions and verification keys is a separate licensable piece of data.

2.2.3.1.3.9 SUBJECTID1

The SUBJECTID1 element contains the certificate subject identifier of the component that is communicating directly with the **Digital Rights Management** client component. The subject identifier is taken from the certificates that are used to establish secure channels between components. It uniquely identifies a component. For example, the version of an application or an SDK can be a subject identifier.

2.2.3.1.3.10 SUBJECTID2

The SUBJECTID2 element contains the **certificate** subject identifier of the component that is communicating with another component, which is in turn communicating with the **Digital Rights Management** client component. This element is used only if SUBJECTID1 is the subject identifier of an SDK. The subject identifier is taken from the certificates that are used to establish secure channels between components. It uniquely identifies a component. For example, the version of an application or an SDK can be a subject identifier.

The SUBJECTID2 element MUST be present, even if empty.

2.2.3.1.3.11 V1CHALLENGE

The <V1CHALLENGE> element is a base64-encoded **Digital Rights Management** version 1 license request with a **KeyID** field consisting of all zeros.

2.2.3.1.3.12 WMRMHEADER

The WMRMHEADER element contains data that is taken verbatim from the header of the content. The content is dictated by the Windows Media Rights Manager (WMRM).

For more information about the WMRMHEADER and how it is generated, see [MSDN-WMRMHEADOBJ].

2.2.3.2 DRM Version 7 License Response

The DRM Version 7 License Response packet is used by the license server to send a license for Windows Media content to a client. The format of the response, which is in XML, can include any number and combination of WMDRM: License Protocol version 1 and version 7 licenses, encoded with the base64 encoding algorithm.

2.2.3.2.1 Silent Acquisition

The license response is returned directly to the client as the body of the HTTP response.

2.2.3.2.2 Nonsilent Acquisition

The license response is returned to the client as an HTML page that uses a COM object to store a base64-encoded LICENSERESPONSE XML blob in the local license store of the client. The LICENSERESPONSE XML is embedded in a script section of the Web page.

2.2.3.2.3 Errors

If the license request is refused after a silent request, the server returns any HTTP error status code that is returned to the higher layer. The higher layer then attempts nonsilent acquisition.

If the license request is refused after a nonsilent request, the license server responds with a user-visible HTML page. It is suggested to display an explanation of what the error condition was and why it was encountered.

There is no comprehensive set of errors. Protocol implementations will want to coordinate custom client and custom server implementations to handle their own custom error codes.

2.2.3.2.4 XML Schema for Version 7 License Response

The following is an XML schema for the version 7 license response packet per [XML], [XMLSCHEMA1/2], and [XMLSCHEMA2/2]. Where required, elements, attributes, and values are described in greater detail after the schema. This is not a strictly correct XML schema because the **LICENSE** element can be either a version 1 license or a version 7 license. The version attribute of the **LICENSE** element differentiates the two. The license server MAY mark nodes with either license version, or a combination of versions in different nodes. This allows the server to return licenses usable to multiple WMDRM protocol implementations simultaneously. This decision is entirely server-dependent on factors unrelated to the protocol.

If the version attribute indicates a version 7 license, then the first 80 bytes (called **EncRandNum**) of the **base64**-decoded version 7 license are used to decrypt the remainder of the base64-decoded bytes in the following manner.

Before encryption, **EncRandNum** contains the following byte values:

- byte 0: MUST be the value 0x07.
- byte 1: MUST be the value 0x01.
- bytes 2 – 8: Used as the initialization vector (IV) to create an RC4 key (KR).
- bytes 9 – 19: Not used.

Cryptographic sequence:

1. $\{EncRandNum\}_{KM}$
2. KR {version 7 license}

The encrypted **EncRandNum** and version 7 license are concatenated and then base64-encoded.

```
<?xml version="1.0" encoding="utf-8"?>
<xs:schema xmlns:xs="http://www.w3.org/2001/XMLSchema">
  <xs:element name="LICENSERESPONSE">
    <xs:complexType>
      <xs:sequence>
        <xs:element name="REVOCATION" minOccurs="0"
             maxOccurs="unbounded">
          <xs:complexType>
            <xs:simpleContent>
              <xs:extension base="xs:string">
                <!-- base64-encoded -->
                <xs:attribute name="type" use="required"
                     type="RevocationType" />
```

```
              </xs:extension>
            </xs:simpleContent>
          </xs:complexType>
        </xs:element>
        <xs:element name="LICENSE" minOccurs="1"
             maxOccurs="unbounded">
          <xs:complexType>
            <xs:simpleContent>
              <xs:extension base="xs:string">
                <!-- base64-encoded -->
                <xs:attribute name="version" use="required"
                     type="LicenseVersion" />
              </xs:extension>
            </xs:simpleContent>
          </xs:complexType>
        </xs:element>
      </xs:sequence>
    </xs:complexType>
  </xs:element>
  <xs:simpleType name="RevocationType">
    <xs:restriction base="xs:string">
      <xs:enumeration value="WMDRMNET" />
      <xs:enumeration value="DeviceRevocationList" />
      <xs:enumeration value="RevocationList" />
      <xs:enumeration
           value="{66DD5134-4E34-40ae-9D5D-13A112B7591F}" />
    </xs:restriction>
  </xs:simpleType>
  <xs:simpleType name="LicenseVersion">
    <xs:restriction base="xs:string">
      <xs:enumeration value="0.1.0.0" />
      <xs:enumeration value="2.0.0.0" />
    </xs:restriction>
  </xs:simpleType>
</xs:schema>
```

2.2.3.2.4.1 DRM Version 1 License Format Within a Version 7 License Response

If the **version** attribute of the **LICENSE** element is equal to "0.1.0.0", the **LICENSE** element contains a base64-encoded version 1 **CERTIFIED_LICENSE**.

2.2.3.2.4.2 DRM Version 7 License Format

If the **version** attribute of the **LICENSE** element is equal to "2.0.0.0", the **LICENSE** element is a version 7 license as described in the following.

A WMDRM: License Protocol version 7 license is represented in XML format. The schema for a version 7 license is as follows. This schema does not include the child elements of the META element because they are specified by the content provider and are outside the scope of this document.

```
<?xml version="1.0" encoding="utf-8"?>
<xs:schema xmlns:xs="http://www.w3.org/2001/XMLSchema">
  <xs:element name="LICENSE">
    <xs:complexType>
      <xs:sequence>
        <xs:element name="LICENSORINFO">
          <xs:complexType>
            <xs:sequence>
              <xs:element name="DATA">
                <xs:complexType>
                  <xs:sequence>
                    <xs:element name="LID" type="xs:string" />
                    <xs:element name="KID" type="xs:string" />
                    <xs:element name="ISSUEDATE" type="xs:string" />
```

```
                    <xs:element name="PRIORITY" type="xs:integer" />
                    <xs:element name="CONTENTPUBKEY" type="xs:string" />
                    <xs:element name="RevocationList" type="xs:string" /> <!-- base64-encoded
-->
                    <xs:element name="META" minOccurs="0">
                      <xs:complexType>
                        <xs:sequence>
                          <xs:any />
                        </xs:sequence>
                      </xs:complexType>
                    </xs:element>
                    <xs:choice minOccurs="0" maxOccurs="unbounded">
                    <xs:element name="ONSTORE">
                      <xs:complexType>
                        <xs:sequence>
                          <xs:element name="CONDITION" type="xs:string"
                              minOccurs="0" />
                          <xs:element name="ACTION" type="xs:string"
                              minOccurs="0" />
                        </xs:sequence>
                      </xs:complexType>
                    </xs:element>
                    <xs:element name="ONSELECT">
                      <xs:complexType>
                        <xs:sequence>
                          <xs:element name="CONDITION" type="xs:string" />
                          <xs:element name="ACTION" type="xs:string"
minOccurs="0" />
                        </xs:sequence>
                      </xs:complexType>
                    </xs:element>
                    <xs:element name="ONCLOCKROLLBACK" minOccurs="0">
                      <xs:complexType>
                        <xs:sequence>
                          <xs:element name="ACTION" type="xs:string"
minOccurs="0" />
                        </xs:sequence>
                      </xs:complexType>
                    </xs:element>
                    <xs:element name="ONACTION" minOccurs="0">
                      <xs:complexType>
                        <xs:sequence>
                          <xs:element name="RESTRICTIONS" minOccurs="0" maxOccurs="1">
                            <xs:complexType>
                              <xs:choice minOccurs="0" maxOccurs="6">
                                <xs:element name="ANALOGVIDEO" minOccurs="0" maxOccurs="1">
                                    <xs:complexType>
                                      <xs:attribute name="level" type="xs:integer" />
                                    </xs:complexType>
                                 </xs:element>
                                 <xs:element name="COMPRESSEDDIGITALAUDIO" minOccurs="0"
maxOccurs="1">
                                   <xs:complexType>
                                     <xs:attribute name="level" type="xs:integer" />
                                   </xs:complexType>
                                 </xs:element>
                                 <xs:element name="COMPRESSEDDIGITALVIDEO" minOccurs="0"
maxOccurs="1">
                                   <xs:complexType>
                                     <xs:attribute name="level" type="xs:integer" />
                                   </xs:complexType>
                                 </xs:element>
                                 <xs:element name="COPY" minOccurs="0" maxOccurs="1">
                                   <xs:complexType>
                                     <xs:attribute name="level" type="xs:integer" />
                                   </xs:complexType>
                                 </xs:element>
                                 <xs:element name="UNCOMPRESSEDDIGITALAUDIO" minOccurs="0"
maxOccurs="1">
```

```
                                <xs:complexType>
                                  <xs:attribute name="level" type="xs:integer" />
                                </xs:complexType>
                              </xs:element>
                              <xs:element name="UNCOMPRESSEDDIGITALVIDEO" minOccurs="0"
maxOccurs="1">
                                <xs:complexType>
                                  <xs:attribute name="level" type="xs:integer" />
                                </xs:complexType>
                              </xs:element>
                            </xs:choice>
                          </xs:complexType>
                        </xs:element>
                        <xs:element name="CONDITION" type="xs:string" />
                        <xs:element name="ACTION" type="xs:string"
minOccurs="0" />
                      </xs:sequence>
                      <xs:attribute name="type" type="ActionNameType" />
                    </xs:complexType>
                  </xs:element>
                  </xs:choice>
                  <xs:element name="ENABLINGBITS">
                    <xs:complexType>
                      <xs:sequence>
                        <xs:element name="ALGORITHM" minOccurs="0">
                          <xs:complexType>
                            <xs:sequence />
                            <xs:attribute name="type" use="required"
                                fixed="MSDRM" />
                          </xs:complexType>
                        </xs:element>
                        <xs:element name="PUBKEY">
                          <xs:complexType>
                            <xs:simpleContent>
                              <xs:extension base="xs:string"> <!-- base64-encoded -->
                                <xs:attribute name="type" use="required"
                                    fixed="machine" />
                              </xs:extension>
                            </xs:simpleContent>
                          </xs:complexType>
                        </xs:element>
                        <xs:element name="VALUE" type="xs:string" /> <!-- base64-encoded --
>
                        <xs:element name="SIGNATURE"
type="xs: string" /> <!-- base64-encoded -->
                      </xs:sequence>
                    </xs:complexType>
                  </xs:element>
                  <xs:element name="CONTENTREVOCATION" minOccurs="0"
                      maxOccurs="unbounded">
                    <xs:complexType>
                      <xs:sequence>
                        <xs:element name="DATA">
                          <xs:complexType>
                            <xs:sequence>
                              <xs:element name="SEQUENCENUMBER"
                                  type="xs:integer" />
                              <xs:element name="CONTENTPUBKEY"
                                  type="xs:string" /> <!-- base64-encoded -->
                              <xs:element name="LICENSESERVERPUBKEY"
                                  type="xs:string" /> <!-- base64-encoded -->
                              <xs:element name="CONDITION"
type="xs:string" />
                            </xs:sequence>
                          </xs:complexType>
                        </xs:element>
                        <xs:element name="SIGNATURE">
                          <xs:complexType>
                            <xs:sequence>
```

```
                                <xs:element name="HASHALGORITHM">
                                  <xs:complexType>
                                    <xs:sequence />
                                    <xs:attribute name="type" fixed="SHA"
                                        use="required" />
                                  </xs:complexType>
                                </xs:element>
                                <xs:element name="SIGNALGORITHM" minOccurs="0">
                                  <xs:complexType>
                                    <xs:sequence />
                                    <xs:attribute name="type" fixed="MSDRM" />
                                  </xs:complexType>
                                </xs:element>
                                <xs:element name="VALUE" type="xs:string" /> <!-- base64-
encoded -->
                              </xs:sequence>
                            </xs:complexType>
                          </xs:element>
                        </xs:sequence>
                      </xs:complexType>
                    </xs:element>
                    <xs:element name="REVOCATION" minOccurs="0" maxOccurs="unbounded">
                      <xs:complexType>
                        <xs:sequence>
                          <xs:element name="INDEX" type="xs:integer" />
                        </xs:sequence>
                        <xs:attribute name="type" type="RevocationType" />
                      </xs:complexType>
                    </xs:element>
                  </xs:sequence>
                </xs:complexType>
              </xs:element>
              <xs:element name="SIGNATURE">
                <xs:complexType>
                  <xs:sequence>
                    <xs:element name="HASHALGORITHM">
                      <xs:complexType>
                        <xs:sequence />
                        <xs:attribute name="type" fixed="SHA"
use="required" />
                      </xs:complexType>
                    </xs:element>
                    <xs:element name="SIGNALGORITHM" minOccurs="0">
                      <xs:complexType>
                        <xs:sequence />
                        <xs:attribute name="type" fixed="MSDRM" />
                      </xs:complexType>
                    </xs:element>
                    <xs:element name="VALUE" type="xs:string" /> <!-- base64-encoded -->
                  </xs:sequence>
                </xs:complexType>
              </xs:element>
              <xs:element name="CERTIFICATECHAIN">
                <xs:complexType>
                  <xs:sequence>
                    <xs:element name="CERTIFICATE" type="xs:string"
                        minOccurs="2" maxOccurs="unbounded" /> <!-- base64-encoded -->
                  </xs:sequence>
                  <xs:attribute name="type" fixed="MSDRM" />
                </xs:complexType>
              </xs:element>
            </xs:sequence>
          </xs:complexType>
        </xs:element>
      </xs:sequence>
      <xs:attribute name="version" use="required" fixed="2.0.0.0" />
    </xs:complexType>
  </xs:element>
  <xs:simpleType name="ActionNameType">
```

```
      <xs:restriction base="xs:string">
        <xs:enumeration value="CollaborativePlay" />
        <xs:enumeration value="Copy" />
        <xs:enumeration value="Play" />
        <xs:enumeration value="PlaylistBurn" />
        <xs:enumeration value="Print.redbook" />
        <xs:enumeration value="CREATE_PM_LICENSE" />
        <xs:enumeration value="Backup" />
        <xs:enumeration value="Restore" />
      </xs:restriction>
    </xs:simpleType>
    <xs:simpleType name="RevocationType">
      <xs:restriction base="xs:string">
        <xs:enumeration value="wmdrmnet" />
        <xs:enumeration value="device" />
      </xs:restriction>
    </xs:simpleType>
  </xs:schema>
```

For a sample version 7 license, see DRM Version 7 License Example.

2.2.3.2.5 ACTION

The ACTION element contains a single expression that defines the action to take after an event is raised.

The return value of the expression is ignored and can be of any type.

The action to take is contained in a **CDATA** block, as in the following example.

```
<ACTION>
 <![CDATA[
   deletelicense()
 <]]>
</ACTION>
```

2.2.3.2.6 ANALOGVIDEO

If the action is being taken on an analog video stream, the application MUST query the client DRM system for this value and perform additional security checks at content consumption time that conform to the WMDRM compliance rules.

2.2.3.2.7 CERTIFICATE

The CERTIFICATE element contains a base64-encoded version 1 **CERT** structure. During response parsing, the client will validate the format and signature using the definition of the **CERT** structure.

2.2.3.2.8 CERTIFICATECHAIN

The CERTIFICATECHAIN element specifies the **certificate** chain, which contains the credentials needed to issue licenses. This element is used to inform the **Digital Rights Management** client that the license server is authorized to issue licenses.

The first CERTIFICATE child element is the certificate issued by the root authority (CR). The second CERTIFICATE child element is the license server certificate (CS). Subsequent CERTIFICATE child elements comprise a certificate chain downward from CS. All valid version 7 and greater license servers MUST have at least two certificates that are maintained and retrieved by the service protocol implementation. These certificates are not processed on the server, but are included for reference by

the client to allow for signature validation of the license through a trusted certificate chain. The client will process each CERTIFICATE element and the CERTIFICATECHAIN element in a license response.

2.2.3.2.9 COMPRESSEDDIGITALAUDIO

If the action is being taken on a compressed digital audio stream, the application MUST query the client DRM system for value specified in the **level** attribute, and perform additional security checks at content consumption time that conform to the WMDRM compliance rules.

2.2.3.2.10 COMPRESSEDDIGITALVIDEO

If the action is being taken on a compressed digital video stream, the application security level must meet or exceed the value specified in the **level** attribute and perform additional security checks at content consumption time that conform to the WMDRM compliance rules.

2.2.3.2.11 CONDITION When Used Under the ONACTION, ONSELECT, and ONSTORE Elements

The <CONDITION> element is used under the <ONACTION>, <ONSELECT>, and <ONSTORE> elements. The <CONDITION> element is an expression that is evaluated to determine if an event is allowed. This evaluation is done before the event is performed.

A condition is an expression. The return value of the expression must be an integer (**LONG**).

A condition that evaluates to 0 is **FALSE**; one that evaluates to nonzero is **TRUE**. If the condition is **TRUE**, the event is allowed; otherwise, it is not.

The condition is contained in a **CDATA** block, as in the following example:

```
<CONDITION>
  <![CDATA[
    secstate.playcount > 0
   ]]>
</CONDITION>
```

2.2.3.2.12 CONDITION When Used Under the CONTENTREVOCATION/DATA Element

The <CONDITION> element is used within the <DATA> element when the <DATA> element is contained directly by a CONTENTREVOCATION element. The <CONDITION> element contains an expression that describes the condition under which the content owner selects licenses to be revoked. The default <CONDITION> element that the server generates specifies that a particular license can be deleted and that no other events can be allowed. No other conditions are generated. This element MUST be included in a license if the custom server business logic is issuing a license that performs content revocation.

The default <CONDITION> element is as follows:

```
<CONDITION>
  <![CDATA[ deletelicense();0 ]]>
</CONDITION>
```

2.2.3.2.13 CONTENTPUBKEY

The CONTENTPUBKEY element contains the base64-encoded public key of the content packager (KL). During content consumption, outside of the license acquisition protocol, this element is used to verify

the signature in the content header to detect tampering. These actions include performing an ECC verification operation over the content header.

2.2.3.2.14 CONTENTREVOCATION

The CONTENTREVOCATION element enables a content owner to disable licenses issued by that owner.

During response parsing, a client that receives a license with a CONTENTREVOCATION element must update an external data store of content revocation entries.

These entries are indexed with the CONTENTREVOCATION/CONTENTPUBKEY element value and are replaceable by responses with the same CONTENTPUBKEY but greater values of SERIALNUMBER.

After processing the CONTENTREVOCATION element from the license after acquisition, the element is no longer referenced.

2.2.3.2.15 COPY

If the action is being taken is a copy action, the application must query the client DRM system for the value specified in the **level** attribute and perform additional security checks at content consumption time that conform to the WMDRM compliance rules. Inclusion of this element is optional and entirely third-party server implementation-dependent.

2.2.3.2.16 ENABLINGBITS

The ENABLINGBITS element contains the encrypted key and information needed to unlock the content.

Cryptographic sequence:

1. <ENABLINGBITS> value = **base64**-encoded $\{length \mid key\}_{KM}$

Where length is a single BYTE representing the length of **key** in BYTES.

2.2.3.2.17 Events in DRM Licenses

An event is an element of a license. The ONSTORE, ONSELECT, and ONACTION events can have a <CONDITION> expression to be evaluated. All events can have an optional <ACTION> expression. A condition that evaluates to 0 is false; one that evaluates to nonzero is true. If the condition is true, the event is allowed and the action is taken.

A license can specify any of the following events.

Event	Description
ONSTORE	Raised when the license is stored.
ONSELECT	Raised when the license is selected.
ONCLOCKROLLBACK	Raised when the client detects a clock rollback.
ONACTION	Raised when the application queries or consumes a requested right.

2.2.3.2.18 Expressions in DRM Licenses

An expression in a **Digital Rights Management (DRM)** license is a combination of operators and identifiers that specifies a computation of a value or that designates a variable or a constant.

The <CONDITION> element (when used under the <ONACTION>, <ONSELECT>, and <ONSTORE> elements) and the ACTION element of a license are expressions. An expression can consist of the following items: Identifier (section 2.2.3.2.18.1), Function Symbol (section 2.2.3.2.18.2), Constant (section 2.2.3.2.18.3), Variable (section 2.2.3.2.18.4), and Final Value (section 2.2.3.2.18.5).

If more than one statement is used within an expression, each statement must be terminated with a semicolon (;). The evaluation of the last statement in a semicolon-delimited list is treated as the result for the evaluation of that list of expressions. For example, in the expression "1+2;4", the first statement evaluates to the value of three and the second statement evaluates to the value of four. The entire expression then evaluates to the value of four.

2.2.3.2.18.1 Identifier

An identifier is a sequence of characters that starts with an alphanumeric character and consists entirely of any number and combination of alphabetical characters, digits, underscores ("_"), and the dot (".") symbol. The characters are case-sensitive.

2.2.3.2.18.2 Function Symbol

An identifier is a function symbol if it is included in the list of predefined functions in DRM expressions.

2.2.3.2.18.3 Constant

A constant is an identifier of the **DATETIME**, **LONG**, or **STRING** data type, as specified in section 2.2.3.2.20.

2.2.3.2.18.4 Variable

An identifier is a variable if it is not a function symbol or a constant. Variables can be valid or invalid. A valid variable starts with one of the predefined prefix categories, which are specified in section 2.2.3.2.32. It is followed by a dot symbol and an attribute (for example, machine.datetime or app.count).

An invalid variable is one that is not in the list of prefixes. If a variable is not valid, the expression evaluation terminates and the expression is treated as false.

The value of a variable is retrieved from a specific location, which depends on the variable's category. For example, content.CID is retrieved from the content header, and license.LID is retrieved from the license.

The existence of a variable can be checked with the exists function.

2.2.3.2.18.5 Final Value

The final value of an expression is one of the three data types allowed in expressions: **DATETIME**, **LONG**, or **STRING**.

A <CONDITION> expression must result in a **LONG**. If the final value is 0, the condition is considered false; if the final value is non-zero, the condition is considered true.

The result of an ACTION expression can be of any type because the final value is ignored.

2.2.3.2.19 Operators in DRM Expressions

A **Digital Rights Management** license expression can include some of the operators that are found in the C programming language. All binary operators are used in infix notational form.

2.2.3.2.19.1 Operator Behavior

The behavior of operators in an expression depends on the types of the operands. The following table lists the allowed operators and the results they produce with operands of various types.

Operator	Operand1	Operand2	Result description
+	**LONG**		Unary plus.
+	**LONG**	**LONG**	Binary addition.
+	**STRING**	**STRING**	Concatenation of strings.
-	**LONG**		Unary minus.
-	**LONG**	**LONG**	Binary subtraction.
*	**LONG**	**LONG**	Binary multiplication.
/	**LONG**	**LONG**	Integer division (for example, 7/3 = 2).
%	**LONG**	**LONG**	Modulo operator (for example, 7 % 3 = 1).
++	**LONG**		Unary post-increment or pre-increment operator. Variable supports set operations.
--	**LONG**		Unary post-decrement or pre-decrement operator. Variable supports set operations.
=	**LONG**	**LONG**	Simple assignment.
=	**STRING**	**STRING**	Simple assignment.
=	**DATETIME**	**DATETIME**	Simple assignment.
< <= > >= == !=	**LONG**	**LONG**	Relational operator. Result is a **LONG** with a value of 0 or 1.
< <= > >= == !=	**STRING**	**STRING**	Relational operator. Result is a **LONG** with a value of 0 or 1.
< <= > >= == !=	**DATETIME**	**DATETIME**	Relational operator. Result is a **LONG** with a value of 0 or 1.
!	**LONG**		Unary Not. Result is a **LONG** with a value of 0 or 1.
&&	**LONG**	**LONG**	Logical AND. Result is a **LONG** with a value of 0 or 1. Shortcut evaluation is supported. For example, in the expression "a && b", if "a" is false, "b" is not evaluated.
||	**LONG**	**LONG**	Logical OR. Result is a **LONG** with a value of 0 or 1. Shortcut evaluation is supported. For example, in the expression "a || b", if "a" is true, "b" is not evaluated.
()			Allows precedence to be overridden.
?:	Any	Any	Conditional expression; for example, "(a < b)?c:d". If condition "a < b" is true, the value is "c", and "d" is not evaluated. If the condition "a < b" is false, the value is "d", and "c" is not evaluated.
,	Any	Any	Used to separate parameters in a function call or used in an expression to allow multiple statements to be evaluated. For example, "d = (a = b, c = e)" will assign the value of "e" to "d".

2.2.3.2.19.2 Operator Precedence

The following list shows the precedence of the operators in the table above, from highest to lowest.

- ( )
- FunctionCall
- ! ++ -- +(unary) -(unary)
- * / %
- + -
- < > <= =>
- == !=
- &&
- ||
- ?:
- =
- ;
- ,

2.2.3.2.20 Data Types in DRM Expressions

A **Digital Rights Management** expression, and the constants used in it, can have one of three data types: DATETIME (section 2.2.3.2.20.1), LONG (section 2.2.3.2.20.2), or STRING (section 2.2.3.2.20.3). Special rules for casting between data types (section 2.2.3.2.20.4) apply.

2.2.3.2.20.1 DATETIME Data Type

This data type is represented by a specific syntax, which takes one of the following formats:

#YYYYMMDDZ #

#YYYYMMDD HH:MM:SSZ#

The time is represented in Coordinated Universal Time (UTC) format. This portion is optional, and if it is missing, it is assumed to be zero. The Z at the end of the line indicates that the time is in UTC and is required even if the time portion is not present.

2.2.3.2.20.2 LONG Data Type

This data type is represented by an integer, which can be either decimal or hexadecimal. Hexadecimal values must be prefixed by the string "0x". This type is also used to represent Boolean values, where 0 is false and nonzero is true.

2.2.3.2.20.3 STRING Data Type

This data type is represented by double quotation marks (""). A **STRING** can include any character except the double quotation marks. However, a double quotation mark can be represented by using a backslash ("\"). For example, the string "ab\"cd" is rendered as "ab"cd".

The backslash, also referred to as the escape character, can represent a newline character when it is placed before the letter n, as in "\n". The escape character itself can be represented with two backslashes ("\\"). If the escape character is followed by any character other than n, ", or \, the pair of characters is replaced with the character that follows the backslash. For example, "\a" is equivalent to "a".

2.2.3.2.20.4 Casting Data Types

Implicit casting is not allowed. For example, the plus sign ("+") cannot be applied to **DATETIME** operands. However, the **DATETIME**, **LONG**, and **STRING** types can be used as casting operators. The following table lists the possible type conversions.

Conversion	Allowed
DATETIME to **LONG**	No
DATETIME to **STRING**	Yes
LONG to **DATETIME**	No
LONG to **STRING**	Yes
STRING to **LONG**	Yes
STRING to **DATETIME**	Yes

2.2.3.2.21 ISSUEDATE

The format of the <ISSUEDATE> element is the same as for the DATETIME Data Type as specified in section 2.2.3.2.20.1.

2.2.3.2.22 KID

The KID element contains the identifier of the key associated with a license.

The KID element MUST be a value that uniquely identifies content for which the license is issued. Use of a **base64**-encoded **globally unique identifier (GUID)** is recommended.

2.2.3.2.23 LICENSESERVERPUBKEY

The LICENSESERVERPUBKEY element contains the public key of the license server (KL_{pub}). The LICENSESERVERPUBKEY element is populated with custom data that is third party implementation-specific.

2.2.3.2.24 LICENSORINFO

The LICENSORINFO element contains a <DATA> element containing the license details, a <SIGNATURE> element, and a <CERTIFICATECHAIN> element.

2.2.3.2.25 LID

The LID element is a unique license identifier that is automatically created by the license generator. It must be a **curly braced GUID string**, as shown in the following example.

Example Code

```
<LID>{00000507-0000-0010-8000-00AA006D2EA4}</LID>
```

2.2.3.2.26 META

The use of this element is dependent on the client application. The child elements of META are optional and can contain metadata about the license. All child elements of the META element MUST be well-formed XML. **Digital Rights Management** does not depend on specific fields.

2.2.3.2.27 ONACTION

The ONACTION element is an event that is raised when the application consumes or queries a specific action right. For each action the server wishes to grant to the client, an <ONACTION> element must be created and inserted into the license. The contents of the <ONACTION> element, such as the conditions and <ACTION> elements, are third-party implementation-dependent.

An ONACTION event is required for each right that the license allows. If the event is missing, the right is not allowed.

The condition associated with this event is evaluated when the application consumes or queries the requested right. If the condition is true, the action is allowed; if the condition is false, the action is not allowed. If the condition is missing, it is assumed to be true.

If this event has an ACTION element, the action is taken after the right is consumed. If the application queries only the right, the action is not taken. The action's final value is ignored.

2.2.3.2.28 ONCLOCKROLLBACK

The ONCLOCKROLLBACK element is an event that is raised when the **Digital Rights Management (DRM)** client detects a clock rollback. It gives the license a means of reacting to the rollback.

This event has an action only; there are no conditions to evaluate.

When the DRM client detects a clock rollback, it gives every license that includes this event a chance to react to it. The license must specify the action to take; DRM simply evaluates the expression in the action.

For example, upon detecting clock rollback, a license can indicate that it must be deleted.

During license response creation, the server can include the <ONCLOCKROLLBACK> element in a license if, based on third-party implementation details, the server wishes the client to enforce per license clock rollback logic.

2.2.3.2.29 ONSELECT

The ONSELECT element is an event that is raised when the license is selected.

The condition for this event is evaluated when the license is selected. If the condition is true, the license can be selected; if it is false, the license cannot be selected. If the condition is missing, it is assumed to be true.

If this event has an ACTION element, the action is taken after the license is selected. The action's final value is ignored.

If the license does not include this event, no conditions for license selection are present, and the license is selected.

For more information about allowable conditions, see section 2.2.3.2.17.

During license response creation, the server must create an <ONSELECT> element and include it in the license to be returned to the client. The contents of the <ONSELECT> element are dependent on the third-party implementation details of each server.

2.2.3.2.30 ONSTORE

The ONSTORE element is an event that is raised when the license is stored.

The condition for this event is evaluated when the license is stored. If it is true, the license is stored; if it is false, the license is not stored. If the condition is missing, it is assumed to be true.

If this event has an ACTION element, the action is taken after the license is stored.

If the license does not include this event, no conditions for license storage are present, and the license is stored. The action's final value is ignored.

During license response creation, the server MUST create an <ONSTORE> element. The contents of the <ONSTORE> element are variable, based on third-party implementation details.

2.2.3.2.31 Predefined Functions in DRM Expressions

An event is an element of a license. Events can contain predefined function calls. Functions are evaluated as soon as the argument list is closed.

The following table lists and describes the supported functions.

Function	Arg1	Arg2	Arg3	Description
min	LONG	LONG		Returns the smaller of the two arguments. The result is **LONG**.
max	LONG	LONG		Returns the larger of the two arguments. The result is **LONG**.
long	STRING			Converts **STRING** to **LONG**. **STRING** has the syntax "[whitespace][sign][number]". The **number** attribute has at least one digit, which can be decimal or hexadecimal. No white space is allowed after **sign**, but trailing spaces are allowed.
long	LONG			Performs an identity operation.
string	LONG			Converts **LONG** to **STRING**.
string	STRING			Performs a string identity operation.
string	DATETIME			Converts **DATE** to **STRING**.
datetime	STRING			Converts **STRING** to **DATE**.
datetime	DATETIME			Performs an identity operation for **DATE**.
dateadd	STRING	LONG	DATETIME	Adds date elements. Arg1 can be "d" (days), "h" (hours), "n" (minutes), or "s" (seconds). The corresponding amount specified in Arg2 is added to the given datetime to get the target date and time. The result is a **DATETIME**.
datediff	STRING	DATETIME	DATETIME	Subtracts Arg2 from Arg3. The result is given in units, as indicated in Arg1. Arg1 can be "d", "h", "n", or "s". The result is a **LONG**.

Function	Arg1	Arg2	Arg3	Description
datepart	STRING	DATETIME		Returns an integer that represents the specified datepart (Arg1) of Arg2. Arg1 can be "y", "m", "d", "h", "n", or "s". The result is a **LONG**.
index	STRING	STRING		Returns the index of Arg1 in Arg2 if it is found. The first index is 0. If it is not found, return -1. The result is a **LONG**.
length	STRING			Returns the length of Arg1. The result is a **LONG**.
deletelicense				Deletes the current license, returning 1 if successful and 0 otherwise.
exists	variable			Determines whether a variable exists. Returns TRUE if successful and FALSE otherwise.
versioncompare	string	string		Compares two strings, treating them as versions. If they are not versions, the result is undefined. A version string has the form "<n>.<n>.<n>.<n>", where "<n>" is a number. The result is a LONG value: -1 if Arg1 is less than Arg2, 0 if Arg1 is equal to Arg2, and 1 if Arg1 is greater than Arg2.

2.2.3.2.32 Predefined Variables in DRM Expressions

Each license must create and access its own unique collection of attribute/value pairs. After being created, they cannot be deleted. A license can access only the attribute/value pairs that it created.

The attributes must belong to one of the following categories:

- drm
- drmk
- license
- pmlicense
- content
- machine
- server
- app (application)
- secstate (secure state in the client)

The following table enumerates all possible attributes that a license can expose.

Variable	Data type	Description
drm.version	**STRING**	The Digital Rights Management (DRM) version. This variable does not use the build number; it instead uses the hard-coded value in the client.
drm.bb.msdrm.version	**STRING**	The current security version of the WMDRM client. This value will be in the form "a.b.c.d". Because this information is not signed,

Variable	Data type	Description
		the value cannot be trusted by the client.
drmk.version	**STRING**	The version of the kernel mode DRM file (DRMK) on the client computer. This variable does not exist on a computer without DRMK. Use exists (drmk.version) to check for the presence of DRMK.
drmk.parameter	**STRING**	A string to use to set up DRMK. The string takes the form "attr=value;attr=value;" and so on. Supported attributes are **spdif**, **certs**, and **mindrmdriverlevel**. The default values for all are true, false, and 1000. If **certs** is true, audio and/or video output device drivers that are certified by Microsoft for use in the client-side media playback pipeline are required. The **mindrmlevel** attribute indicates the level of security that is needed for the drivers; do not use this attribute if **certs** is false. The **spdif** attribute allows the transfer of audio from one file to another without conversion to and from an analog format. If true, this type of transfer is allowed. If false, this type of transfer is not allowed.
machine.datetime	**DATETIME**	The time, in Coordinated Universal Time (UTC) format, based on the clock of the client computer.
app.count	**LONG**	The number of DRM certificates used currently by client.<4>
app.minseclevel	**LONG**	The minimum security level, which is computed from the supplied application certificates.
app.appsubjid	**LONG**	The application subject ID, provided in the supplied application certificates.
secstate.<attribute>	Any	The specified attribute value (for example, "secstate.firstdateofuse"). If the attribute does not exist, an error is returned. For assignments, the attribute is created if it does not already exist. Its type is the same as the type of the value assigned to it.
secstate.global.saveddatetime	**DATETIME**	The last saved clock time, as recorded by the DRM system. This is particularly useful for the ONCLOCKROLLBACK event. It is a read-only field for the license.
license.<attribute>	**STRING**	The value of the attribute in the license LICENSORINFO>/<DATA section (for example, "license.LID" or "license.KID"). The attribute is case-sensitive. It is possible that the value is an extensible markup language (XML) string. For example, license.META gives the entire XML string for the META section, without the META tags.
content.<attribute>	**STRING**	The value of the attribute in the content header DATA section (for example, "content.CID").
pmlicense.version (see note below)	**STRING**	The version of the Portable Media (PM) license being requested. This field is read-only and can be used in the CONDITION section of the rights that provide the PM license. Here CONDITION refers to the <CONDITION> element when used under the <ONACTION>, <ONSELECT>, or <ONSTORE> element.
pmlicense.rights (see note below)	**LONG**	The rights to use for generating the PM license if creating a license is allowed. Otherwise, this value is ignored. The default value is 0.
pmlicense.appseclevel (see note below)	**LONG**	The application security level to use for generating the PM license if creating a license is allowed. Otherwise, this value is ignored.

Variable	Data type	Description
		The default value is 0.
pmlicense.expirydate (see note below)	**DATETIME**	The date to use for generating the PM license if creating one is allowed. The default value is #19991231Z#.

Note A license server can issue a Portable Media license, which supports the moving of content to devices other than a PC. A server can automatically include this right when issuing a license for a request in which Play is the only requested right. The license is based on the DRM Version 1 License Format. In the DRM Version 7 License Format, the last four variables in the preceding table are used in the ONACTION element to provide the means of generating a PM license.

2.2.3.2.33 PRIORITY

The PRIORITY element indicates the priority of the license, helping the client choose the appropriate one when multiple licenses are available for the same content. A license with a higher priority is consumed before one with a lower priority.

The value of this element is an integer from 0 to 2147483647. The license enumeration process on the client selects licenses based on this priority.

2.2.3.2.34 PUBKEY

The PUBKEY element contains a base64-encoded public key to which the enabling bits are bound.

While not related to the acquisition protocol, the <PUBKEY> element is used to perform lookup of the appropriate client private key needed to decrypt the enabling bits in the case that the client has multiple keypairs available.

The server will populate the <PUBKEY> element with the public key of the client extracted from license request "{CLIENTINFO/CLIENTID/PKCERT/PK}".

Example code:

```
<PUBKEY type="machine">WEJKJKJKert==</PUBKEY>
```

2.2.3.2.35 RESTRICTIONS

The RESTRICTIONS element specifies additional restrictions that must be met before the enclosing <ONACTION> element is allowed. RESTRICTIONS elements are not processed during license acquisition, but rather during content consumption.

During license response creation, the server can include a <RESTRICTIONS> element in each <ONACTION> element with custom restriction logic that is server implementation-dependent.

2.2.3.2.36 REV_INFO

The REV_INFO structure describes the <RevocationList> versions known to either the client or server. The REV_INFO structure consists of the following data:

- One **WMDRMRLVIHEAD (section 2.2.3.2.46)** structure
- Zero or more **WMDRMRLVIVERSION (section 2.2.3.2.48)** structures
- One **WMDRMLRVISIGNATURE** structure
- One **WMDRMRLVICERTCHAIN (section 2.2.3.2.45)** structure

The number of **WMDRMRLVIVERSION** structures is given by the value of WMDRMRLVIHEAD.dwRecordCount.

2.2.3.2.37 REVOCATION

The REVOCATION element indicates the version of a given **CRL** within the <INDEX> element.

2.2.3.2.38 RevocationList

The <RevocationList> element contains a **base64**-encoded REV_INFO element.

2.2.3.2.39 SEQUENCENUMBER

The SEQUENCENUMBER element contains the sequence number of the content revocation. This number, which must be an integer, is generated sequentially by the content owner. It is used to determine whether the current content revocation overrides an existing one. Content revocation information with higher sequence numbers overrides content revocation information with lower sequence numbers. The SEQUENCENUMBER element is populated with custom data that is third party implementation-specific.

2.2.3.2.40 SIGNATURE When Used Under the CONTENTREVOCATION or LICENSORINFO Element

The <SIGNATURE> element is used within the <CONTENTREVOCATION> (section 2.2.3.2.14) and <LICENSORINFO> (section 2.2.3.2.24) elements. The <SIGNATURE> element contains a signature of the <DATA> element when used within the <CONTENTREVOCATION> and <LICENSORINFO> elements in the <VALUE> child element.

The content public key is signed by the license server. This signature is created using ECDSA over curve ECC_1. For more information about ECDSA, see [ELLIPTICCURVE-DSA]. This prevents messages about the revoked license from being altered by an external agent.

Cryptographic sequence:

<VALUE> contents = ["<DATA>" | <DATA> element contents | "</DATA>"]$_{KL}$

2.2.3.2.41 SIGNATURE When Used Under the ENABLINGBITS Element

The <SIGNATURE> element is used within the <ENABLINGBITS> (section 2.2.3.2.16) element. The <SIGNATURE> element contains a signature of the <VALUE> (section 2.2.3.2.44) element contents when the <VALUE> element is contained within the <ENABLINGBITS> element. The server MUST generate a valid ECDSA signature during response creation.

This signature is created using ECDSA over curve **ECC**$_1$. For more information about ECDSA, see [ELLIPTICCURVE-DSA].

Cryptographic sequence:

1. signature = [content decryption key]$_{KL}$

<SIGNATURE> contents = **base64**-encoded signature

2.2.3.2.42 UNCOMPRESSEDDIGITALAUDIO

If the action being taken is on an uncompressed digital audio stream, the client MUST use the value parsed from the UNCOMPRESSEDDIGITALAUDIO element to perform additional security checks at content consumption time that conform to the WMDRM compliance rules.

2.2.3.2.43 UNCOMPRESSEDDIGITALVIDEO

If the action is being taken on an uncompressed digital video stream, the client MUST use the value parsed from the UNCOMPRESSEDDIGITALVIDEO element to perform additional security checks at content consumption time that conform to the WMDRM compliance rules.

2.2.3.2.44 VALUE

The VALUE element is used within the ENABLINGBITS element. The <VALUE> element contains the base64-encoded, encrypted content decryption key, although not processed during the license acquisition protocol, but rather during content consumption.

The <VALUE> element is populated by the server using a key known to the server, but is third-party implementation-defined.

The key is encrypted with the computer's public key so that only the client that requested this license can decrypt it.

Cryptographic sequence:

<VALUE> contents = $\{\text{content decryption key}\}_{KM}$

2.2.3.2.45 WMDRMRLVICERTCHAIN

The **WMDRMRLVICERTCHAIN** structure describes the **certificate** chain used to sign the REV_INFO structure.

```
typedef struct {
  DWORD dwCount[];
  BYTE rgCertChain[];
} WMDRMRLVICERTCHAIN;
```

dwCount: Length, in bytes, of the data in **rgCertChain**.

rgCertChain: Certificate chain representing the entity used to sign the REV_INFO structure.

2.2.3.2.46 WMDRMRLVIHEAD

The **WMDRMRLVIHEAD** structure contains the header for a REV_INFO structure

```
typedef struct {
  BYTE dwId[4];
  DWORD dwLength[];
  BYTE bFormatVersion[];
  BYTE bReserved[3];
  DWORD dwRIV[];
  FILETIME ftIssuedTime[];
  DWORD dwRecordCount[];
} WMDRMRLVIHEAD;
```

dwId: Signature value indicating the start of this structure. MUST be equal to the **ASCII** character sequence { "R", "L", "V", "I" }.

dwLength: The length, in bytes of signed data within the REV_INFO structure, starting with the beginning of this structure.

bFormatVersion: The version of this structure. It MUST be 0x01.

bReserved: Padding, not used.

dwRIV: Version of the enclosing REV_INFO structure.

ftIssuedTime: Date and time that the enclosing REV_INFO structure was generated.

dwRecordCount: The count of **WMDRMRLVIVERSION** structures following this structure. This value MUST be greater than or equal to zero and MUST be less than or equal to 10.

2.2.3.2.47 WMDRMRLVISIGNATURE

The **WMDRMRLVISIGNATURE** structure describes the cryptographic signature associated with a given REV_INFO structure.

```
typedef struct {
  BYTE bSignatureType[];
  BYTE bSignature[128];
} WMDRMRLVISIGNATURE;
```

bSignatureType: MUST be 0x02.

bSignature: RSA/SHA1 signature value.

2.2.3.2.48 WMDRMRLVIVERSION

The **WMDRMRLVIVERSION** structure describes the version number associated with a given **revocation** list.

```
typedef struct {
  GUID RLVIGUID[];
  ULONGLONG RLVIVERSION[128];
} WMDRMRLVIVERSION;
```

RLVIGUID: MUST be 0x02.

RLVIVERSION: Version number of the revocation list.

2.2.4 DRM Version 11 Data Types

The following structures and algorithm are specific to version 11 of the WMDRM: License Protocol.

2.2.4.1 DRM Version 11 License Request

The Digital Rights Management (DRM) version 11 license request is almost identical to the version 7 license request. In version 11 license request packets, the <CLIENTINFO> element has two additional child elements, <MACHINECERTIFICATE> and <REVINFO>, described below.

The following is the XML schema for the version 11 license request packet. Where required, elements, attributes, and values are described in greater detail after the schema.

```
<xs:schema xmlns:xs="http://www.w3.org/2001/XMLSchema">
  <xs:element name="LICENSEREQUEST">
    <xs:complexType>
      <xs:sequence>
        <xs:element name="V1CHALLENGE">
          <xs:simpleType>
            <xs:restriction base="xs:base64Binary" />
          </xs:simpleType>
        </xs:element>
        <xs:element name="ACTIONLIST" minOccurs="0">
```

```
            <xs:complexType>
              <xs:sequence>
                <xs:element name="ACTION" type="ActionNameType"
                            minOccurs="1" maxOccurs="10" />
              </xs:sequence>
            </xs:complexType>
          </xs:element>
          <xs:element name="CLIENTINFO" minOccurs="0">
            <xs:complexType>
              <xs:sequence>
                <xs:element name="CLIENTID" type="xs:base64Binary" />
                <xs:element name="MACHINECERTIFICATE"
                     type="xs:base64Binary" />
                <xs:element name="REVINVO" type="xs:base64Binary" />
                <xs:element name="CLIENTVERSION" type="xs:string" />
                <xs:element name="SECURITYVERSION" type="xs:string" />
                <xs:element name="APPSECURITY" type="xs:string" />
                <xs:element name="SUBJECTID1" type="xs:integer" />
                <xs:element name="SUBJECTID2" type="xs:integer" />
                <!-- SUBJECTID2 tag must be present;
                     content is optional. -->
                <xs:element name="DRMKVERSION" type="xs:string"
                     minOccurs="0" />
              </xs:sequence>
            </xs:complexType>
          </xs:element>
          <xs:element name="WRMHEADER">
            <xs:complexType>
              <xs:sequence>
                <!-- content varies, depending on media
                     file header information. -->
                <xs:any />
              </xs:sequence>
            </xs:complexType>
          </xs:element>
        </xs:sequence>
        <xs:attribute name="version" use="required" fixed="2.0.0.0" />
      </xs:complexType>
    </xs:element>
    <xs:simpleType name="ActionNameType">
      <xs:restriction base="xs:string">
        <xs:enumeration value="Play" />
        <xs:enumeration value="Print.redbook" />
        <xs:enumeration value="CREATE_PM_LICENSE" />
        <xs:enumeration value="Backup" />
        <xs:enumeration value="Restore" />
        <xs:enumeration value="CollaborativePlay" />
        <xs:enumeration value="Copy" />
        <xs:enumeration value="Transfer.SDMI" />
        <xs:enumeration value="Transfer.NONSDMI" />
        <xs:enumeration value="PlaylistBurn" />
      </xs:restriction>
    </xs:simpleType>
  </xs:schema>
```

2.2.4.1.1 MACHINECERTIFICATE

The MACHINECERTIFICATE element contains a base64-encoded copy of the machine certificate (CM). CM is a **certificate** in XML format.

The following is the XML schema for the MACHINECERTIFICATE XML object. Note that this schema defines several elements already defined in [RFC3275]. Specifically, this schema defines the <KeyValue>, <SignedInfo>, <SignatureValue>, and <KeyInfo> elements and their respective child elements. Their definition in this schema is intended only to assist in understanding the processing rules in section 3.2.5.3. Within an instance of the MACHINECERTIFICATE XML object, these elements are not to be prefixed with a namespace qualifier. See the example following the schema.

```
<?xml version="1.0" encoding="utf-8"?>
<xs:schema xmlns:xs="http://www.w3.org/2001/XMLSchema">
  <xs:element name="CertificateCollection">
    <xs:complexType>
      <xs:sequence>
        <xs:element name="Certificate" maxOccurs="unbounded">
          <xs:complexType>
            <xs:sequence>
              <xs:element name="Data">
                <xs:complexType>
                  <xs:sequence>
                    <xs:element name="SecurityLevel" type="xs:string" />
                    <xs:element name="PublicKey">
                      <xs:complexType>
                        <xs:sequence>
                          <xs:element name="KeyValue">
                            <xs:complexType>
                              <xs:sequence>
                                <xs:element name="Modulus" type="xs:base64Binary" />
                                <xs:element name="Exponent" type="xs:base64Binary"
fixed="AQAB" />
                              </xs:sequence>
                            </xs:complexType>
                          </xs:element>
                        </xs:sequence>
                      </xs:complexType>
                    </xs:element>
                  </xs:sequence>
                </xs:complexType>
              </xs:element>
              <xs:element name="Signature">
                <xs:complexType>
                  <xs:sequence>
                    <xs:element name="SignedInfo">
                      <xs:complexType>
                        <xs:sequence>
                          <xs:element name="CanonicalizationMethod">
                            <xs:complexType>
                              <xs:attribute name="Algorithm" type="xs:string" use="required"
fixed="http://www.w3.org/TR/2001/REC-xml-c14n-20010315" />
                            </xs:complexType>
                          </xs:element>
                          <xs:element name="SignatureMethod">
                            <xs:complexType>
                              <xs:attribute name="Algorithm" type="xs:string" use="required"
fixed="http://www.w3.org/2000/09/xmldsig#rsa-sha1" />
                            </xs:complexType>
                          </xs:element>
                          <xs:element name="Reference">
                            <xs:complexType>
                              <xs:sequence>
                                <xs:element name="Transforms">
                                  <xs:complexType>
                                    <xs:sequence>
                                      <xs:element name="Transform" minOccurs="2"
maxOccurs="2">
                                        <xs:complexType>
                                          <xs:attribute name="Algorithm" use="required"
type="TransformAlgorithm"/>
                                        </xs:complexType>
                                      </xs:element>
                                    </xs:sequence>
                                  </xs:complexType>
                                </xs:element>
                                <xs:element name="DigestMethod">
                                  <xs:complexType>
                                    <xs:attribute name="Algorithm" use="required"
fixed="http://www.w3.org/2000/09/xmldsig#sha1" />
                                  </xs:complexType>
```

```
                                </xs:element>
                                <xs:element name="DigestValue" type="xs:base64Binary" />
                              </xs:sequence>
                            </xs:complexType>
                          </xs:element>
                        </xs:sequence>
                      </xs:complexType>
                    </xs:element>
                    <xs:element name="SignatureValue" type="xs:base64Binary" />
                    <xs:element name="KeyInfo">
                      <xs:complexType>
                        <xs:sequence>
                          <xs:element name="KeyValue">
                            <xs:complexType>
                              <xs:sequence>
                                <xs:element name="RSAKeyValue">
                                  <xs:complexType>
                                    <xs:sequence>
                                      <xs:element name="Modulus" type="xs:base64Binary" />
                                      <xs:element name="Exponent" type="xs:base64Binary"
fixed="AQAB" />
                                    </xs:sequence>
                                  </xs:complexType>
                                </xs:element>
                              </xs:sequence>
                            </xs:complexType>
                          </xs:element>
                        </xs:sequence>
                      </xs:complexType>
                    </xs:element>
                  </xs:sequence>
                </xs:complexType>
              </xs:element>
              <xs:element name="ManufacturerData" minOccurs="0">
                <xs:complexType>
                  <xs:sequence>
                    <xs:element name="ManufacturerName" type="xs:string" />
                  </xs:sequence>
                </xs:complexType>
              </xs:element>
            </xs:sequence>
            <xs:attribute name="Version" use="required" fixed="2.0" />
          </xs:complexType>
        </xs:element>
      </xs:sequence>
      <xs:attribute name="Version" use="required" fixed="2.0" />
    </xs:complexType>
  </xs:element>
  <xs:simpleType name="TransformAlgorithm">
    <xs:restriction base="xs:string">
      <xs:enumeration value="http://www.microsoft.com/DRM/CERT/v2/Data" />
      <xs:enumeration value="http://www.w3.org/TR/2001/REC-xml-c14n-20010315" />
    </xs:restriction>
  </xs:simpleType>
</xs:schema>
```

The following is an example of the MACHINECERTIFICATE XML object.

```
<?xml version="1.0" encoding="UTF-8" ?>
<c:CertificateCollection xmlns="http://www.w3.org/2000/09/xmldsig#"
     xmlns:c="http://schemas.microsoft.com/DRM/2004/02/cert"
     c:Version="2.0">
  <c:Certificate c:Version="2.0"
       xmlns:c="http://www.microsoft.com/DRM/2004/02/cert" >
    <c:Data xmlns:c="http://www.microsoft.com/DRM/2004/11/cert"
         xmlns:l="http://www.microsoft.com/DRM/2004/11/mslp"
```

```
          xmlns:ds="http://www.w3.org/2000/09/xmldsig#">
        <c:SecurityLevel>a.b.c.d</c:SecurityLevel>
        <c:PublicKey>
          <KeyValue>
            <RSAKeyValue>
              <Modulus> <!-- base64-encoded modulus value --> </Modulus>
              <Exponent>AQAB</Exponent>
            </RSAKeyValue>
          </KeyValue>
        </c:PublicKey>
      </c:Data>
      <c:Signature>
        <SignedInfo>
          <CanonicalizationMethod
               Algorithm="http://www.w3.org/TR/2001/REC-xml-c14n-20010315" />
          <SignatureMethod
               Algorithm="http://www.w3.org/2000/09/xmldsig#rsa-sha1" />
          <Reference>
            <Transforms>
              <Transform
                   Algorithm="http://www.microsoft.com/DRM/CERT/v2/Data" />
              <Transform
                   Algorithm="http://www.w3.org/TR/2001/REC-xml-c14n-20010315" />
            </Transforms>
            <DigestMethod
                 Algorithm="http://www.w3.org/2000/09/xmldsig#sha1" />
            <DigestValue> <!-- base64-encoded digest --> </DigestValue>
          </Reference>
        </SignedInfo>
        <SignatureValue> <!-- base64-encoded signature -->
             </SignatureValue>
        <KeyInfo> <!-- key used to sign the SignedInfo data. -->
          <KeyValue>
            <RSAKeyValue>
              <Modulus> <!-- base64-encoded modulus value --> </Modulus>
              <Exponent>AQAB</Exponent>
            </RSAKeyValue>
          </KeyValue>
        </KeyInfo>
      </c:Signature>
    </c:Certificate>
  </c:CertificateCollection>
```

Certificates are listed in c:CertificateCollection from leafmost to rootmost certificate. The leafmost certificate is signed by the certificate next closer to the rootmost certificate. The rootmost certificate is signed by the Microsoft Root WMDRM Certificate, represented by the following public key:

```
pjoeWLSTLDonQG8She6QhkYbYott9fPZ8tHdB128ZETcghn5KHoyin7HkJEcPJ0Eg4UdSva0KDIYDjA3EXd69R3CN2Wp/
QyOo0ZPYWYp3NXpJ700tKPgIplzo5wVd/69g7j+j8M66W7VNmDwaNs9mDc1p2+VVMsDhOsV/Au6E+E=
```

Follow these steps to evaluate the signature of a given certificate:

1. Accumulate a **SHA-1 hash** over <c:Data> including the <c:Data ... > and </c:Data> end tags.
2. Compare hash to <DigestValue>.
3. Verify signature over <SignedInfo> with the RSA public key using the RSA/SHA1 algorithm.

Note that the value for <Modulus> must be reversed after **base64**-decoding.

2.2.4.1.2 REVINFO

The <REVINFO> element contains a **base64**-encoded copy of the **revocation** list (REV_INFO) known to the client application. This list is persisted in an application-specific manner and SHOULD be retrieved and included in the challenge if it exists. If this revocation list is out of date, the license server can provide an updated revocation list in the <REVOCATION> element.

2.2.4.1.3 ACTION

The <ACTION> element contains the action rights that the client is requesting. The meaning of the element contents is described in the following table.

Predefined string	Meaning
Play	Play content on the client computer.
Print.redbook	Burn content to a CD.
CREATE_PM_LICENSE	Transfer content to a portable device.
Backup	Permit backup of the license.
Restore	Allow the license to be restored from another location.
CollaborativePlay	Play the file as part of a collaborative peer-to-peer networking scenario.
Copy	Copy the file to a device.
Transfer.SDMI	Copy file to an **SDMI** device.
Transfer.NONSDMI	Copy file to a non-SDMI device.
PlaylistBurn	Copy the file to Red Book audio CD as part of a playlist.

2.2.4.2 DRM Version 11 License Response

The version 11 license response is identical to the version 7 license response.

3 Protocol Details

3.1 Client Details

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.

Note The following variables are logical, abstract parameters that an implementation MUST maintain and expose to provide to the higher layer. How these variables are maintained and exposed is up to the client protocol implementation.

ClientCertificate: A PKCERT structure that contains client machine-specific cryptographic data.

RightsRequested: A list of action strings in ActionNameType; see section 2.2.3.1.3.

ClientAppSec: A 4 byte value as described in section 2.2.2.1 under AppSec.

KeyIdentifier: A base-64 encoded string related to the content as described in section 2.2.2.1.

ClientVersion: This string variable maps to the element in the protocol named clientversion and described in section 2.2.3.1.3.5.

SecurityVersion: This string variable maps to the element in the protocol named securityversion and described in section 2.2.3.1.3.8.

SubjectId1: This string variable maps to the element in the protocol named subjectid1 and described in section 2.2.3.1.3.9.

SubjectId2: This string variable maps to the element in the protocol named subjectid2 and described in section 2.2.3.1.3.10.

DrmKVersion: This string variable maps to the element in the protocol named drmkversion and described in section 2.2.3.1.3.6.

RevocationInfo: This string variable maps to the element in the protocol named revocationinfo and described in section 2.2.3.1.3.7.

WRMHeader: This variable string maps to the element in the protocol named wrmheader and described in section 2.2.3.1.3.12.

RevInfo: This string variable maps to the element in the protocol named revinfo and described in section 2.2.4.1.2.

MachineInfo: This string variable maps to the element in the protocol named MACHINEINFO and described in section 2.2.4.1.1.

3.1.2 Timers

None.

3.1.3 Initialization

None.

3.1.4 Higher-Layer Triggered Events

None.

3.1.5 Message Processing Events and Sequencing Rules

3.1.5.1 DRM Version 1 Client Message Processing Events and Sequencing Rules

3.1.5.1.1 Request Behavior

The DRM Version 1 License Request packet is used by the client to request a license for protected media content.

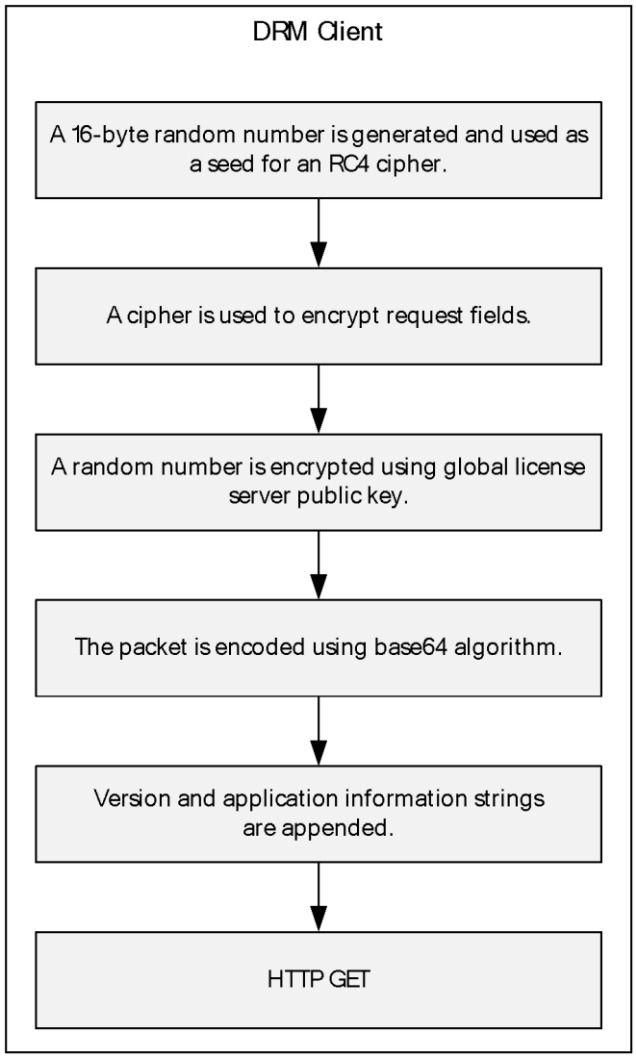


Figure 3: DRM Version 1 License Request packet process

The client generates a license request and sends it to a license server. The request is a binary string that is partially **RC4**-encrypted and then encoded by using the base64 encoding algorithm.

The higher layer MUST set appropriate values for **ClientCertificate**, **RightsRequested**, **ClientAppSec**, and **KeyIdentifier**. The client generates a 16-byte random number as **SessionKey** (internal state). The client populates a DRM Version 1 License Request packet with the following mapping:

ClientCertificate ->**pkcert**

RightsRequested ->Mapping of the rights strings to a 4 byte array of **Rights** described in section 2.2.3.1.3

ClientAppSec ->**AppSec**

KeyIdentifier ->**KeyID**

SessionKey is encrypted with a global license server public key KS and placed in the packet as **EncRandNum**.

SessionKey is used as the seed to RC4, and all packet elements after **EncRandNum** are encrypted with the resulting RC4 stream cipher.

After encryption, the string is base64-encoded. The client application then appends the string "&DRMVer=1.4". Note that the value for "DRMVer" can vary based on the client application. Depending on the implementation, the client application can append "&embedded=true" if the player is embedded in an HTML page; it can append "&embedded=false" if the player is not embedded in an HTML page; or it can also append nothing. The resulting string is a uniform resource locator (URL) that is sent to the license server as an HTTP GET request.

3.1.5.1.2 Response Behavior

The response from a server is an HTML document as described in DRM Version 1 License Response (section 2.2.2.2). The client must extract the string value labeled "base64-encoded CERTIFIED_LICENSE", base64 decode the **CERTIFIED_LICENSE** structure, and return that value to the higher layer.

3.1.5.2 DRM Version 7 Client Message Processing Events and Sequencing Rules

3.1.5.2.1 Request Behavior

The Digital Rights Management (DRM) version 7 License Request packet is used by the client to request a license for protected media content.

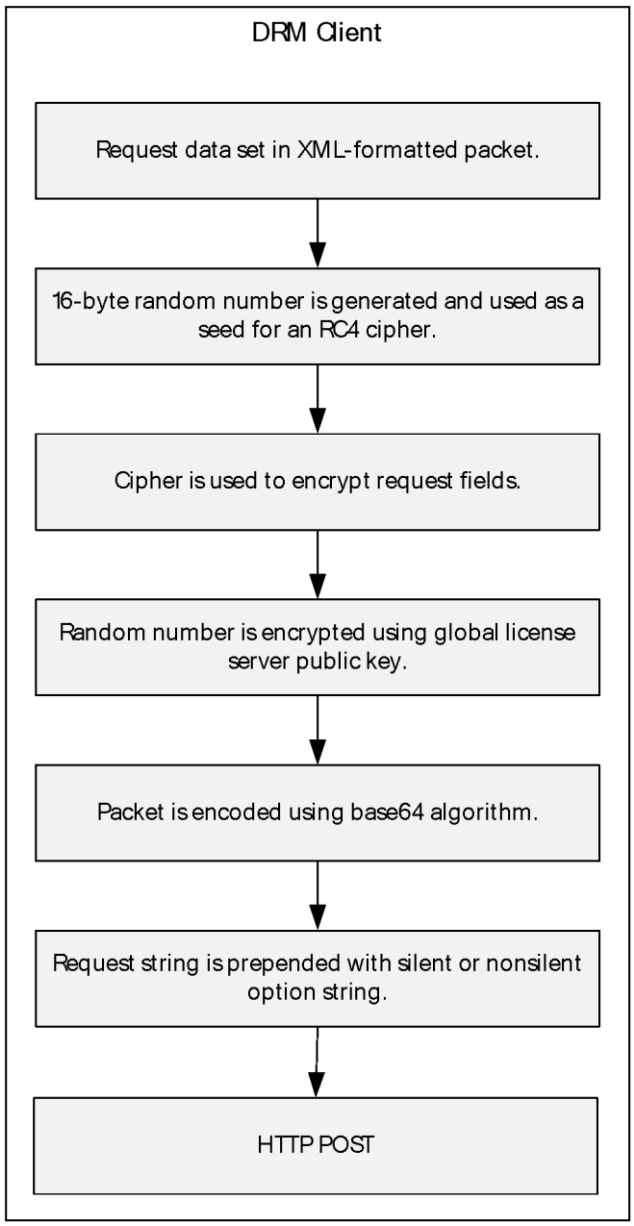


Figure 4: DRM version 7 request packet process

The higher layer MUST set appropriate values for **ClientCertificate**, **RightsRequested**, **ClientAppSec**, **ClientVersion**, **SecurityVersion**, **SubjectId1**, **SubjectId2**, **DrmKVersion**, **RevocationInfo**, and **WRMHeader**. The client generates a 16-byte random number as **SessionKey**. The client populates a DRM Version 1 License Request packet with the following mapping:

ClientCertificate -> **pkcert**

RightsRequested ->A sequence of <ActionNameType> elements under the <ACTION> element

ClientAppSec ->**AppSec**

KeyID is set to all zeros

The client populates a V7 license challenge request mapping the following state variables to the appropriate XML elements:

ClientVersion -> <CLIENTVERSION>

SecurityVersions -> <SECURITYVERSION>

ClientAppSec -> <APPSECURITY>

SubjectId1 -> <SUBJECTID1>

SubjectId2 -> <SUBJECTID2>

DrmKVersion -> <DRMKVERSION>

WRMHeader -> <WRMHEADER>

A **CLIENTID** structure is populated by the client as described in section **2.2.3.1.3.4** using **ClientCertificate** as **CLIENTID.pkcert**, and **SessionKey** is encrypted with KS and included as **CLIENTID.EncRandNum**. The populated **CLIENTID** structure is base64 encoded and included in the request packet as XML element CLIENTID.

The version 1 request packet is base64 encoded and included in the version 7 license request packet as the XML element <V1CHALLENGE>.

The client-generated license request is sent to a license server by using an HTTP POST request.

3.1.5.2.2 Response Behavior

The license response returned from the server is described in section 2.2.3.1.3. The client MUST parse the XML document and process the **LICENSE** and **REVOCATION** nodes as described in LICENSERESPONSE.LICENSE nodes (section 3.1.5.2.2.1) and LICENSERESPONSE.Revocation nodes (section 3.1.5.2.2.2).

3.1.5.2.2.1 LICENSERESPONSE.LICENSE nodes

For each **LICENSE** node found, the client MUST inspect the version attribute. The client MUST **base64** decode the data in the **LICENSE** node and return the decoded data and the value in the version attribute to the higher layer so that the higher layer is aware of the license version being returned.

The higher layer is responsible for any and all usages of the return license. Discussion of this higher-layer responsibility is out of scope of this document.

3.1.5.2.2.2 LICENSERESPONSE.Revocation nodes

For each Revocation node found, the client must inspect the type attribute. The contents of the node must be **base64** decoded and returned with the value of the type attribute to the higher layer.

3.1.5.3 DRM Version 11 Client Message Processing Events and Sequencing Rules

3.1.5.3.1 Request Behavior

The Digital Rights Management (DRM) version 11 license request packet is used by the client to request a license for protected media content.

The version 11 request processing sequence is identical to the version 7 sequence with the following additions:

- The higher layer MUST additionally set appropriate values for MachineInfo and RevInfo.
- In addition to the V7 XML population, the client performs the following state mappings into the V11 request:
  - **MachineInfo** -> MACHINEINFO
  - **RevInfo** -> REVINFO

3.1.5.3.2 Response Behavior

The V11 response processing is identical to the V7 response processing.

3.1.6 Timer Events

None.

3.1.7 Other Local Events

None.

3.2 Server Details

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.

Note The following variables are logical, abstract parameters that an implementation MUST maintain and expose to provide to the higher layer. How these variables are maintained and exposed is up to the server protocol implementation.

ClientCertificate: A PKCERT structure that contains client machine-specific cryptographic data.

RightsRequested: A list of action strings in ActionNameType; see section 2.2.3.1.3.

ClientAppSec: A 4-byte value as described in section 2.2.2.1 under AppSec.

KeyIdentifier: A base-64 encoded string related to the content as described in section 2.2.2.1.

ClientVersion: This string variable maps to the element in the protocol named clientversion and described in section 2.2.3.1.3.5.

SecurityVersion: This string variable maps to the element in the protocol named securityversion and described in section 2.2.3.1.3.8.

SubjectId1: This string variable maps to the element in the protocol named subjectid1 and described in section 2.2.3.1.3.9.

SubjectId2: This string variable maps to the element in the protocol named subjectid2 and described in section 2.2.3.1.3.10.

DrmKVersion: This string variable maps to the element in the protocol named drmkversion and described in section 2.2.3.1.3.6.

RevocationInfo: This string variable maps to the element in the protocol named revocationinfo and described in section 2.2.3.1.3.7.

WRMHeader: This variable string maps to the element in the protocol named wrmheader and described in section 2.2.3.1.3.12.

RevInfo: This string variable maps to the element in the protocol named revinfo and described in section 2.2.4.1.2.

MachineInfo: This string variable maps to the element in the protocol named MACHINEINFO and described in section 2.2.4.1.1.

3.2.1.1 TransmitLicensesToClient

The **TransmitLicensesToClient** abstract interface is provided by the protocol implementation for the higher layer to call. The higher layer constructs the licenses and uses this abstract interface to have the Digital Rights Management License Protocol send them to the client.

```
TransmitLicensesToClient(LicenseToSend [] Licenses)
```

Licenses: An array of licenses that need to be transmitted to the client. The length of each license is variable, and the number of licenses is variable. This parameter uses the **LicenseToSend** structure defined in section **2.2.1.7**.

3.2.2 Timers

None.

3.2.3 Initialization

Certificate revocation within the Windows Media DRM ecosystem is handled by means of **certificate revocation lists (CRLs)**. CRLs flow from a public Microsoft server, the "enrollment server", to the license server, and then to the client. Each CRL is identified by a **GUID** or text string and contains a version number and a list of hashes of revoked **certificates**. A **revocation** version information list (REV_INFO) contains a list of CRL versions and is itself versioned with a revocation information version (RIV). Hence, each time a new CRL version is released, the current RIV is also increased.

The client application identity is given via CA. This is transmitted from the client to the server within all license acquisition requests.

The client application maintains a list of CRLs known to it and the associated REV_INFO and RIV. If the RIV reported by the client is lower than the RIV known to the license server, the license server MUST transmit the latest REV_INFO and CRLs to the client within the license acquisition response.

To transmit the latest REV_INFO and CRLs to the client, it is not necessary to understand the entire REV_INFO structure or the format of the CRL data. It is only necessary to understand the REV_INFO.WMDRMRLVIHEAD.dwRIV and compare that to the RIV reported by the client.

Given the previous statements a server implementation must be initialized with the CRLs, REV_INFO and client white list values from the aforementioned enrollment server, before it can successfully validate and interact with a Windows Media DRM client implementation.

3.2.3.1 Retrieving Revocation Data from the Enrollment Server

The current client **certificate** white list, current REV_INFO data, and current **CRLs** are all available in **base64**-encoded form within the body of the **HTTP** response to an HTTP GET of

http://licenseserver.windowsmedia.com/v2revocation.asp. The license server MUST retrieve this data at least once every 30 days to ensure that the **revocation** data transmitted to the client is no older than 30 days.

3.2.3.1.1 Client Certificate White List

The Client Certificate White List entries are in the following format within the response:

```
enrollobj.StoreVerificationKey("client version", "pubkey");
```

client version: A numeric version string of the form "a.b.c.d". The value provided by the client in CLIENTID.pkcert.pk.version maps to one of these values in the white list.

pubkey: A **base64**-encoded **PUBKEY** identifying the client **certificate** associated with the given client version. The value provided by the client in CLIENTID.pkcert.pk.pubkey matches the value associated with the given client version.

3.2.3.1.2 Revocation Information List

The REV_INFO data is in the following format within the response:

```
enrollobj.StoreRevocationInformation("REV_INFO");
```

REV_INFO: A **base64**-encoded REV_INFO structure.

3.2.3.1.3 Certificate Revocation List

Multiple **certificate revocation lists (CRLs)** are present within the response and are in one of the following formats:

```
enrollobj.StoreRevocationList("CRL");
enrollobj.StoreRevocationLists("GUID", "CRL");
enrollobj.StoreNamedRevocations("name", "CRL");
```

CRL: A **base64**-encoded CRL.

GUID: The name of the CRL in string form. In this case, the CRL name is a **GUID** string.

name: The name of the CRL in string form.

3.2.4 Higher-Layer Triggered Events

None.

3.2.5 Message Processing Events and Sequencing Rules

3.2.5.1 DRM Version 1 Server Message Processing Events and Sequencing Rules

The Digital Rights Management (DRM) Version 1 License Response packet is used by the license server to send a license for content to a client.

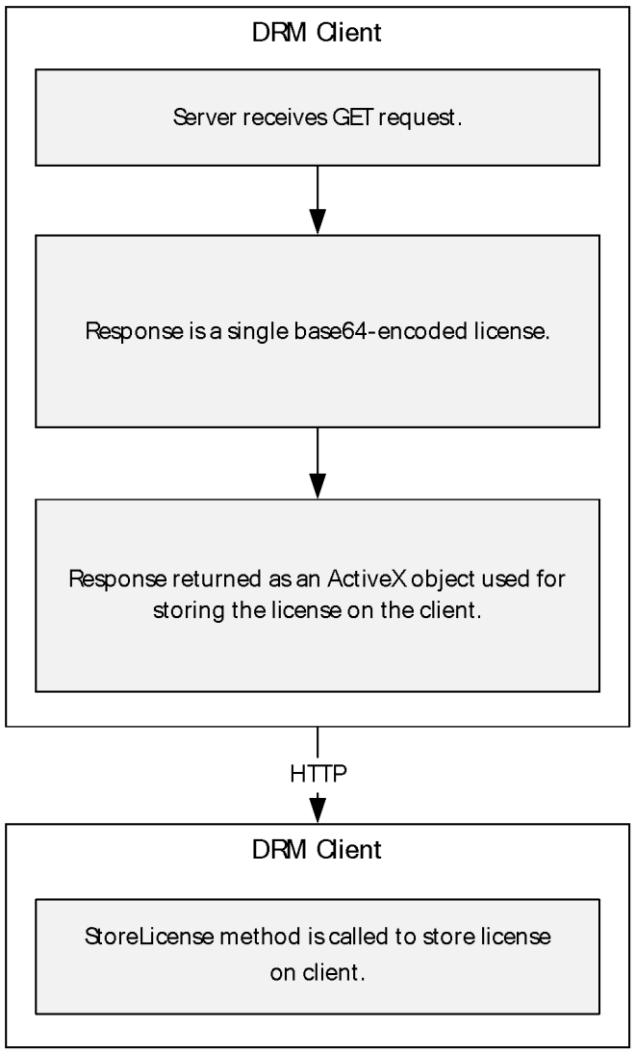


Figure 5: DRM client/server response sequence

The server response to the client request is a single license that is encoded with the base64 encoding algorithm embedded within an HTML page.

Request Validation

If the client request is longer than 201 bytes, the server MUST return the error "2007:Challenge String does not have correct length."

If the **Version** field is not equal to { 0x00, 0x01, 0x00, 0x01 }, the server MUST return the error "2008:License Request Version does not match the version supported."

If decryption of the client request fails, the server MUST return the error "2003:Unable to interpret the challenge blob. Probably incompatible client."

If the **AppSec** field contains an application security level lower than that required for the desired license, the server MUST return the error "2025:The requesting application security level is lower than specified level."

If the **pkcert.pk** is not found within the white list of nonrevoked client **certificates** (see Client Certificate White List (section 3.2.3.1.1)), the server MUST return the error "4004:Client version is not supported. Probably missing client verification key."

If signature verification of **pkcert.pk** fails, the server MUST return the error "2009:Unauthorized client request. Signature Verification Failure."

Response Generation

members

CERTIFIED_LICENSE.license.licVersion MUST contain the value { 0x00, 0x01, 0x00, 0x01 }.

CERTIFIED_LICENSE.license.datLen MUST contain the value { 0x75, 0x00, 0x00, 0x00 }.

CERTIFIED_LICENSE.license.sign contains the signature of **CERTIFIED_LICENSE.license.ld** as described in **LICENSE (section 2.2.2.3.4)**.

CERTIFIED_LICENSE.license.ld.KID contains the key ID for the issued license. This value MUST match the value requested by the client in the **KeyID** field.

CERTIFIED_LICENSE.license.ld.key contains the encrypted **RC4** content key as described in **LICENSEDATA (section 2.2.2.3.5)**.

CERTIFIED_LICENSE.license.ld.rights contains a bitmask of the rights granted in the license. This MUST NOT include more rights than were requested by the client in the **Rights** field.

CERTIFIED_LICENSE.license.ld.appSec contains either the minimum application security level required to use the license or the same value requested by the client in the **AppSec** field.

CERTIFIED_LICENSE.license.ld.expiryDate contains the license expiration date as described in **LICENSEDATA**.

CERTIFIED_LICENSE.cert1 contains the Microsoft-signed certificate representing the license server. This certificate is supplied in CS.

CERTIFIED_LICENSE.cert2 contains the root certificate representing the Microsoft certificate authority. This certificate is supplied in CS.

3.2.5.2 DRM Version 7 Server Message Processing Events and Sequencing Rules

The Digital Rights Management (DRM) version 7 License Response packet is used by the license server to send a license for protected media content to a client.

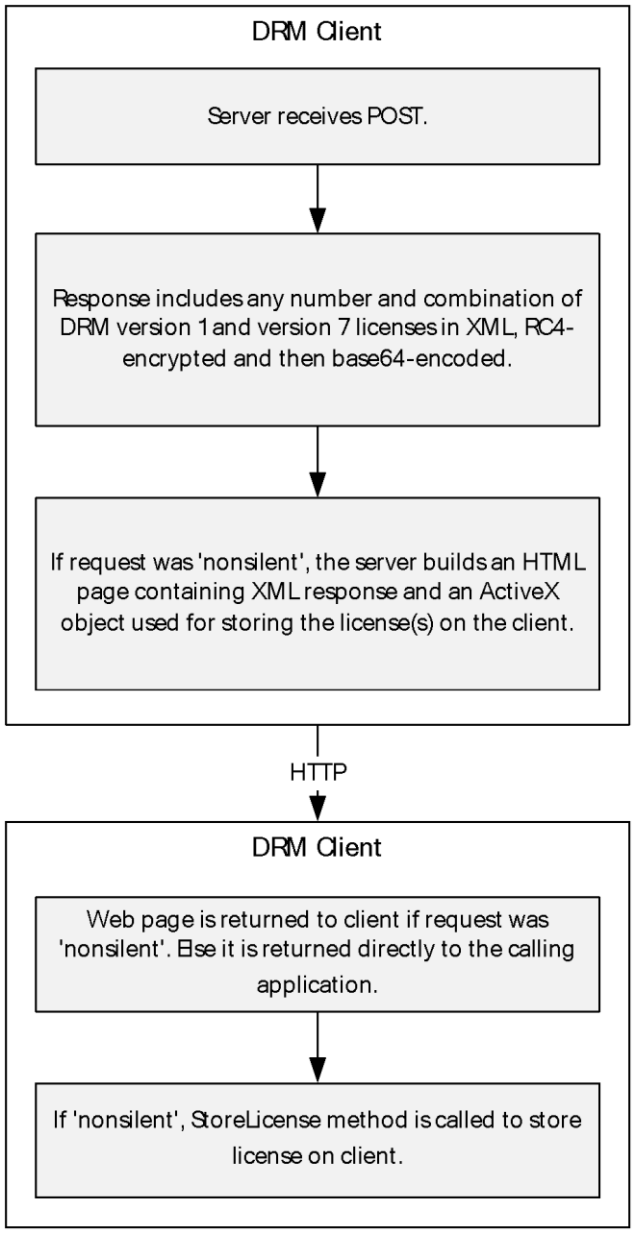


Figure 6: DRM client/server response sequence

If the client request was silent, the response is returned directly to the client. If the request was nonsilent, the response is returned to the client embedded within an HTML page. The HTML page is formatted as follows. The text enclosed in braces ("{{" and "}}") can be replaced as appropriate.

```
<HTML><HEAD><TITLE>{{optional page title}}</TITLE>
<Script Language="JavaScript">function StoreV2License(hr)
{ netobj.StoreLicense("{{license response}}"); } </Script></HEAD>
<BODY onLoad="StoreV2License()">{{optional descriptive text}}
<OBJECT classid=clsid:A9FC132B-096D-460B-B7D5-1DB0FAE0C062
height=0 id=netobj width=0 VIEWASTEXT>
<EMBED MAYSCRIPT TYPE="application/x-drm-v2" HIDDEN="true"></OBJECT>
{{optional descriptive text}}
</BODY></HTML>
```

The format of the response, which is in extensible markup language (XML), can include any number and combination of **DRM** version 1 and version 7 licenses, encoded with the base64 encoding algorithm.

Request Validation

Unless otherwise noted, most fields in the request are not strictly validated, because they are intended to be used within the implementer's business logic for license issuance. However, the server expects that the license request conforms to the schema.

c:/LICENSEREQUEST/V1CHALLENGE is validated according to the rules for validating a version 1 license request.

If **c:/LICENSEREQUEST/CLIENTINFO/CLIENTID** does not **base64**-decode to a 168-byte CLIENTID structure, the server MUST return DRM_E_LIC_CLIENDID_DECODING_FAILURE (0x8004800FL) to the client.

The server MUST validate CA against a white list of nonrevoked client **certificates** as described in Client Certificate White List (section 3.2.3.1.1). If the CA is not found within the white list or if signature validation of the client-signed data fails, the license server MUST return DRM_E_LIC_UNAUTHORIZED_DRM_CLIENT (0x8004800EL) to the client.

The server SHOULD map the following XML elements to associated state variables so that they are exposed to the higher layer for use:

c:/LICENSEREQUEST/CLIENTINFO/WRMHEADER -> WRMHeader

c:/LICENSEREQUEST/CLIENTINFO/SUBJECTID1 -> SubjectId1

c:/LICENSEREQUEST/CLIENTINFO/SUBJECTID2 -> SubjectId2

c:/LICENSEREQUEST/CLIENTINFO/DRMKVERSION -> DrmKVersion

c:/LICENSEREQUEST/CLIENTINFO/REVOCATIONINFO -> RevocationInfo

c:/LICENSEREQUEST/CLIENTINFO/SECURITYVERSION -> SecurityVersion

c:/LICENSEREQUEST/CLIENTINFO/CLIENTVERSION -> ClientVersion

c:/LICENSEREQUEST/CLIENTINFO/APPSEC -> ClientAppSec

After c:/LICENSEREQUEST/CLIENTINFO/CLIENTID is decoded, the result is mapped into a **CLIENTID** structure. **CLIENTID.EncRandNum** MUST be decrypted with KS and stored as internal state SessionKey. SessionKey is used to initialize an RC4 cipher, which is in turn used to decrypt **CLIENTID.pk** and is stored and made available to the higher layer as ClientCertificate.

Response Generation

The **XML** response is generated according to the schema described in XML Schema for Version 7 License Response (section 2.2.3.2.4).

The server's determination of how many licenses are generated and the format of each license generated is third-party server implementation-specific, although all state in the Abstract Data Model is provided to the higher layer if it makes a determination to use values to make deterministic decisions on the data used to populate licenses for the response.

After the higher layer has created all necessary licenses, it must call the **TransmitLicensesToClient** abstract interface (section 3.2.1.1) with a properly populated array of **LicenseToSend** structures (section **2.2.1.7**). Each element in the array is assigned the correct value according to license version (either 1 or 7) as well as the correct license size and license buffer.

The higher layer provides a variable number of already-generated licenses in either version 1 or version 7 as well as indicating which version of license they are.

For each version 1 license, the server includes the license in the response as described in DRM Version 1 License Format Within a Version 7 License Response (section 2.2.3.2.4.1).

For each version 7 license, the server includes the license in the response as described in DRM Version 7 License Format (section 2.2.3.2.4.2).

If, within the license request, the client sends a **CRL** version or REV_INFO version lower than that known to the server, then the server MUST send the latest known REV_INFO and CRL data to the client within the license response. The **REV_INFO.WMDRMRLVIHEAD.dwRIV** field contains the version of the REV_INFO data.

3.2.5.3 DRM Version 11 Server Message Processing Events and Sequencing Rules

The Digital Rights Management (DRM) Version 11 License Response packet is used by the license server to send a license for protected media content to a client.

The version 11 processing sequence is identical to the version 7 license response processing sequence.

Request Validation

A version 11 server MUST validate the request using the rules given for a version 7 server in DRM Version 7 Server Message Processing Events and Sequencing Rules (section 3.2.5.2).

/**c:LICENSEREQUEST/CLIENTINFO/MACHINECERTIFICATE** MUST be validated according to the following rules:

1. Validate /c:CertificateCollection
    1. If the /c:CertificateCollection element is missing or does not have a "c:Version", "xmlns:c" attribute, the server MUST return E_FAIL (0x80000008L).
    2. If the "c:Version" attribute value is not "2.0", the server MUST return DRM_E_INVALID_CERTCHAIN_VERSION (0xC0042945L).
    3. If the "xmlns:c" attribute value is not http://schemas.microsoft.com/DRM/2004/02/cert, the server MUST return DRM_E_INVALID_CERTCHAIN_NAMESPACE (0xC0042954L).
    4. There MUST be at least one child c:Certificate element. If not, the server MUST return E_FAIL (0x80000008L).
2. Validate /c:CertificateCollection/c:Certificate
    1. If the inner text of this element does not contain either "<c:Data" or "</c:Data>", the server MUST return DRM_E_MACHINE_CERT_DATATAG_MISSING (0xC0042961L).
3. Validate /c:CertificateCollection/c:Certificate/c:Data/c:PublicKey/KeyValue/RSAKeyValue/Exponent
    1. If this element does not exist, the server MUST return E_FAIL (0x80000008L).
    2. If the value of this element is not "AQAB", the server MUST return DRM_E_INVALID_EXPONENT (0xC0042959L).
4. Validate /c:CertificateCollection/c:Certificate/c:Data/c:PublicKey/KeyValue/RSAKeyValue/Modulus
    1. If this element does not exist, the server MUST return E_FAIL (0x80000008L).

2. If the value of this element is blank, the server MUST return DRM_E_INVALID_MODULUS (0xC0042953L).

5. Validate /c:CertificateCollection/c:Certificate/c:Data/c:SecurityLevel

    1. If this element does not exist, the server MUST return E_FAIL (0x80000008L).

6. Validate /c:CertificateCollection/c:Certificate/Signature/SignedInfo/CanonicalizationMethod

    1. If this element does not exist or does not have the attribute "Algorithm", the server MUST return E_FAIL (0x80000008L).

7. Validate /c:CertificateCollection/c:Certificate/Signature/SignedInfo/SignatureMethod

    1. If this element does not exist or does not have the attribute "Algorithm", the server MUST return E_FAIL (0x80000008L).

    2. If the value for the attribute "Algorithm" is not "http://www.w3.org/2000/09/xmldsig#rsa-sha1", then the server MUST return DRM_E_INVALID_SIGNATURE_METHOD_ALG (0xC0042949L).

8. Validate /c:CertificateCollection/c:Certificate/Signature/SignedInfo/Reference/DigestMethod

    1. If this element does not exist or does not have the attribute "Algorithm", the server MUST return E_FAIL (0x80000008L).

    2. If the value for the attribute "Algorithm" is not "http://www.w3.org/2000/09/xmldsig#sha1", then the server MUST return DRM_E_INVALID_DIGEST_ALG (0xC0042948L).

9. Validate /c:CertificateCollection/c:Certificate/Signature/SignedInfo/Reference/Transforms/Transform

    1. If there are not exactly two instances of this element, or if either instance does not have the attribute "Algorithm", the server MUST return E_FAIL (0x80000008L).

    2. If the value of the attribute "Algorithm" for the first instance is not "http://www.microsoft.com/DRM/CERT/v2/Data", then the server MUST return DRM_E_INVALID_TRANSFORM_ALG (0xC0042947L).

    3. If the value of the attribute "Algorithm" for the second instance is not "http://www.w3.org/TR/2001/REC-xml-c14n-20010315", then the server MUST return DRM_E_INVALID_TRANSFORM_ALG (0xC0042947L).

10. Validate /c:CertificaetCollection/c:Certificate/Signature/SignedInfo/Reference/DigestValue

    1. If this element does not exist, the server MUST return E_FAIL (0x80000008L).

    2. If the value of this element does not match the computed digest as described in section 2.2.4.1.1 <MACHINECERTIFICATE>, the server MUST return DRM_E_INVALID_DIGEST (0xC0042958L).

11. Validate /c:CertificateCollection/c:Certificate/Signature/SignatureValue

    1. If this element does not exist, the server MUST return E_FAIL (0x80000008L).

    2. If the value of this element does not match the computed signature as described in section 2.2.4.1.1 <MACHINECERTIFICATE>, the server MUST return E_FAIL (0x80000008L).

12. Validate /c:CertificateCollection/c:Certificate/Signature/SignedInfo

    1. If this element does not exist, the server MUST return E_FAIL (0x80000008L).

13. Validate /c:CertificateCollection/c:Certificate/c:Data/c:ManufacturerData

    1. If this element is present, c:ManufacturerName must be present. If it is not present, the server MUST return E_FAIL (0x80000008L).

Response Generation

The version 11 license response is generated in the same manner as the version 7 license response.

3.2.6 Timer Events

None.

3.2.7 Other Local Events

None.

4 Protocol Examples

4.1 DRM Version 1 License Request Example

The following example shows a version 1 license challenge.

```
http://www.contoso.com/license.asp?challenge=AAEAAeAIkaw4fVjvbsmN3lmgKi
ELAk040wTNFyAf65mzEKOwqNRPatHJByPumUTOy1g07toer!8agV0ux8qvBNnYQBOEgB1coMSZFHK7SRfOlV5
dco3f!wxYIhPHZ*kyUJQgzis1B8JGM9eyG4OuOofxV!*TcJQoUsLYDI*0L7pPrpMs0Ol0u2dHYNvMsPrXSN
mn5frVucfkOtIQPmHbTd1rP!i4GoGrYN*MmotuoOALTz!fXbyzkP7t&DRMVer=1.3
```

4.2 DRM Version 1 License Response Example

The following example shows a response webpage that contains an encoded license.

```
<HTML>
<HEAD>
    <TITLE>V1 Licensing</TITLE>
    <Script Language="VBScript">
        const LICENSE_EXPIRED           = &H80041000
        const LICENSE_INCONSISTENT      = &H80041001
        const LICENSE_INCORRECT_VERSION = &H80041003
        Sub Window_OnLoad()
            On Error Resume Next
            DrmStore.StoreLicense(
                "AAEAAHUAAAB3IFadTI8UJy3PzB9yilDoxgf5DRjqL4NXqFkns7*!
                Z4jFwCPX!oCDS1pPTHhMcmhaVStId0dMSlY4V3RhUT09AEeTvPQpG
                Nt!AJ5BE6tB4ZJ5tDQJo*bnTOnAxatFIYch72C8A04kdFz8ZK*!UT
                j52e4dIRkQkMBHXXnma4xe9KFZB3QypiOMM6LQFyPs0ViJGwAAAAA
                AAJYUBwMWAAEAADgAAADRQt0mNlnxj7as*ys3NSMJaaWViZC1Ppnl
                LxYqUdqCMm2iPIlzxu4zm5xxu39qj47qy33j5mXGbpviYTFldxMwN
                RRSckf6kyEdHDya3LyAc2NjDB8AAAAAAAAAAAAAAAAAAQAAOAAAAG
                N!793njE8kEVW*BhFk*W5xfYgP*ymWlfUQely7kQCMci!Q6wPkIhG
                9LfC2Z85Uf01UPGTZ7pNCns0OdMfy85CZ5ceKkC0KYaQK*OrdqAQN
                Y2MMHwAAAGMAAAABAAAAAQ==")
            if (err.number <> 0) then
                if (err.number = LICENSE_EXPIRED) then
                    StoreLicenseResult.innerHTML = "You just received a
                        license that is already expired. Maybe your
                        clock is set wrong. Check and try again."
                elseif (err.number = LICENSE_INCONSISTENT) then
                    StoreLicenseResult.innerHTML = "You just received a
                        corrupt license. Check with the license
                        server."
                else
                    StoreLicenseResult.innerHTML = "Error:" &
                        CStr(hex(err.number)) & ":" & err.Description
                end if
            end if
        End Sub
    </Script>
</HEAD>
<BODY>
    <p align="center"><B>V1 License</B></p>
    <OBJECT classid=CLSID:760C4B83-E211-11D2-BF3E-00805FBE84A6 id=DrmStore>
        <EMBED MAYSCRIPT TYPE="application/x-drm" HIDDEN="true"
            LICENSE="AAEAAHUAAAB3IFadTI8UJy3PzB9yilDoxgf5DR
            jqL4NXqFkns7*!Z4jFwCPX!oCDS1pPTHhMcmhaVStId0dMS
            lY4V3RhUT09AEeTvPQpGNt!AJ5BE6tB4ZJ5tDQJo*bnTOnA
            xatFIYch72C8A04kdFz8ZK*!UTj52e4dIRkQkMBHXXnma4x
            e9KFZB3QypiOMM6LQFyPs0ViJGwAAAAAAJYUBwMWAAEAAD
            gAAADRQt0mNlnxj7as*ys3NSMJaaWViZC1PpnlLxYqUdqCM
            m2iPIlzxu4zm5xxu39qj47qy33j5mXGbpviYTFldxMwNRRS
```

```
            ckf6kyEdHDya3LyAc2NjDB8AAAAAAAAAAAAAAAAAAQAAOAA
            AAGN!793njE8kEVW*BhFk*W5xfYgP*ymWlfUQely7kQCMci
            !Q6wPkIhG9LfC2Z85Uf01UPGTZ7pNCns0OdMfy85CZ5ceKk
            C0KYaQK*OrdqAQNY2MMHwAAAGMAAAABAAAAAQ==">
    </OBJECT>
    <span id="StoreLicenseResult">You have received a v1 license.</span>
</BODY>
</HTML>
```

4.3 DRM Version 7 License Request Example

The following example shows a sample WMDRM: License Protocol version 7 license request that contains all of the required elements. The fields might not contain complete data. For a complete description of an element, see the respective element topic.

```
<LICENSEREQUEST version="2.0.0.0">
  <V1CHALLENGE>

  <!--  DRMv1 challenge with empty KEYID string -->

  </V1CHALLENGE>
  <ACTIONLIST>

  <!--  Application is requesting the right to play the content. -->
  <!--  More than one right can be requested. -->

    <ACTION>Play</ACTION>
  </ACTIONLIST>

  <!-- Information about the client that is
       requesting the license. -->
  <CLIENTINFO>
    <CLIENTID>
    <!-- Client ID structure -->
    </CLIENTID>

    <!-- Version of the DRMv7 client -->
    <CLIENTVERSION>9.00.00.2778</CLIENTVERSION>

    <!-- Application security of client application -->
    <APPSECURITY>2000</APPSECURITY>

    <!-- Certificate subject ID of the component
         that is talking to DRM -->
    <SUBJECTID1>212</SUBJECTID1>

    <!-- Certificate subject ID of the component
         that is talking to the component-->
    <!-- talking to DRM. -->
    <SUBJECTID2>1107</SUBJECTID2>

    <!-- Version of Kernel Mode DRM on the client computer -->
    <DRMKVERSION>2.2.0.0</DRMKVERSION>
  </CLIENTINFO>


  <!-- WMRMHEADER section, verbatim from the header of the content -->

  <WMRMHEADER version="2.0.0.0">
    <DATA>

    </DATA>
    <SIGNATURE>

    </SIGNATURE>
```

```
  </WMRMHEADER>

</LICENSEREQUEST>
```

4.4 DRM Version 7 License Response Example

The following example shows a sample WMDRM: License Protocol version 7 license response that contains all of the required elements. The fields might not contain complete data. For a complete description of an element, see the respective element topic.

```
<LICENSERESPONSE>
  <LICENSE version="2.0.0.0">
    <!-- encrypted and base64 encoded XML license -->
  </LICENSE>
</LICENSERESPONSE>
```

4.5 DRM Version 7 Nonsilent License Response Example

The following example shows a response web page containing an encoded license.

```
<HTML>
<HEAD>
<TITLE></TITLE>
  <Script Language="JavaScript">

  function StoreV2License(hr)
  {
    netobj.StoreLicense("AAEAAHUAAADnIFW4Ec2j0JXEId5cfdhQoXCZJSPIjaKE
                         5L!FlSp0YM!7pSCayfFHVDBlTnRRS2tqa09GZENpcW54
                         akhnZz09AMnjjoZs5X9ZjuZCvFGDfSymhnp29w!0v0u9
                         t!NLeS5mw0I!iDNHqX0T5pZ0ie8HxJqQ23WRU1zOp*p8
                         OreBn3L1NzR2qaqJwSIP97XtS04mEwAAAAAAAJYUBQYQ
                         AAEAADgAAAAB2ZQI!btK6AOOJI68EEuHnnfVDsPjufRe
                         9FseC8IsW14EnD1HgjkJQ3*VKD9zKJB3oJQ9ZnbtJ10u
                         kgWxZtc5NwkIMU85AR8Aj6y0IcZpIxQFCBQAAG!JAAAA
                         AgAAqLkAAQAAOAAAADmkObY2USONnrxHr140bmTk!T9n
                         o7hB7EibVZl463LmVqQkywubKQ4l8RM!RwonN23ygvlh
                         efHw0BG2IAyFJ0GFxaThS1yagLYrZnxWSkc6FAYDAQAA
                         AAoAAAABAAAAAQ==");
  }

  </Script>
  </HEAD>
  <BODY onLoad="StoreV2License()">
  <OBJECT classid=clsid:A9FC132B-096D-460B-B7D5-1DB0FAE0C062
          height=0 id=netobj width=0 VIEWASTEXT>
  <EMBED MAYSCRIPT TYPE="application/x-drm-v2" HIDDEN="true">
  </OBJECT>

  </BODY>

</HTML>
```

4.6 DRM Version 11 License Request Example

The following example shows a sample WMDRM: License Protocol version 11 license request that contains all of the required elements. The fields might not contain complete data. For a complete description of an element, see the respective element topic.

```
<LICENSEREQUEST version="2.0.0.0">
  <V1CHALLENGE>

  <!--  DRMv1 challenge with empty KEYID string -->

  </V1CHALLENGE>
  <ACTIONLIST>

  <!--  Application is requesting the right to play the content. -->
  <!--  More than one right can be requested. -->

    <ACTION>Play</ACTION>
  </ACTIONLIST>

  <!-- Information about the client that is
       requesting the license. -->
  <CLIENTINFO>
    <MACHINECERTIFICATE>
    <!-- Base64 encoded XML certificate -->
    </MACHINECERTIFICATE>

    <REVINFO>
    <!-- Base64 encoded REV INFO structure -->
    </REVINFO>

    <CLIENTID>
    <!-- Client ID structure -->
    </CLIENTID>

    <!-- Version of the DRMv11 client -->
    <CLIENTVERSION>11.0.6002.18005</CLIENTVERSION>

    <SECURITYVERSION>2.5.0.0</SECURITYVERSION>

    <!-- Application security of client application -->
    <APPSECURITY>2000</APPSECURITY>

    <!-- Certificate subject ID of the component
         that is talking to DRM -->
    <SUBJECTID1>212</SUBJECTID1>

    <!-- Certificate subject ID of the component
         that is talking to the component-->
    <!-- talking to DRM. -->
    <SUBJECTID2>1107</SUBJECTID2>

  </CLIENTINFO>


  <!-- WMRMHEADER section, verbatim from the header of the content -->

  <WMRMHEADER version="2.0.0.0">
    <DATA>

    </DATA>
    <SIGNATURE>

    </SIGNATURE>
  </WMRMHEADER>

</LICENSEREQUEST>
```

4.7 DRM Version 11 License Response Example

The following example shows a sample WMDRM: License Protocol version 11 license response that contains all of the required elements. The fields might not contain complete data. For a complete description of an element, see the respective element topic.

```
<LICENSERESPONSE>
  <REVOCATION type="WMDRMNET">
    <!-- base64 encoded revocation list -->
  </REVOCATION>
  <REVOCATION type="DeviceRevocationList">
    <!-- base64 encoded revocation list -->
  </REVOCATION>
  <LICENSE version="2.0.0.0">
    <!-- encrypted and base64 encoded XML license -->
  </LICENSE>
</LICENSERESPONSE>
```

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 NT operating system
- Windows 2000 operating system
- Windows XP operating system
- Windows Server 2003 operating system
- Windows Vista operating system
- Windows Server 2008 operating system
- Windows 7 operating system
- Windows Server 2008 R2 operating system
- Windows 8 operating system
- Windows Server 2012 operating system
- Windows 8.1 operating system
- Windows Server 2012 R2 operating system
- Windows 10 v1511 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 1.7: Windows NT applies only to **DRM** version 1.

Windows 2000 and Windows Server 2003 apply only to DRM version 1 and version 7.

Windows XP, Windows Vista, and Windows 7 apply to DRM version 1, version 7, and version 11.

Windows Server 2008, Windows Server 2008 R2 operating system, Windows 8, Windows Server 2012, Windows 8.1, Windows Server 2012 R2, and Windows 10 v1511 apply to DRM version 1, version 7, and version 11, but only when acting as a client.

<2> Section 2.1: Transport mechanism is implemented in Windows 2000, Windows XP operating system Service Pack 2 (SP2), Windows Server 2003, Windows Vista, and Windows Server 2008. The WMDRM: License Protocol uses **HTTP** or **HTTPS** for data transfer, depending on the license acquisition URL specified in the WMRMHEADER object. The TCP ports are configurable by the implementer.

<3> Section 2.2.2.2: The normal sequence of operations that the web page script takes in order to store the license locally is as follows:

1. Create an IWMDRMProvider instance via WMDRMCreateProvider or WMDRMCreateProtectedProvider.

2. Create an IWMDRMLicenseManagement instance by calling IWMDRMProvider::CreateInstance, passing in the IID of IWMDRMLicenseManagement.

3. Call IWMDRMLicenseManagement::StoreLicense to store the license locally.

<4> Section 2.2.3.2.32: The value of the predefined script variable "app.count" is 2 if the client application uses the Windows Media Digital Rights Management (WMDRM) SDK or 1 if an application uses DRM directly.

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

N

O

P

R

S

T

V

W