Marlin – Core System Specification

Version 1.3.6
Final

Source	Marlin Developer Community
Date	March 13, 2013

Notice

THIS DOCUMENT IS PROVIDED "AS IS" WITH NO REPRESENTATION OR WARRANTY, EXPRESS OR IMPLIED, CONCERNING THE COMPLETENESS, ACCURACY, OR APPLICABILITY OF ANY INFORMATION CONTAINED IN THIS DOCUMENT. THE MARLIN DEVELOPER COMMUNITY ("MDC") ON BEHALF OF ITSELF AND ITS PARTICIPANTS (COLLECTIVELY, THE "PARTIES") DISCLAIM ALL LIABILITY OF ANY KIND WHATSOEVER, EXPRESS OR IMPLIED, ARISING OR RESULTING FROM THE RELIANCE OR USE BY ANY PARTY OF THIS DOCUMENT OR ANY INFORMATION CONTAINED HEREIN. THE PARTIES COLLECTIVELY AND INDIVIDUALLY MAKE NO REPRESENTATIONS CONCERNING THE APPLICABILITY OF ANY PATENT, COPYRIGHT (OTHER THAN THE COPYRIGHT TO THE DOCUMENT DESCRIBED BELOW) OR OTHER PROPRIETARY RIGHT OF THIS DOCUMENT OR ITS USE, AND THE RECEIPT OR ANY USE OF THIS DOCUMENT OR ITS CONTENTS DOES NOT IN ANY WAY CREATE BY IMPLICATION, ESTOPPEL OR OTHERWISE, ANY LICENSE OR RIGHT TO OR UNDER ANY PATENT, COPYRIGHT, TRADEMARK OR TRADE SECRET RIGHTS WHICH ARE OR MAY BE ASSOCIATED WITH THE IDEAS, TECHNIQUES, CONCEPTS OR EXPRESSIONS CONTAINED HEREIN.

Use of this document is subject to the agreement executed between you and the Parties, if any.

Any copyright notices shall not be removed, varied, or denigrated in any manner.

Intellectual Property

A commercial implementation of this specification requires a license from the Marlin Trust Management Organization.

Contact Information

Feedback on this specification should be addressed to:
editor@marlin-community.com

Contact information for the Marlin Trust Management Organization can be found at: http://www.marlin-trust.com/

Contents

1 Introduction

1.1 Document Organization

This document covers the complete Marlin Core Specifications. It is organized as follows:

- (this) introduction, including abbreviations, definitions and references
- An overview of the Marlin core system architecture
- Sections for each of the normative specification elements. These are:
  - DRM Objects
  - DRM Roles and Services
  - Client-Service Role Protocols
  - Protocol Bindings
  - Key Management
  - Renewability
  - Trust Management
  - Content File Formats (see [MFF1.0])
  - Usage Rules
  - Naming Conventions (e.g. names that must be used uniformly across implementation to achieve the interoperability goals of marlin
- A set of attachments:
  - WSDLS and XML Schemas

1.2 Conformance Conventions

The key words “MUST”, “MUST NOT”, “REQUIRED”, “SHALL”, “SHALL NOT”, “SHOULD”, “SHOULD NOT”, “RECOMMENDED”, “MAY”, and “OPTIONAL” in this specification are to be interpreted as described in IETF RFC 2119 [RFC2119].

These capitalized key words are used to unambiguously specify requirements and behavior that affect the interoperability and security of implementations. When these key words are not capitalized they are meant in their natural-language sense.

All Elements of this specification are considered Normative unless specifically marked Informative. All Normative Elements are Mandatory to implement, except where such an element is specifically marked OPTIONAL. Finally, where Normative elements are described as OPTIONAL, they MAY be omitted from an implementation, but when implemented, they MUST be implemented as described.

1.3 Namespaces and Identifiers

This specification defines schemas conforming to XML Schemas [Schema] and normative text to describe the syntax and semantics of XML-encoded objects and protocol messages. In cases of disagreement between the schema documents and the schema listings in this specification the schema documents take precedence. Note that is some cases the normative text of this specification imposes constraints beyond those indicated by the schema documents.

1.3.1 Namespaces and Notation

The following table summarizes the normative schemas defined by this specification and their XML namespace [XMLns] URIs. These URIs MUST be used by implementations of this specification:

Prefix	XML Namespace	Schema File Name	Description
mc:	urn:marlin:core:1-3:schemas	marline-core.xsd	Marlin core schema
ml:	http://marlin-drm.com/1.0	Marlin.xsd	Marlin DRM schema
mncs:	urn:marlin:core:1-1:nemo:services:schemas	marlin-nemo-core-services.xsd	Marlin NEMO Services
exc:	urn:marlin:core:1-2:nemo:services:schemas:exceptions	marlin-nemo-core-exceptions.xsd	Fault Response schema

Table 1-1 Namespace Definitions

In addition to the schemas defined by this specification, we leverage existing schemas to achieve our design goals. The following table summarizes the external schemas used in this specification:

Prefix	XML Namespace	Description
oct:	http://www.octopus-drm.com/profiles/base/1.0	Octopus
pk:	http://www.octopus-drm.com/profiles/base/1.0/plankton	[8pus] §4
sf:	http://marlin-drm.com/starfish/1.2	[Starfish]
nemoc:	http://nemo.intertrust.com/2005/10/core	[NEMO] §2
nemosec:	http://nemo.intertrust.com/2005/10/security	[NEMO] §3
xs:	http://www.w3.org/2001/XMLSchema	[Schema]
xsi:	http://www.w3.org/2001/XMLSchema-instance	[Schema]
ds:	http://www.w3.org/2000/09/xmldsig#	[xmldsig]
xenc:	http://www.w3.org/2001/04/xmlenc#	[xmlenc]
wsdl:	http://schemas.xmlsoap.org/wsdl/	[WSDL]
wsa:	http://www.w3.org/2005/08/addressing	[WS-Addr]
wsx:	http://schemas.xmlsoap.org/ws/2004/09/mex	[WS-MEX]
wsnt:	http://docs.oasis-open.org/wsn/b-2	[WS-BASENOTE]
wsrf-rp:	http://docs.oasis-open.org/wsrf/rp-1	[WSRF-RP]
wsrf-rl:	http://docs.oasis-open.org/wsrf/rl-1	[WSRF-RL]
wsrf-bf:	http://docs.oasis-open.org/wsrf/bf-2	[WSRF-BF]
S11:	http://schemas.xmlsoap.org/soap/envelope	[SOAP11]
wsu:	http://docs.oasis-open.org/wss/2004/01/oasis-200401-wss-wssecurity-utility-1.0.xsd	[WS-SEC]
wsse:	http://docs.oasis-open.org/wss/2004/01/oasis-200401-wss-wssecurity-secext-1.0.xsd	[WS-SEC]
wsp:	http://schemas.xmlsoap.org/ws/2004/09/policy	[WS-POL]

Table 1-2 Supporting Namespaces

As a convention throughout this document we use the namespace prefixes described above to qualify XML elements and attributes which are specified elsewhere. That is the typographical convention is: <MarlinElement>, <ns:ForeignElement>, XMLAttribute, **Datatype**, OtherKeyword.

1.3.2 Names and Identifiers

This section complements naming patterns and names introduced in subsequent sections of this specification. Additional naming conventions are described in Section 12.5.

This specification uses Uniform Resource Identifiers [RFC2396] to identify various entities including resources, algorithms, policies, attributes and other application specific objects. Implementations of this specification MUST support URIs of at least 256 bytes long.

A standard naming pattern is specified to simplify management of the identifiers defined across all systems that leverage the underlying technology (e.g., Nemo and Octopus), as well as across all the implementations of Marlin. The naming pattern used allows each layer/entity to delegate management of namespaces to subordinate layers/entities. This approach guarantees uniqueness of names generated by those subordinate layers/entities without active coordination between them.

All Marlin names SHALL be specified as URIs and use a consistent root (prefix) based on the syntax of the identifier. Two URIs and their respective prefixes are defined in the table below:

Identifier Type	Prefix
URN	urn:marlin:
URL	http://marlin-drm.com

The identifiers defined within the above declared namespaces are normative to this specification. Per [URNMARLIN], URNs SHALL be treated in a case sensitive manner. The namespace prefix SHALL be the lowercase string “urn:marlin:”.

Note that this specification uses identifiers defined by other specifications and their normative definition solely comes from the specification that defines them. As a convenience to the reader these identifiers may appear in examples or in giving guidance. However, such uses are informative and do not supercede the identifier definition.

Device manufacturers, service providers, and perhaps other organizations that implement Marlin specifications may need to define Marlin related object names specific to themselves or their extensions to the Marlin specifications. To keep organization specific object names distinct without requiring active coordination between all such entities, organization specific urns are used. Such organizations can register their own unique URN namespace (e.g. urn:SomeOrganization) and define such names using a URN registered to their organization. In order to not require all such organizations register their own URN namespace, they can be registered with Marlin and issued a

unique URN namespace within the Marlin namespace. Thus there are two patterns for organization specific identifiers.

Identifier Type	Prefix
URN	urn:marlin:organization:<organization-name>
URN	urn:<organizations-own-namespace>

In addition to naming patterns, specific attributes names and values SHALL be defined when these are global in scope (e.g. the value used for a device capability or the role of a system entity.) In general attribute names are qualified by a namespace. In some instances an attribute namespace is defined by a URL and for purposes of convenience we define an URN equivalent to maintain consistency and simplify processing. The following table gives the set of URLs and their URN equivalent.

URL Attribute Namespace	URN Equivalent
http://nemo.intertrust.com/2004/attribute	urn:marlin:nemo:2004:attribute

Table 1-3 Attribute Namespace Equivalences

To minimize backward compatibility issues we adopt the policy of not putting version information in the attribute namespace qualifiers. Going forward, we will accommodate behavioral changes of an entity for which an attribute refers by defining a new attribute name.

1.3.3 Marlin Naming

In the following sections we specify naming conventions which will aid implementations and delivery system specifications in defining identifiers in a consistent and interoperable manner.
The general syntax can be described using the following BNF grammer;

urn:marlin:<specid>:<objattrid>:<attribute specific data>

where;

<specid> is a constant identifier for the particular specification
<objattrid> is the object or attribute identifier for which the URN applies

The <specid> identifier SHALL adopt the prefix identifiers in the following table.

Specification Identifiers	Description
core	Prefix identifier defined by this specification.
broadband	Prefix identifier defined in the Broadband Delivery System specification.
broadcast	Prefix identifier defined in the Broadcast Delivery System specification.
omav2gw	Prefix identifier defined in the OMA Gateway specification.
<other>	Future identifiers

Table 1-4 Specification Identifiers

The value space of URN identifiers may also need to be standardized so that independent implementations by the same organization do not collide. Therefore it is recommended that URN values use the specification id prefix followed by the organization identifier.
The general syntax can be described using the following BNF grammer;

urn:marlin:organization:<orgid>:*

where;

<orgid>	is the organization specific identifier. Note this identifier can include sub-organization identifiers which are managed by the organization itself.
*	any valid URN namespace specific string

Table 1-5 Organization Identifiers

1.4 Abbreviations

AES	Advanced Encryption Standard
AES-CTR-128	AES Counter mode with 128-bit key
BF	Broadcast Flag
BKB	Broadcast Key Block
CAS	Conditional Access System
CBC	Cipher Block Chaining
CCI	Copy Control Information
CPRM	Content Protection for Recordable Media
CTR	Counter
DCF	DRM Content Format
DCS	Data Certification Service
DLNA	Digital Living Network Alliance
DUS	Data Update Service
HBES	Hierarchical Hash-Chain Broadcast Encryption Scheme
ISO	International Organization for Standardization
MAC	Message Authentication Code
MDCF	Marlin DRM Content Format
MDD	Marlin Device Domain
MG-R	MagicGate Type-R
NEMO	Networked Environment for Media Orchestration
OMA	Open Mobile Alliance
PAT	Program Association Table
PMT	Program Map Table
PSI	Program Specific Information
SAML	Security Assertions Markup Language
SDP	Security Data Provider
SI	Service Information
SOAP	Simple Object Access Protocol
SPTS	Single Program Transport Stream
SSDP	Simple Service Discovery Protocol
TCP	Transmission Control Protocol
TS	Transport Stream
TTS	Timed Transport Stream

UPnP	Universal Plug and Play
VCPS	Video Content Protection System
WSDL	Web Services Description Language
XML	Extensible Markup Language

347

348 ## *1.5 Terms and Definitions*

Usage information	Information that indicates usage rules and governance applied for protected content.
Protection information	Protection-related information along with protected content.
Marlin Device Domain	Set of devices that have established a membership relationship which is reflected by an Octopus link.
Renewability	The process by which security related elements of Marlin Core System implementations can be renewed if necessary
Security Metadata	Metadata necessary for managing the security and trustworthiness of the Marlin System, such as Certificate Revocation Lists
Roles	A combination of client/service functions supported by an implementation
Marlin DRM Client	A Role with the necessary functionality to allow it to render content
Domain Manager	A Role with the necessary functionality to allow it to link devices into domains
Marlin Import Function	A Role with the necessary functionality to allow it to generate licenses and package content
Marlin Device	The most basic Role any implementation must support
Marlin User	The representation of a user
Marlin Subscription Node	The representation of the token that makes it possible to bind many content licenses together
Marlin Delivery System Specifications	Specifications that define how Marlin Content and Licenses are created and how such content is imported into a Marlin User's Device Domain
Marlin User	The Octopus Node that represents the user who "owns" the content
Marlin Content	Media packaged into a Marlin File Format Container
Marlin Content Format	The File Format and Media Codec Profile that Marlin Content is contained in.
Marlin Domain Information Provider	A Role with the necessary functionality to advertise which domains DRM Client can register to and the DRM Object Provider
Content is targeted to a Node	License uses the Node in isHostReachable
Content is bound to a Node	License uses that Node for SCUBA key management purposes.
Content Key	The symmetric key that encrypts the payload of the content

Starfish	The Marlin broadcast encryption scheme based on HBES (Hierarchical Hash-Chain Broadcast Encryption Scheme)
Harpoon	A protocol used to measure the network proximity of 2 devices

349

350 ## *1.6 References*

351 ### 1.6.1 Normative References

352

[8pus]	Octopus DRM Technology Platform Specifications, Version 1.0
[AES]	NIST FIPS 197: Advanced Encryption Standard (AES). November 2001. http://csrc.nist.gov/publications/fips/fips197/fips-197.pdf
[AES-MODES]	Recommendation of Block Cipher Modes of Operation. NIST. NIST Special Publication 800-38A. http://csrc.nist.gov/CryptoToolkit/modes/800-38_Series_Publications/SP800-38A.pdf
[CPRM-DVD]	Content Protection for Recordable Media Specification, DVD Book, Revision 0.96, http://www.4centity.com/data/tech/spec/cprm-dvd096.pdf
[CPRM-SD-AUDIO]	Content Protection for Recordable Media Specification, SD Memory Card Book, SD-Audio Part, Revision 0.96, http://www.4centity.com/data/tech/spec/Cprm-sd-audio096.pdf
[CPRM-SD-VIDEO]	Content Protection for Recordable Media Specification, SD Memory Card Book, SD-Video Part, Revision 0.93, http://www.4centity.com/data/tech/spec/Cprm-sd-video-part-093-1.pdf
[DTCP14]	Digital Transmission Content Protection Specification Revision 1.4 Informational Version
[hmacwithsha1]	H. Krawczyk, M. Bellare, and R. Canetti. *HMAC: Keyed-Hashing for Message Authentication*. IETF RFC 2104. February 1997. http://www.ietf.org/rfc/rfc2104.txt
[hmacwithsha256]	D. Eastlake and T. Hansen. *US Secure Hash Algorithms (SHA and HMAC-SHA)*. IETF RFC 4634. July 2007. http://www.ietf.org/rfc/rfc4634.txt
[ISMACryp]	"ISMA Encryption and Authentication Specification", version 1.0, February 2004.
[MEXP]	Marlin - Export Parameter Specification
[MFF1.0]	Marlin – File Formats Specification version 1.0. December 2005.
[MIME]	N. Freed & N. Borenstein. *Multipurpose Internet Mail Extensions (MIME) Part One: Format of Internet Message Bodies*. IETF RFC 2045. November 1996. http://www.ietf.org/rfc/rfc2045.txt

[MOCS1.0]	Marlin – Output Control Specification version 1.0. June 2007
[NEMO]	NEMO Technology Platform Specifications, Version 1.1
[OBEX13]	IrDA Object Exchange Protocol (IrOBEX), Version 1.3, January 2003, http://www.IrDA.org/
[PKIX]	R. Housley, W. Ford, W. Polk, D. Solo. *Internet X.509 Public Key Infrastructure Certificate and CRL Profile*. IETF RFC 3280. April 2002. http://www.ietf.org/rfc/rfc3280.txt
[PKIXALGS]	R. Housley, B. Kaliski, J. Schaad. *Additional Algorithms and Identifiers for RSA Cryptography for use in the Internet X.509 Public Key Infrastructure Certificate and Certificate Revocation List (CRL) Profile*. IETF RFC 4055. June 2005. http://www.ietf.org/rfc/rfc4055.txt.
[QNAMEIDS]	Using Qualified Names (QNames) as Identifiers in XML Content, http://www.w3.org/2001/tag/doc/qnameids
[RFC2119]	S. Bradner, *Key words for use in RFCs to Indicate Requirement Levels*, IETF RFC 2119, March 1997. http://www.ietf.org/rfc/rfc2119.txt.
[RFC2396]	T. Berners-Lee, R. Fielding, L. Masinter. *Uniform Resource Identifiers (URI): Generic Syntax*. IETF RFC 2396. August 1998. http://www.ietf.org/rfc/rfc2396.txt
[RFC2585]	R. Housley, P. Hoffman. *Internet X.509 Public Key Infrastructure Operational Protocols: FTP and HTTP*. IETF RFC 2585. May 1999. http://www.ietf.org/rfc/rfc2585.txt
[RFC2616]	R. Fielding, J. Gettys, J. Mogul, H. Frystyk, L. Masinter, P. Leach, T. Berners-Lee, eds. *Hypertext Transfer Protocol – HTTP/1.1*. IETF RFC 2616 http://www.ietf.org/rfc/rfc2616.txt
[RFC2630]	Cryptographic Message Syntax. Network Working Group. R. Housley, Request for Comments: 2630. June 1999. http://www.ietf.org/rfc/rfc2630.txt
[RFC3279]	W. Polk, R. Housley, L. Bassham. *Algorithms and Identifiers for the Internet X.509 Public Key Infrastructure Certificate and Certificate Revocation List (CRL) Profile*. IETF RFC 3279. http://www.ietf.org/rfc/rfc3279.txt
[RFC4051]	D. Eastlake 3rd. *Additional XML Security Uniform Resource Identifiers (URIs)*. IETF RFC4051. April 2005. http://www.ietf.org/rfc/rfc4051.txt
[RSA-1_5]	B. Kaliski, J. Staddon. *PKCS #1: RSA Cryptography Specifications Version 2.0.* IETF RFC2437. October 1998. http://www.ietf.org/rfc/rfc2437.txt

[SAML1.1]	Eve Maler, Prateek Mishra and Rob Philpott, eds., *Assertions and Protocol for the OASIS Security Assertion Markup Language (SAML) V1.1,* http://www.oasis-open.org/committees/download.php/3405/oasis-sstc-saml-bindings-1.1.pdf
[Schema]	XML Schema Part 1: Structures. W3C Recommendation. D. Beech, M. Maloney, N. Mendelsohn, H. Thompson. May 2001. http://www.w3.org/TR/2001/REC-xmlschema-1-20010502/ XML Schema Part 2: Datatypes W3C Recommendation. P. Biron, A. Malhotra. May 2001. http://www.w3.org/TR/2001/REC-xmlschema-2-20010502/
[SHA1]	FIPS PUB 180-1. *Secure Hash Standard.* U.S. Department of Commerce/National Institute of Standards and Technology. http://www.itl.nist.gov/fipspubs/fip180-1.htm
[SHA256]	FIPS PUB 180-2. *Secure Hash Standard.* U.S. Department of Commerce/National Institute of Standards and Technology. http://csrc.nist.gov/publications/fips/fips180-2/fips180-2.pdf
[SOAP11]	"Simple Object Access Protocol (SOAP) 1.1," Box, Don, Ehnebuske, David , Kakivaya, Gopal, Layman, Andrew, Mendelsohn, Noah, Nielsen, Henrik Frystyk, Winer, Dave, eds. World Wide Web Consortium W3C Note (08 May 2000). http://www.w3.org/TR/2000/NOTE-SOAP-20000508/
[Starfish]	*Starfish - Marlin Broadcast Encryption Scheme* v1.2
[UPnPDevArch1.0]]	UPnP Device Architecture v1.0 http://upnp.org/resources/documents/CleanUPnPDA101-20031202s.pdf
[UPnPDevArch1.0.1]	UPnP Device Architecture v1.0.1 Draft http://www.upnp.org/resources/documents/CleanUPnPDA101-20031202s.pdf
[UPnPBasicDev]	Basic:1.0 Device Definition Version 1.0 http://upnp.org/standardizeddcps/documents/BasicDevice-1.0.pdf
[UPnPImplGuid]	UPnP Vendors Implementation Guide http://www.upnp.org/download/UPnP_Vendor_Implementation_Guide_Jan2001.htm
[URL]	T. Berners-Lee, L. Masinter, and M. McCahill. *Uniform Resource Locators (URL).* IETF RFC 1738. December 1994. http://www.ietf.org/rfc/rfc1738.txt

[URN]	R. Moats.. *URN Syntax.* IETF RFC 2141. May 1997. http://www.ietf.org/rfc/rfc2141.txt L. Daigle, D. van Gulik, R. Iannella, P. Falstrom.. *URN Namespace Definition Mechanisms.* IETF RFC 2611. June 1999. http://www.ietf.org/rfc/rfc2611.txt
[URNMARLIN]	*A Uniform Resource Name (URN) Namespace for the Marlin Development Community L.L.C. IETF* Draft 02, June 27, 2007
[VCPS]	Video Content Protection System for the DVD+R/+RW, Video Recording Format System Description, Version 1.2, February 2005 http://www.licensing.philips.com/vcps/
[WS-Addr]	*Web Services Addressing 1.0 - Core,* W3C Candidate Recommendation, 17 August 2005, http://www.w3.org/TR/2005/CR-ws-addr-core-20050817 Web Services Addressing 1.0 - SOAP Binding, W3C Candidate Recommendation, 17 August 2005, http://www.w3.org/TR/2005/CR-ws-addr-soap-20050817
[WS-BASENOTE]	*Web Services Base Notification 1.3,* Committee Specification, http://docs.oasis-open.org/wsn/wsn-ws_base_notification-1.3-spec-cs-01.pdf
[WSRF-BF]	*Web Services Base Faults 1.2,* OASIS Standard. http://docs.oasis-open.org/wsrf/wsrf-ws_base_faults-1.2-spec-os.pdf
[WSDL]	*Web Services Description Language* http://www.w3.org/TR/wsdl
[WS-MEX]	Web Services Metadata Exchange, September 2004 http://xml.coverpages.org/WS-MetadataExchange200409.pdf
[WSRF-RP]	*Web Services Resource Properties 1.2,* OASIS, Committee Draft 01, 18 May 2005 http://docs.oasis-open.org/wsrf/wsrf-ws_resource_properties-1.2-spec-cd-01.pdf
[WSRF-RL]	*Web Services Resource Lifetime 1.2,* OASIS, Committee Draft 01, 19 May 2005, http://docs.oasis-open.org/wsrf/wsrf-ws_resource_lifetime-1.2-spec-cd-01.pdf
[WS-SEC]	Web Services Security (WS-Security), Version 1.0, OASIS, April 5, 2002. http://docs.oasis-open.org/wss/2004/01/oasis-200401-wss-soap-message-security-1.0.pdf
[WS-SECX509]	Phillip Hallam-Baker *et al.,* eds., *Web Services Security X.509 Certificate Token Profile,* OASIS Standard 200401, March 2004, http://docs.oasis-open.org/wss/2004/01/oasis-200401-wss-x509-token-profile-1.0.pdf

[WS-SECX509-ER]	Phillip Hallam-Baker *et al.,* eds., *Web Services Security X.509 Certificate Token Profile,* Errata 1.0, December 2005, http://www.oasis-open.org/committees/download.php/16796/oasis-200512x509-token-profile-1.0-errata-005.pdf
[WS-SECSAML]	Phillip Hallam-Baker *et al.,* eds., *Web Services Security: SAML Token Profile,* OASIS Standard, December 2004, http://docs.oasis-open.org/wss/oasis-wss-saml-token-profile-1.0.pdf
[WS-TOPICS]	*Web Services Topics 1.2,* OASIS, Working Draft 01, 22 July 2004
[X509]	*ITU-T Recommendation X.509* (1997 E): *Information Technology - Open Systems Interconnection - The Directory: Authentication Framework,* June 1997.
[XML-1-1]	*Extensible Markup Language (XML) 1.1 (Second Edition),* http://www.w3.org/TR/xml11/
[xmldsig]	*XML-Signature Syntax and Processing W3C Recommendation* http://www.w3.org/TR/xmldsig-core/
[xmlenc]	*XML Encryption Syntax and Processing.W3C Recommendation* http://www.w3.org/TR/xmlenc-core/
[xml-exc-c14n]	*Exclusive XML Canonicalization, Version 1.0, W3C Recommendation 18 July 2002* http://www.w3.org/TR/xml-exc-c14n/
[XMLns]	Namespaces in XML. W3C Recommendation. T. Bray, D. Hollander, A. Layman. January 1999. http://www.w3.org/TR/1999/REC-xml-names-19990114

353

354 ### 1.6.2 Informative References

355

[Coral]	Coral Consortium Corporation http://www.coral-interop.org
[3GPP]	3rd Generation Partnership Project, “Transparent end-to-end packet switched streaming service (PSS); 3GPP file format (3GP) (Release6)”, 3GPP TS 26.244 V6.2.0, 2004-12.
[ISDB]	ARIB STANDARD, “SERVICE INFORMATION FOR DIGITAL BROADCASTING SYSTEM” ARIB STD-B10, version 3.8. (http://www.dibeg.org/aribstd/STD-B10-v3.8e.pdf).
[ItruObjectId]	Intertrust X.500 Object Identifier Specification
[MGRSVRMS-Info]	MagicGate Type-R for Secure Video Recording for Memory Stick PRO Specification – Informational Version – ver1.03
[MGRSARMS-Info]	MagicGate Type-R for Secure Audio Recording for Memory Stick and Memory Stick PRO Specification – Informational Version – ver0.9

[PKCS7]	RSA Laboratories. *PKCS #7: Cryptographic Message Syntax Standard.* Version 1.5, November 1993.
[PKCS8]	RSA Laboratories. *PKCS #8: Private-Key Information Syntax Standard*. Version 1.2, November 1993.
[RFC3852]	R. Housley, *Cryptographic Message Syntax (CMS).* IETF RFC 3852. July 2004. http://www.ietf.org/rfc/rfc3852.txt
[WS-POL]	Web Services Security Policy Language (WS-SecurityPolicy), Version 1.0, December 18, 2002

356

2 Marlin Core System Overview (Informative)

Marlin defines a Digital Rights Management [DRM] system for consumer-friendly content consumption models that allow consumers to enjoy appropriately licensed content transparently on compliant and capable devices.

The Marlin functional model (see Figure 2-1) allows content from different content delivery channels to be governed using Marlin DRM. The model also enables content to be made available on a set of compliant devices comprising an affiliation of the user's devices. We commonly refer to this affiliation of devices as a Marlin Domain or "consumer domain" or even more succinctly as simply a "domain."

Marlin DRM ensures that Marlin Content is handled according to the original content owner's intents. Marlin specifies how content is made available to devices in a consumer domain (Marlin Import Function), how Marlin Licenses express the rights to access such content (Marlin Licenses), the protocols by which Marlin devices communicate so that Content can be transferred between devices in the domain (or exported from it), and the (file) formats for the content (Marlin Content Format.) This level of specificity is necessary so that content can not only be transferred onto a Marlin-compliant device but also consumed on all devices in the Domain.

Note that in this version of the specification, the Marlin Delivery System Specifications – not the Marlin Core System Specifications - specify the Marlin Domains, as well as how content is made available to the devices in a Domain, how Marlin Licenses are generated from original content licenses or content owner intents, and how content is packaged into a file container (Marlin Content) that can be consumed by other Marlin Devices. In summary:

- The core specifications mandate only what is necessary to enable devices to obtain and play content governed by a basic set of usage rules, become part of or leave a domain, and obtain any necessary security metadata and DRM objects to achieve the above.
- The delivery system specifications extend the core specifications, to specify how different type of domains are formed and managed, how content is made available on standalone devices or devices part of a domain, the usage rules that govern access to content on devices in the domain and the movement of content and licenses between devices. They enable an end to end DRM system implementation for a given content delivery mechanism.
- The core specifications may be updated from time to time to include specifications required to support a delivery system specification that was defined later. Domain definitions that are independent of the delivery mechanisms are an example of such an update.

Marlin Interoperability

Marlin-compliant devices from different manufacturers should be interoperable – once content has been imported into a Marlin Device Domain, it should be possible to play it

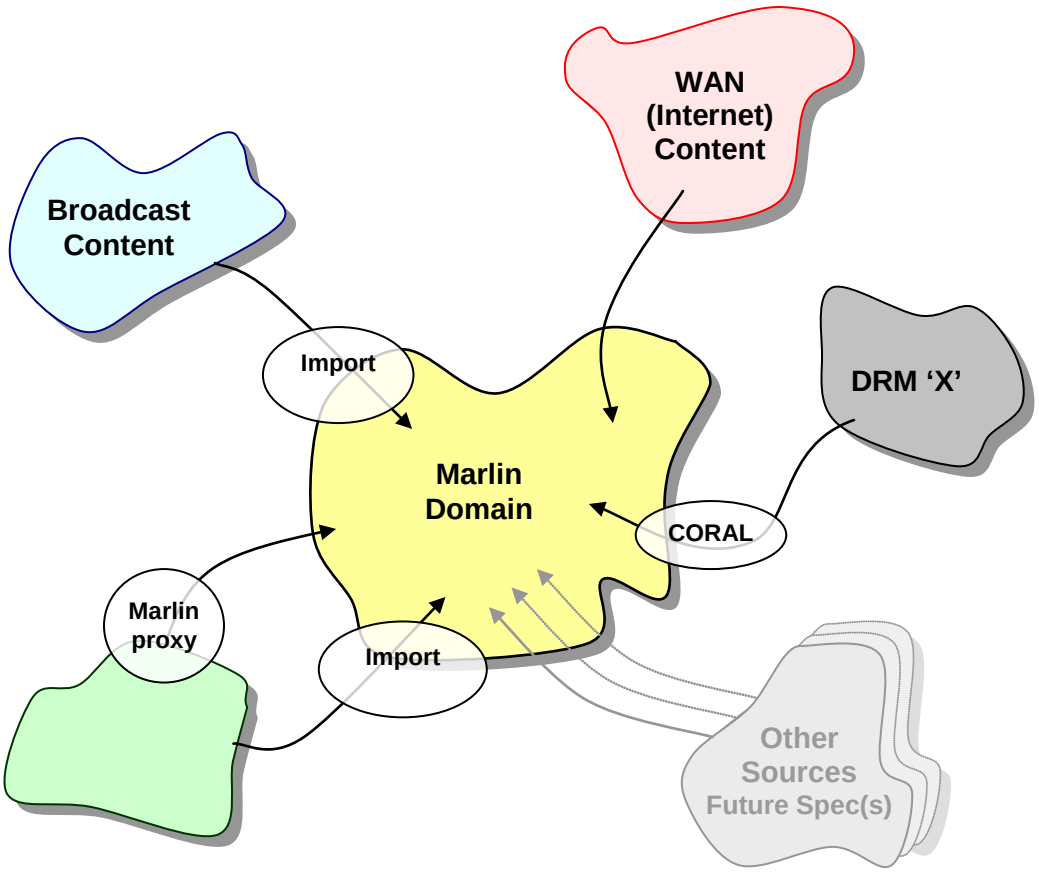


Figure 2-1: Marlin Functional Model

on any of the Domain's devices as long as the device's implementation of the core system supports the consumption of that type of content. E.g.

- Any Marlin video-capable Marlin DRM Client should be able to play any Marlin formatted video content as long as the codec profiles match. That is, incompatibilities should not reside at the container, content protection, or "minimum" license expression, functional or protocol level

Note the use of the "minimum" qualifier above: It is expected that specialized DRM Client features may be necessary in order to support certain market-driven features. In order to allow implementers to develop light-weight, conformant clients, some features of the core specifications will be considered optional to be implemented, while those necessary to ensure minimum interoperability will be mandatory.

In summary, the Marlin Core Specification defines the DRM Objects, the Marlin Content Formats, and the set of Marlin Device-side functions and protocols that all Marlin Core System conformant implementations support and that any implementation of a Marlin Delivery System Specification can rely on.

2.1 Scope of the Marlin Core System Specifications

Marlin Core System Specifications are defined as those specifications that, when implemented:

- Enable consumption of content imported via one of the Marlin Import functions
- Enable participation in the domains referenced by the licenses of such imported content, using local protocols (broadband connectivity is not mandated for devices implementing the Marlin Core System specification)

Note that it may be necessary for a proxy application to implement a given Marlin Delivery System mechanism to acquire the DRM objects representing participation in a domain and acquire the necessary security metadata tokens – and other devices in the domain should be able to use this proxy instead of implementing the Marlin Delivery System specifications themselves. An example of this would be a Marlin compliant, USB connected-only device that acquires content, licenses and metadata from a host gateway to a broadband service.

2.2 Marlin Core System Entities

Marlin Entities are the Marlin objects and roles (client or service functions) that realize the Marlin functional model.

Marlin Entities are to a large extent specified using Octopus and Nemo technology (not re-defined in this document). The necessary extensions of Octopus Nodes, Link and License objects and of Nemo Nodes needed for Marlin are specified in detail in the later sections of this document.

Marlin clients or services are hosted by a Nemo Node which binds the client or service to a Marlin certified identity for authentication purpose, and provides it the keys necessary for message confidentially and integrity. Marlin further groups client and service functionalities into Marlin Roles. The Roles are certified by a Marlin Certificate Authority and required for establishing trust between clients and services.

Marlin Nemo Nodes should at most only host one Marlin Role of each type.

The Marlin Entities not already defined by Octopus or Nemo specifications are defined in this section. Note that for ease of expression, “Octopus <type> Node” is equivalent to “Octopus Node of Type <type>”.

Marlin Content:

- Marlin Content is any media that has been encoded, encrypted and packaged into a file container according to the specifications in this document. It is expected that any Marlin DRM Client with the appropriate media rendering capabilities can consume any Marlin Content, as long as the license permits it to do so.

Marlin User (Node & Link objects):

- Marlin Content is generally, although not always, associated with a user. A user is a principal that holds the rights to the content. A user in Marlin is called a Marlin User and is represented by an Octopus User Node. Marlin Devices may be linked to User Nodes

Marlin Domain (Nodes & Link objects) :

- A Marlin Domain Node represents an entity to which Marlin Devices may be linked to form a collection of devices which make up a domain. A Marlin Domain is represented by an Octopus Domain Node.

Marlin Subscription (Node, Link, License objects):

- A Marlin Subscription Node is an entity used for grouping access to multiple Marlin Content items under a single token. A Subscription Node is represented by an Octopus Subscription Node. A Subscription License will typically include a condition that a given Subscription Node be reachable from the Octopus Personality node of the DRM Client attempting to access the content. The access to many Marlin Content items can thus be governed by the availability of a Subscription Link to a Subscription Node, and controls on this link may govern the availability of the entire group of content items whose licenses reference this Subscription Node.

Marlin Device (role):

- A Marlin Device defines the minimum role implemented by any physical device that conforms to the Marlin Core Specifications. It offers limited functionality and is used primarily for Discovery and Identification. Usually a physical device will implement additional roles, such as those described below.

Marlin Domain Manager (role):

- A Domain Manager performs such functions as managing the initial formation of a Marlin Devices Domain, allowing Marlin devices to register and remain in the Domain according to the Domain Policy, and administering the departure of Marlin Devices from the Domain. A Domain Manager hosts an Octopus Domain Node, or a Marlin User Node if the domain is defined as linking Octopus Personality Nodes to User Nodes. It implements any and all functions needed to support Marlin Devices including Domain registration and de-registration. It usually is accompanied by a Marlin DRM Object Provider and a Marlin Domain Information Provider.

Marlin Domain Information Provider (role):

- A Marlin Domain Information Provider is able to give information about Marlin domains (ID and associated policy) it knows about and where DRM Clients can register to these domains.

Marlin DRM Client (role):

- A Marlin DRM Client is the entity authorized to evaluate Octopus Licenses and release content keys to a DRM application for the uses indicated in Octopus License. A Marlin DRM Client is represented by an Octopus Personality Node. It hosts an Octopus Plankton DRM engine and other functions necessary to execute the Octopus Controls included in the Octopus License or other objects

such as Octopus Links and Octopus Agents. It implements any and all functions needed to evaluate licenses, execute the obligations of the licenses, acquire DRM and Security Metadata Objects, and participate in Domain registration and de-registration.

Marlin Import function (role):

- A Marlin Import function may package non-Marlin Content into Marlin Content. It generates an Octopus License that encodes the rights that the content owner intends. The license should govern the content as it is accessed on devices.

Marlin Security Data Provider (role):

- A Marlin Security Data Provider makes Security Metadata Objects (assertions, CRLs, etc…) available to other entities, primarily Marlin DRM Clients. The Security Data Provider is analogous to a Data Update Service which supplies current security metadata objects (see Section 9.3.3.)

Marlin DRM Object Provider (role):

- A Marlin DRM Object Provider makes DRM Objects (links and agents primarily) available to other entities, primarily Marlin DRM Clients. A DRM Object Provider also enforces any conditions required to make these objects available, such as proximity checks if applicable

Marlin Certification Authorities

- Marlin Certification Authorities issue certificates to services that directly or indirectly provision other entities with Octopus Nodes, Links and Licenses, Nemo Nodes, and Role or Security Metadata Assertions. They also issue keys and their certificates needed by individual entities to verify the chain of trust and validate signed objects.

Implementations will most likely group specific sets of roles with a single Nemo node, as for example in the following:

- A typical device enabled for playing content would host a single Nemo node that groups the following roles
  - Marlin Device
  - Marlin DRM Client (which includes the Octopus Personality Node part of any entity that evaluates content licenses)
- A device that implements a domain manager would host a single Nemo node that groups the following roles
  - Marlin Device
  - Marlin Domain Manager (as defined in a Delivery System specification)
  - Security Data Provider
  - Marlin DRM Object Provider
- A device that acts as a proxy to a Domain Manager would host a single Nemo node and could group the following roles (note that such as device would not be provisioned with an Octopus Node)
  - Marlin Device
  - Marlin Domain Information Provider
  - Marlin DRM Object Provider

2.3 Marlin Domains

This section illustrates the concept of Marlin Device Domains. Domain specifications have direct implications on specifications for protocols and DRM objects, so while this section is informative, this specification may not support other types of domains than those illustrated here.

In the simplest terms, a Marlin Domain represents a collection of Marlin DRM Clients that a private consumer considers their own and on which they expect to be allowed to enjoy legally acquired digital content.

By necessity, this collection is a compromise between the desire of the customer and the wants of the content and service industry that originate and distribute the content. Since no unified definition of this compromise is currently available, this collection itself can have several definitions depending on the type of content and the service provider.

In order to establish and manage these collections of devices while respecting the terms of the consumer-content industry compromise, Marlin defines Domain Managers, Domain Policies and related data objects and functions that Marlin compliant devices may optionally implement but must conform to.

- A Domain Manager (usually a software entity, defined above) manages the registration/deregistraton of Marlin Devices according to a Domain Policy
- A Domain Policy expresses the rules for forming, registering and deregistering devices to and from domains. It also defines what is permissible once a device has left a domain. Domain Policies are enforced by the Domain Manager. While DRM Clients may participate in the enforcement of the policy (e.g. participating in a proximity control check), interpretation of the policies of the domains DRM Clients are linked to is not required for the DRM Clients themselves.

A domain is implemented by establishing and managing Nodes for Users and Domains, and links between Marlin DRM Clients and Marlin User Nodes or Domain Nodes. Presently, the policies that govern these are defined in the Delivery System Specifications. The definition of the resulting nodes and links are part of the core specification.

Marlin Domain Topologies

A Marlin Domain Topology describes how Marlin Users, DRM Clients and Domain Nodes are associated in a domain.

Topologies are described in terms of:

- DRM Clients – represented by an Octopus Personality Node that is the root of the DRM engine that consumes content.
- Domain Managers – represented by an Octopus Domain or User Node that represents the domain they administer.
- Marlin Users – represented by an Octopus User Node
- Octopus Links – that express relationships between Octopus Nodes of DRM Clients, User Nodes, and Domain Nodes according to a desired domain topology

The Marlin Core Specifications define Octopus Object extensions that can be used by domain managers to encode in nodes, links and licenses the necessary domain-related metadata and controls for demonstrating active membership in a domain. They also define protocols by which the objects can be obtained, and the metadata queried.

A certain subset of these domain–related metadata and controls is mandatory to implement.

Marlin Domain Policies

The rules that govern whether a DRM Client can register with a Domain, and what the consequences are when a device deregisters from a domain, are implemented by the Domain Manager and the DRM Client Device in accordance with the Domain Policy.

Note that different types of contents may be processed differently in a same domain, according to the usage rules that govern the content.

This version of the specifications defines DRM Objects and functions that can support a set of Domain Policies Elements such as those defined below.

Marlin Domain Policy Elements - Examples

Domain Policies can be defined in terms of the following elements

- Domain Policy Identifier
- Registering Policy (per device type if differentiated)
    - Number of domains of this type a device can register
    - Max number of devices of a certain type that can be linked to a domain node
    - Max age of link before self-deactivation if not renewed
    - Proximity check required for registering & acceptable RT delay.

- Deregistration Policy - one of (per device type if differentiated)
    - The deregistration policy uses Marlin Core System Protocols and Octopus Objects in order to implement the intent of the policy, such as whether the content that used to be accessible on the device prior to the deregistration action should
        - still play after the deregistration, or
        - no longer play after the deregistration

2.4 Content Binding and Movement under Marlin Governance

Marlin Content is usually bound to an Octopus User Node, Domain Node, or Personality Node and targeted to an Octopus User Node, Subscription Node or Personality Node

When the content is bound to a Node, the key that encrypts the content key is one of the Scuba Sharing keys of that Node. To decrypt the content key, the DRM engine will look for a set of links with scuba extensions, from the Octopus Personality Node of the DRM client that attempts to access the content key, to the Node that the content is bound to.

When the content is targeted to a Node, the control in the license will have a condition that checks whether that Node is reachable. When the control is executed, the DRM engine will look for a set of links (with or without scuba extensions) from the Octopus Personality Node of the DRM client that attempts to access the content key, to the Node that the content is targeted to

Note: If the content is bound to another node than it is targeted to, it is good practice to also include a condition of node reachability for the node that the content is bound to. This will allow the license evaluation to fail should the content be bound to a node that is not reachable from the Octopus node of the DRM engine that attempts to access the content.

Combining Binding and Targeting offers a variety of content handling options:

- Binding and Targeting content to a (User or Domain) node linked to a set of Marlin DRM clients (with scuba extensions on the links) allows the content to be equally accessed on all the DRM clients (all other things such as device capabilities and controls on the links being equal)
- Binding and/or Targeting the content to the Octopus Personality Node of a Marlin DRM Client node prevents this content from being accessed on any other device than that one

Marlin defines move/copy protocols and actions that allow for the following:

- The license of content bound and/or targeted to the Octopus Personality Node of a source DRM client can be “copied” or “moved” to the Octopus Personality Node of a sink DRM Client. The conditions on this copy or move differ, depending on whether or not the source and sink are linked to the same Octopus User or Domain Node. For example:
    - If the source and sink are linked to the same node, the action is a simple re-binding of the license
    - If the source and sink are not linked to the same node, or not linked to any node, the action requires a proximity check between source and sink

  The difference between move and copy is whether or not access to the content on the source device is maintained after the completion of the transaction. Move disables access on the source device, copy does not. All other terms of the license remain in effect.

  Whether copy or move or both are allowed is expressed in the license of the content, and therefore determined entirely by the import function of the domain the content is imported into.

Export from Marlin to Other DRM systems

Export is an operation by which Marlin contents are transferred to other DRM systems. Whether the export operation is permitted is specified by usage rules in a license. Also specified must be the usage rules to apply to the content in the target DRM.

In export operations, Marlin export functionality does not specify the means of transformation of license or contents for other DRM systems.

For basic export mode, the original content (license) remains valid at the source Marlin DRM Client after the export operations.

3 Marlin DRM Objects

Marlin DRM Objects are Octopus objects ([8pus] §2) with Marlin-specific attributes extensions, and encoding.

Note: Certain Octopus Object Attributes or Extensions defined in this section are optional. Unless expressly mentioned otherwise, when they are present in the Object, they MUST be made available to the DRM engine for reference in control programs.

3.1 Octopus Objects

3.1.1 Node and Link Objects

As explained in [8pus] §2, Octopus Nodes represent entities that content can be "bound to" in the DRM system (e.g. a device, a device group or a user). The Octopus node representing a "license evaluating" entity (e.g. a device) is represented by a Personality Node.

Links are directed assertions between two Nodes. For example, a link from a device node to a user node may indicate that the device belongs to the user. Links can be thought of as "license enablers" in the sense that if a license requires user Node X to be reachable, then the condition would be met if the device has a link from its personality node to user Node X.

3.1.2 License Objects

As explained in [8pus] §2, the Octopus license objects are the following:

- ContentKey: represents the key, CK. CK is the key which encrypts the content. The ContentKey object MAY contain one or more expressions of the encrypted CK.
- Protector: associates one or more Content objects and a ContentKey object. There is no need for a secure association here as the ContentKey object is needed to decrypt the content.
- Control: represents how the content key, and therefore the content, is governed. A Control object embeds [Plankton] bytecode that is executed according to a protocol (see [8pus] §3) that returns a result for a particular Action query (e.g. PLAY). Note that the Controls can be used in a different context than a License: they can be used as "Agents" in order to execute plankton bytecode on a DRM Client in order to initialize a counter for example.
- Controller: associates securely one or more ContentKey objects and a Control object.

3.1.3 Lookup Scope for Spawned Controls

When a control program uses the System.Host.SpawnVm system call [8pus] §4.7.2.10 the 'ModuleId' parameter identifies a Control object to load. In this specification, the following lookup rule to locate the Control object with the specified ID SHALL be used:

- Look in the same Bundle in which the control program issuing the system call is located.

Since Control objects have globally unique IDs, the lookup search order is not relevant, because in the case where more than one representation of the same Control object with the specified ID exist, they all represent the same Control object.

3.1.4 Agent Conveyance for License Transfer

The Agent for Transfer action (see Section 5.7.6) included in a Control object MUST be carried as an External Extension. The External Extension MUST have the following properties:

- The External Extension is placed as the direct child element of the corresponding License Bundle.
- The External Extension has the subject attribute which value has the same value as the uid attribute of a Control object which includes the corresponding Transfer action.

The Control object which includes the Agent and carried in External Extension MUST have PKI signature. The External Extension MUST contain one Bundle element. The Control object and the PKI signature MUST be placed as the direct child elements of that Bundle element. The PKI signature MUST have the following properties:

- The PKI signature has only one Reference which refers to the Control object for the Agent.
- The signer of the PKI signature is same as the signer of the Controller object associated with the Control object including the corresponding Transfer action.

If a license contains an Agent which uses the System.Host.SpawnVm system call to load other Control objects, such Control objects SHALL be placed in the same Bundle element in which the Agent issuing the system call is placed. Therefore, during License Transfer, all the elements in the Bundle element containing the Agent SHALL be included in the Agent Carrier.

Note that HMAC and PKI signature for the Controller objects SHALL NOT have the Reference for the External Extension nor the Control object for the Agent.

3.2 Octopus Object Attributes and Extensions

3.2.1 Octopus Nodes

Marlin defines various types of Octopus nodes with different attributes and extensions. A DRM Client is NOT REQUIRED to check whether the required attributes appear in the Octopus Object but it MAY return an error if a required attribute is not in the object.

As introduced in Section 1.3.2, attribute and extension identifiers must begin with the appropriate prefix. The following table defines the attribute prefix, types of nodes, their values and the REQUIRED extensions. Unspecified attribute types have a default type of string.

urn:marlin:core:node:attribute:type	
Node Type	**Attribute Value(s)**
Personality	personality
Domain	domain
User	user

Subscription	subscription
Other	*

Table 3-1 Octopus Node Types

The Scuba key requirements for the above Octopus Node types are outlined below. When present these keys SHALL be carried in Octopus Node object extensions as defined in [MCSV13] §3.3.3.1 and [8pus] §6.2.2. Server implementations MAY provision and process Scuba keys for the Octopus Node types described below. Client implementations SHALL support the Octopus Node and Octopus Link types it acquires through the client implemented protocols. Client implementations SHALL be able to derive Scuba Sharing keys carried in these acquired Octopus Links as defined in [8pus] §6.5.

- Personality nodes SHALL include a Public/Private Scuba Sharing key pair
- Personality nodes SHALL include a Secret Scuba Sharing key
- Personality nodes SHALL include a HBES Device Keyset [Starfish]
- Personality nodes MAY include a Public/Private Scuba Confidentiality key pair
- User nodes SHALL include a Public/Private Scuba Sharing key pair and/or a Secret Scuba Sharing key
- User nodes MAY include a Public/Private Scuba Confidentiality key pair
- Subscription nodes MAY include a Public/Private Scuba Sharing key pair and/or a Secret Scuba Sharing key
- Domain nodes SHALL include a Secret Scuba Sharing key
- Domain nodes MAY include a Public/Private Scuba Sharing key pair

Note that services distribute and use Scuba keys based on the business models they implement. Octopus Nodes, Links and Scuba keys are generic technologies, when implemented accordingly, enable services and clients to interoperate in a semantic-free manner.

Whenever an Octopus node contains one or more scuba public keys, the format of these public key SHALL be the DER encoding of the ASN.1 structure SubjectPublicKeyInfo defined in [RFC3280] §4.1. The identifier for this format is “SPKI”.

In addition to the attributies which identify the node to be of a particular type, the following attributes SHALL also be specified for certain node types:

Node Type	Attribute Identifier	Attribute Type	Attribute Value(s)
Personality	urn:marlin:core:node:attribute:device-class	string	Dedicated Device
			Personal Computer
			Portable Device
	urn:marlin:core:node:attribute:device-features	array	To be defined
Domain	urn:marlin:core:node:attribute:domain-policy-type	string	To be defined in each delivery system specification

828 **Table 3-2 Node Attributes Values**

829 The attribute values defined in Table 3-2 are terms which enable classification of the
830 node at a very coarse granularity. The following table provides guidance on the
831 semantics of these terms.
832

Attribute Value	Description
Dedicated Device	A node in this class implies that it hosts a dedicated application which, in general, permanently resides in a home setting.
Personal Computer	A node in this class implies that it hosts the DRM application on an open application platform which, in general, permanently resides in a home setting.
Portable Device	A node in this class implies that it hosts the DRM application on a dedicated platform which, in general, is not necessarily permanently resident in a home setting.

833 **Table 3-3 urn:marlin:core:node:attribute:device-class Attribute Value Descriptions**

834 The urn:marlin:core:node:attribute:device-features attribute is an array of attributes (see
835 [8pus] §2.2.1.2). This OPTIONAL attribute is used as a container for describing the
836 permanent features of a device[1].

837 ### 3.2.1.1 Octopus Personality Nodes

838 The following data elements SHALL be included in the private part of an Octopus
839 personality node.
840

Data	Description
Data for Secret Key of Scuba Sharing Key	
Key Id	URN for the scuba sharing secret key
Key Data	Key data of the secret key
Data for Private Key of Scuba Sharing Key	
Key Pair Id	URN for the scuba sharing public/private key pair
Key Data	Key data of the private key
Data for Device Key Set [Starfish]	
Device Id	Device Id defined in [Starfish] §3.2.2
Key Tree Id	URN for the HBES Key Tree.
Algorithm Id	Algorithm Identifier for the Starfish. It MUST be set to http://marlin-drm.com/starfish/algorithmID/1.0
Device Key Set	Device Key Set defined in [Starfish] §3.3.1

841 **Table 3-4 Data elements in the private part of an Octopus personality node**

[1] This attribute is intended to provide additional qualifiers to further distinguish the capabilities of a device. Values will be defined in the future.

842 Refer to Section 3.3.3.1.1 for a description of the XML encoding of an Octopus
843 personality node.

844 3.2.2 Octopus Links

845 Links MAY have Scuba extensions that carry the encrypted private and secret scuba
846 sharing keys of the node “to” with the public or secret sharing key of the node “from”.
847 Links from personality to domain nodes MAY have several attributes:
848

urn:marlin:link:attribute	
Attribute Identifier	**Description**
urn:marlin:link:attribute:domain-serial-number	Domain serial number of the link
urn:marlin:link:attribute:link-type	Type of link
urn:marlin:link:attribute:domain-policy	Domain Policy
urn:marlin:link:attribute:domain-id	Domain ID Context

849 **Table 3-5 Link Attributes**

850 • domain-serial-number: This OPTIONAL attribute carries the domain serial
851 number of the link. The presence of this attribute depends on whether the policy
852 of the domain requires it.
853 • link-type: This OPTIONAL attribute type of link. An example of value would be
854 device-to-domain.
855 • domain-policy: This attribute specifies the policy ID identifying the domain policy.
856 It is REQUIRED for all domain related links.
857 • domain-id: This attribute indicates the domain context for which this link is apart
858 of. It is REQUIRED for all domain related links.[2]
859
860 If the Link includes a Control Object, the attributes of the Link Object SHALL be visible to
861 the Plankton Virtual Machine running this Control under the following container:
862

/Octopus/Link/Attributes/

863
864 External Extensions SHALL have uid attributes that follow the URN definitions and
865 conventions defined in Section 1.3.3.

866 3.2.3 License objects

867 When the Control is run, all the attributes of the Control and Controller Objects SHALL
868 be visible to the Plankton Virtual Machine under the following containers (as specified in
869 [8pus] §3):
870

/Octopus/Control/Attributes
/Octopus/Controller/Attributes

871

[2] A domain-id attribute MUST be a globally unique value. The representation MAY be a URI and it may be best implemented as a random value encoded in the URI. A UUID would also be a reasonable URI representation.

3.2.4 License Object Contexts

The purpose of license object contexts is to allow an implementation to delay the loading of certain persistent objects, such as links, until it encounters a license that indicates that those objects are needed for a successful evaluation of the license.

3.2.4.1 Contexts IDs

A Context ID is simply a unique ID that is used to tag objects, in order to indicate that these objects are associated with a certain context.

3.2.4.2 Context Tag in Objects

For Link and Node objects, the context ID, if any exists, SHALL be carried as the value of an object attribute named ‘ContextTag’.

3.2.4.3 Contexts in Bundles

When content licenses are created, and stored as a set of objects, signatures and other elements in a Bundle, the license issuer MAY include one or more context IDs. This signals to the host application that is processing the bundle that it SHOULD load in the runtime environment all persistent objects that have a context tag equal to one of the context IDs present in the Bundle. If the persistent objects with matching tags had already been loaded, no action is required of the application or device. The application or device MAY unload or cleanup those objects when it is done processing the content related to that Bundle.

3.3 XML Encoding of Octopus Objects

3.3.1 Overview

Marlin defines its own XML schema. This schema imports the Octopus XML schema and adds elements specific to Marlin (for example, the revocation extension.)
The encoding of Octopus objects in XML MUST be valid instances of the schema elements defined by this specification.

3.3.2 General Schema Design

The XML representation of all the Octopus objects is based on the complex type <oct:OctopusObjectType>. Thus, all the Octopus objects SHALL support attributes and extensions. The type of each Octopus Object element is derived from this base type. These types may aggregate other elements such as the <oct:SecretKey> element for the <oct:ContentKeyType> for instance.

The key distribution system keys ([8pus] §6) SHALL be described in the terms of an extension: the <oct:ScubaKeys> element SHALL then be a child of the <oct:Extension> element. The same SHALL be true for revocation keys with the <oct:Torpedo> revocation extension defined in Marlin schema.

As explained in [8pus] §2, there are different kinds of Octopus Objects (ContentKey, Protector, Controller, Control, Node and Link). These objects can be bundled together as well as Extensions using the <oct:Bundle> element. If objects or extensions are signed

within the <oct:Bundle>, the <oct:Bundle> SHALL contain <ds:Signature> elements defined in [xmldsig] and in accordance with the profile in Section 3.3.4.

In the case where a license uses a revocation extension such as broadcast encryption (a DRM Client has been revoked), the <sf:BroadcastKeyblock> SHALL be carried inside the <oct:Bundle> (see Section 7.4).

In the case where the license creator wants to advertise that this license needs a certain context to be loaded (which maps to a set of Octopus Links), the <oct:Bundle> MAY also embed a <ml:ContextList> element carrying all the needed <ml:Context> URIs.

3.3.3 Additional Constraints on the Schema

3.3.3.1 Nodes

Nodes contain keys (in Extensions such as ScubaKeys.) It should be possible to separate the public information of the Node (the id, attributes and public keys) and its private extensions (that will carry the secret and private keys). Moreover, there should be one signature per part (the public and the private) so that the public node with its signature can be exported as is (as a parameter of the request to a registration service for example.)

According to the taxonomy of [8pus] §2, the private extensions must be carried in an ExternalExtension and signed. Thus for the XML representation of the public Node and its private Extensions MAY be packaged in the same <oct:Bundle> element or MAY arrive separately.

Whenever an Octopus node contains one or more scuba public keys, the attributes of the KeyData element carrying these keys SHALL be:

- encoding="base64"
- format="X509SPKI"

3.3.3.1.1 Personality Nodes

Personality Nodes may include confidential information. When the data elements concerning a Secret Key and/or a Private Key of a Scuba Sharing Key are encoded in an XML format, then the schema defined by the namespace *http://www.octopus-drm.com/profiles/base/1.0* may be used for the encoding.
The following is an example of an XML fragment of the private part of an Octopus personality node within a Scuba extension. Note that the namespace xmlns:oct="http://www.octopus-drm.com/profiles/base/1.0" is assumed for this example.

```
[01]  <oct:ScubaKeys>
[02]    <oct:SecretKey
          uid="urn:marlin:organization:phony:octopus-personality:01:secret-sharing"
          usage="KeySharing">
[03]      <oct:KeyData encoding="base64"
                       format="RAW">Z8Zpc1H…=</oct:KeyData>
[04]    </oct:SecretKey>
[05]    <oct:PrivateKey
            uid="urn:marlin:organization:phony:octopus-personality:01:private-sharing"
```

```
            pair="urn:marlin:organization: phony:octopus-personality:01:pair-sharing"
            usage="KeySharing">
[06]      <oct:KeyData encoding="base64"
                        format="PKCS#8">MIIC…=</oct:KeyData>
[07]    </oct:PrivateKey>
[08]  </oct:ScubaKeys>
```

Line [02], in the above example, uid attribute indicates Key Id in Table 3-4.

Line [03], in the above example, the <oct:KeyData> element carries the base64 encoded value of the secret key.

Line [05], in the above example, pair attribute indicates Key Pair Id in Table 3-4.

Line [06], in the above example, the <oct:KeyData> element carries the base64 encoded value of the private key in the format specified by [PKCS8].

When data elements concerning a Device Key Set [Starfish] are encoded in an XML format, the schema defined by the *http://marlin-drm.com/starfish/1.2* namespace MAY be used for the encoding.
The following is an example of an XML fragment of the Torpedo extension of the private part of an Octopus personality node. Note that we assume the *xmlns:ml="http://marlin-drm.com/1.0"* attribute has been defined in this example.

```
[01]  <ml:Torpedo>
[02]    <ml:BroadcastKey uid="0F2A130003000002"
[03]              source=" urn:marlin:starfish:keytree:1">
[04]      <ml:BroadcastKeyMethod
[05]              Algorithm="http://marlin-drm.com/starfish/algorithmID/1.0"/>
[06]      <oct:KeyData encoding="base64"
[07]               format="RAW">Z8Zpc1H…=</oct:KeyData>
[08]    </ml:BroadcastKey>
[09]  </ml:Torpedo>
```

Line [02], in the above example, uid attribute indicates ASCII hex representation of Device ID, and source attribute indicates the name of the Key Tree Id in Table 3-4.

Line [04], in the above example, algorithm attribute indicates Algorithm Id in Table 3-4.

On Line [06], in the above example, the <oct:KeyData> element carries the base64 encoded value of the Device Key Set in Table 3-4.

3.3.3.1.2 Attributes

Each XML encoding of a Node object SHALL carry an <oct:AttributeList> with the <oct:Attribute>(s) as defined in Section 3.2.1.
Example:

```
<oct:AttributeList xmlns="http://www.octopus-drm.com/profiles/base/1.0">
  <oct:Attribute name="urn:marlin:core:node:attribute:type">user</oct:Attribute>
```

```
</oct:AttributeList>
```

3.3.3.1.3 Extensions

All the public keys SHALL be carried inside the <oct:Node> element (in an <oct:Extension> element in the <oct:ExtensionList>). Other keys SHALL be carried in a separate <oct:Extension> element outside of the <oct:Node> element.

Marlin SHALL follow the Octopus Objects specification ([8pus] §2). The <oct:ScubaKeys> extensions SHALL be signed in the <oct:Node>. This means that the internal <oct:Extension> carrying <oct:ScubaKeys> inside the <oct:Node> (public keys) SHALL include a <ds:DigestMethod> element as well as a <ds:DigestValue> element. These elements MUST follow the guidance given in Section 3.3.4.

3.3.3.2 Links

The <oct:LinkTo> and <oct:LinkFrom> elements SHALL only carry a <oct:Uid> element (i.e. no <oct:Digest> element).

The <oct:Control> element is optional.

3.3.3.2.1 Attributes

Link attributes defined in Section 3.2.2. MUST be encoded in XML following the Octopus schema and packaged in an <oct:AttributeList>.

Example:

```
<oct:AttributeList xmlns=" http://www.octopus-drm.com/profiles/base/1.0">
  <oct:Attribute name="urn:marlin:link:attribute:domain-id">
    urn:marlin:domain:29490343</oct:Attribute>
  <oct:Attribute name="urn:marlin:link:attribute:domain-policy">
    urn:marlin:broadcast:domain-policy:organization:phony:fuse:policy:0
  </oct:Attribute>
</oct:AttributeList>
```

3.3.3.2.2 Extensions

Links MAY have <oct:ScubaKeys> internal extensions carried inside the <oct:Link> element. When they do, the <oct:ExtensionList> element MUST be present.

According to [8pus] §2, within a Link, the <oct:ScubaKeys> extension is not signed. Therefore, the <ds:DigestMethod> and <ds:DigestValue> elements SHALL NOT be carried inside the <oct:Extension> element.

The <oct:ScubaKeys> extension contains an encrypted version of the private/secret Scuba Sharing keys (in a <oct:PrivateKey> and a <oct:SecretKey> element) of the "to" Node with the public or secret Scuba Sharing key of the "from" Node. This encryption SHALL be expressed using the syntax specified in [xmlenc].

The oct:encoding attribute of the <oct:KeyData> element, a child of the both the <oct:PrivateKey> and <oct:SecretKey> elements, MUST be set to "xmlenc".

The child of this <oct:KeyData> element MUST be an <xenc:EncryptedData> element. The name of the encryption key MUST be advertised in the <ds:KeyInfo>/<ds:KeyName> element.

Note: Encrypting with a public key

If the encryption key is a public key, then:

- The <ds:KeyName> element MUST be the name of the pair to which the key belongs to. This element MUST reflect the same identity as paired field specified in [8pus] §2.4.5.
- In case the encrypted data (a private key for example) is too big to be encrypted directly with a public key, an intermediary 128 bit secret key is generated. The data MUST be encrypted with this intermediary key using [aes-128-cbc] and the intermediary key MUST be encrypted with the public key. The encrypted intermediary key MUST be encoded in a <ds:EncryptedKey> element. A example follows:

```
<!-- E(I, data) -->
<EncryptedData xmlns="http://www.w3.org/2001/04/xmlenc#">
  <EncryptionMethod
Algorithm="http://www.w3.org/2001/04/xmlenc#aes128-cbc"/>
  <KeyInfo xmlns="http://www.w3.org/2000/09/xmldsig#">
    <!-- E(PUBa, I) -->
    <EncryptedKey xmlns="http://www.w3.org/2001/04/xmlenc#">
      <EncryptionMethod
Algorithm="http://www.w3.org/2001/04/xmlenc#rsa-1_5"/>
      <KeyInfo xmlns="http://www.w3.org/2000/09/xmldsig#">
        <KeyName>urn:x-octopus:key-pair:300a</KeyName>
      </KeyInfo>
      <CipherData>
        <CipherValue>
fFeGD4KAPEmESz/jW6CkbRegpM5kyH0Oy/o/uDQ78PaShtvUMoozeO4a0b785YnB
13Qa1ZUEYqR9V5TCUaOcH7wxxvBEIsd1nYKkVOgW/kFnRr98UDFvU90PRqaEP/SA
Bb+JuAUmvxYX47qOVQqBQGGqzFssBDKmUk+s98dkPR8=
        </CipherValue>
      </CipherData>
    </EncryptedKey>
  </KeyInfo>
  <CipherData>
    <CipherValue>
c8LBj4BLzGOYv/GT3Y4w2XcwTYbr8fHNJhCOQjULuvoha/QYvZKKCpUY+nuCXC/s
t9TU+8tMtaMt1GUpkCZQhSaTNcluCSxOyBoA6Xh/bmyZLDJ78+aJ/sITmfNpJGdb
vTaI7x9DD1Mp1mvFEjpAUjTTvruN32g4bxsF7FD8C1RWNAc4hS96nFDgrmzoO5pR
dda6mswFKG5B0kY7mYbhacb1owXkAk1Wc/OuXA+QLHdUthxeajoXNPfAGRz9FM3b
puJxbxDAaaAJDxoReiTtS1nGaHhqa1hvLCpKk1zHBowHyvTvDLElLjHYEPeG6xSH
BbzpT298tdKUhXfaY6vvdceMdVXuBVL3eZP1jkJHDxeaBy1ce8xlQKZpo6Pjuxlb
bn5KUMt/PxWp7rLa5s786S740cwuN63+ZRgienxPK1CnYO3htMJ7hh/agvO9IyUD
RvcgnSEY9KA5Exy/6gIS/gouIjFU8r7056XcE4/IBodTWDkfyli/y8q5QA/0VaD9
Y3oER1p3pYuHwn/IeXM4gsBD31cgd7nvfK7lKYkZjowR9P6pSy57a+K4LZKDmfUH
zG/gZs2XcoPb9o6mVAEEej7+aLwqmoileykkR+0pkFntvqvXYRPkphhcVdzjzlMV
scpXBXfWx7wbQURXkiew7R4RihQy3wcv+ZFJpl9NsAE1yqyWy4rBobzZ7cTNMtfR
znhVlt+Wwq5G0IBxzU9WIFzFd/Rn2H9L4TI71LCa4VR3uNpf+XM8lp9LjLPRUnNh
28KrMdAddceyopYyiIF5p8idfh0//a/LKdE7JAk0q9ewk19ryqfl6CFeKI5oOMjh
kzNx3BR/iHxm31HIe3ZKtA==
```

```
    </CipherValue>
</CipherData>
</EncryptedData>
```

3.3.3.3 Protector

The <oct:ContentKeyReference> element MUST contain a <oct:Uid> element and MUST NOT include a <oct:Digest> element.

The <oct:ContentReference> elements (inside the <oct:ProtectedTargets> element) MUST contain a <oct:Uid> element and MUST NOT include a <oct:Digest> element.

Protector objects contain no mandatory attributes or extensions.

3.3.3.4 ContentKey

ContentKey objects contain no mandatory attributes or extensions. Therefore the <oct:AttributeList> and <oct:ExtensionList> elements are optional and MAY be ignored.

<oct:ContentKey> elements contain a <oct:SecretKey> element which represent the actual key needed to decrypt the content. The <oct:KeyData> associated with the <oct:SecretKey> MUST be encrypted. Therefore the oct:encoding attribute of <oct:KeyData> MUST be set to “xmlenc”. There MAY be several <oct:KeyData> elements if there are multiple expressions of the encrypted content key in the object.

There are two distinct cases for ContentKey objects (see Section 7):

a. Before the first revocation of a device or a PC application: in this case, the content key CK represented by the <oct:SecretKey> element MUST be encrypted by a Scuba key (public or secret) of the entity to which the content is bound to (for example the user.) See Section 7.3.

b. After the first revocation the content key MUST first be encrypted according to the [Starfish] broadcast encryption scheme. The resulting data is then encrypted with a Scuba key (public or secret) of the entity to which the content is bound to. See Section 7.4.

3.3.3.5 Controller

Controller objects MAY contain attributes or internal extensions. However, when these elements are present in either a <oct:AttributeList> or <oct:ExtensionList> elements then these elements SHALL be processed.

An optional Domain Serial Number MAY be attached to the Controller object depending on the domain policy. When present it SHALL be visible to the Plankton Virtual Machine as described in Section 3.2.3.

The <oct:ControlReference> MUST have a <oct:Digest> element. The <ds:DigestValue> element MUST contain the base64 [MIME] encoding of the digest of the referenced control (see [8pus] §2).

The <oct:ContentKeyReference> MUST have a <oct:Digest> element. The <ds:DigestValue> element MUST contain the base64 [MIME] encoding of the digest of

the referenced ContentKey (see [8pus] §2). The the algorithms, identifiers and representation of the <oct:Digest> MUST follow the guidance given in Section 3.3.4

3.3.3.6 Control

Control objects MAY contain attributes or internal extensions. When these elements are present in either a <oct:AttributeList> or <oct:ExtensionList> elements, these elements SHALL made available to the DRM engine.

The oct:protocol attribute of the <oct:ControlProgram> element MUST be signaled using the Standard Control Protocol identifier defined in [8pus] §3.2.3.

The oct:type attribute of the <oct:CodeModule> element MUST be signaled using the Plankton Control Module identifier defined in [8pus] §3.2.4.

3.3.4 Signatures: Use of XML Digital Signature [xmldsig]

According to the Octopus Objects specification ([8pus] §2) the Octopus objects that are directly signed (unless an exception is given) include:

- Nodes
- Links
- Controllers
- Controls
    - A Control need not be directly signed if it is indirectly covered by a secure digest in a referring Controller object.
    - A Control need not be independently signed if it is covered by the signature of a containing Link object.
- Extensions (depending on the data they carry)

The signatures MUST follow the profile given in Section 12.2. A <ds:Signature> element SHALL be present in the <oct:Bundle> object that contains the XML representation of the signed Octopus objects. There MAY be multiple signatures covering the same object such as a Controller being covered by a (certified) public key signature and an HMAC (symmetric key) signature.

3.3.4.1 Controller Objects

Controller objects MUST have at least one HMAC signature for each ContentKey referenced in its list of controlled targets. The key used for each of those signatures MUST be the content key contained in the ContentKey object referenced.
The DRM Client MUST verify the HMAC signature for every Controller object prior to invoking the Perform routine of any Action [8pus] §3.2.6.3 on an Octopus Control.

Controllers MAY also have a public key signature. If such a signature is present, this signature MUST also appear as a <ds:Reference> in each of the HMAC signatures for the object. To achieve this, this <ds:Signature> element MUST have an xs:ID-typed attribute (unique within the enclosing XML document) which is used as the ds:URI attribute in one of the <ds:Reference> elements of each of the HMAC signatures. The verifier MUST verify the public key signature if it is present. The verifier MUST reject public key signatures that are not corroborated by the HMAC signature.

Example:

1121

```
<ds:Signature Id="Signature.0" >
  <ds:SignedInfo>
    <ds:CanonicalizationMethod Algorithm="http://www.w3.org/2001/10/xml-exc-c14n#"/>
    <ds:SignatureMethod Algorithm="http://www.w3.org/2000/09/xmldsig#rsa-sha1"/>
    <ds:Reference URI="#Controller">
      <ds:Transforms>
        <ds:Transform Algorithm="http://www.octopus-drm.com/octopus/specs/cbs-1_0"/>
      </ds:Transforms>
      <ds:DigestMethod Algorithm="http://www.w3.org/2000/09/xmldsig#sha1"/>
      <ds:DigestValue>G1zXF9Sz/zCwH6MaFm0ObOQcxuk=/ds:DigestValue>
    </ds:Reference>
  </ds:SignedInfo>

<ds:SignatureValue>mjoyW+w2S9iZDG/ha4eWYD1RmhQuqRuuSN977NODpzwUD02FdsAIC
VjAcw7f4nFWuvtawW/clFzY
P/pjFebESCvurHUsEaR1/LYLDkpWWxh/LlEp4r3yR9kUs0AU5a4BDxDxQE7nUdqU9YMpnjAZ
EGpu
xdPeZJM1vyKqNDpTk94=</ds:SignatureValue>
  <ds:KeyInfo>
    <ds:X509Data><ds:X509Certificate>MIICh...</ds:X509Certificate></ds:X509Data>
</ds:KeyInfo>
</ds:Signature>
<ds:Signature>
  <ds:SignedInfo>
    <ds:CanonicalizationMethod Algorithm="http://www.w3.org/2001/10/xml-exc-c14n#"/>
    <ds:SignatureMethod Algorithm="http://www.w3.org/2000/09/xmldsig#hmac-sha1"/>
    <ds:Reference URI="#Signature.0">
      <ds:DigestMethod Algorithm="http://www.w3.org/2000/09/xmldsig#sha1"/>
      <ds:DigestValue>AqPV0nvNj/vc51IcMyKJngGNKtM=</ds:DigestValue>
    </ds:Reference>
    <ds:Reference URI="#Controller">
      <ds:Transforms>
        <ds:Transform Algorithm="http://www.octopus-drm.com/octopus/specs/cbs-1_0"/>
      </ds:Transforms>
      <ds:DigestMethod Algorithm="http://www.w3.org/2000/09/xmldsig#sha1"/>
      <ds:DigestValue>G1zXF9Sz/zCwH6MaFm0ObOQcxuk=</ds:DigestValue>
    </ds:Reference>
  </ds:SignedInfo>
  <ds:SignatureValue>TcKBsZZy+Yp3doOkZ62LTfY+ntQ=</ds:SignatureValue>
  <ds:KeyInfo>
    <ds:KeyName>urn:x-octopus:secret-key:2001</ds:KeyName>
  </ds:KeyInfo>
</ds:Signature>
```

4 Marlin Core System Roles and Services

Marlin Core system entities implement the defined Marlin Roles and Services. Certain Roles are optional. However, when implemented, they MUST conform to the specification. Certain typical device functions MAY require the aggregation of several Roles. For example, a Media Device that wants to render Marlin content MUST implement the Device Role and the DRM Client Role.

4.1 Overview

This section defines the roles and services part of the Marlin specifications. This section will enumerate these roles and the services and clients they correspond to. A later section of this document specifies the messages and protocols by which the client and services communicate.

Roles	Services	Clients
Device	Inspection	Inspection Provide Security Data
Domain Information Provider (opt)	Provide Domain Information	
Security Data Provider (opt)	Provide Security Data (event-driven)	DRM Client Information
DRM Object Provider (opt)	Provide DRM Objects (event-driven)	Proximity Check DRM Client Information
DRM Client (opt)	Proximity Check License Transfer DRM Client Information	Proximity Check License Transfer Provide DRM Objects Domain Information (opt)

Table 4-1 Role to Client/Service Mapping

(opt) indicates that the implementation of this service/client is OPTIONAL
(event-driven) means that clients MAY subscribe to receive triggers to access this service, see the section on protocols.

The following table summarizes the set of URIs used as attribute values for conveying the above roles.

Role	URI
Device	urn:marlin:core:role:device
Domain Info Provider	urn:marlin:core:role:domain-information-provider
Security Data Provider	urn:marlin:core:role:security-data-provider
DRM Object Provider	urn:marlin:core:role:drm-object-provider
DRM Client	urn:marlin:core:role:drm-client

Table 4-2 Role Identifiers

Refer to Section 12.5.4.2 for a description of the mechanism used to convey the role information.

4.2 Roles Definitions

Note: According to [NEMO] §4, roles SHALL be encoded as SAML 1.1 [SAML1.1] attribute assertions. Also note that we have defined namespace equivalence for this specific attribute namespace (see Table 1-3.)

4.2.1 Device

Each NEMO Node implemented on a Marlin core host SHALL have this role. This means that all the nodes in Marlin MUST implement an Inspection service and client based on WS-MetadataExchange [WS-MEX].

All Marlin devices SHALL be able to access a Security Data Provider so that they can update themselves with new security metadata. They MAY subscribe to Security Data Events and receive notifications when new Security Metadata is available for them.

4.2.2 Domain Information Provider

Each NEMO Node implementing this role SHALL be able to get domain information requests from DRM Clients and provide DRM Clients with the information about the domains they know about including the Provide DRM Objects service endpoint where they can register/deregister to the domain.

The interaction between the Domain Information Provider, the Domain Manager and the DRM Object Provider is not part of the core specification and will be specified for each type of domain in the delivery system specifications.

4.2.3 Security Data Provider

Each NEMO Node implementing this role SHALL be able to provide Devices with Security Metadata. The Provide Security Data Service is event-driven so that Marlin Devices can subscribe to Security Data Events in order to receive notifications when new Security Metadata is available for them (the same way DRM Clients do for DRM Object Providers, see below).

4.2.4 DRM Object Provider

Each NEMO node implementing this role SHALL be able to provide DRM clients with DRM objects in a generic way. Such a role MAY be used in conjunction with a domain manager and serve as a repository for getting the latest membership tokens of a domain.

The service hosted by a DRM Object Provider SHALL be event-driven, which means that clients (here DRM Clients) MAY subscribe to notifications that trigger the invocation of the DRM Object Provider service.
DRM Clients MAY ask to subscribe to DRM Objects notifications. In this case, when the DRM Object Provider has new objects for the subscriber, it SHALL notify the client. Subsequent to notification the DRM client can invoke the DRM Object Provider service and get the objects (Event-driven invocation). The DRM Client MAY also invoke the DRM Object Provide service directly without notification (direct invocation).

DRM Clients SHOULD subscribe to notification events of DRM Object Providers

4.2.5 DRM Client

Each NEMO node implementing this role SHALL be able to:

- Export information about its Octopus Personality Node as well as its Security Metadata.
- Be the sink device in a proximity check (so that a Domain Manager, if required by the domain policy, can determine whether or not the DRM Client can be registered to the domain.
- Check the proximity to another DRM.
- Invoke a Provide DRM Objects Service (directly or through subscription/notification)

Each NEMO node implementing this role MAY:

- Request information of the Domain Information Provider (the IDs of the Octopus Nodes representing the Domains that the Domain Information Provider can access, as well as the type of Policy of these Domains and which DRM Object Provider to talk to in order to register to these domains).

5 Marlin Core System Protocols

This section defines the core connectivity of Marlin devices.

5.1 NEMO Architecture for Marlin

5.1.1 Concepts and Architecture

In Marlin, communicating entities are NEMO nodes. These nodes have a set of credentials which are keys and attributes. The attributes of primary importance in Marlin are called roles. When a NEMO node has been certified as having a certain role, it implies that this node implements a number of services and clients. Of course, a single NEMO node may have multiple roles.

The figure below details the components of the NEMO Marlin Architecture

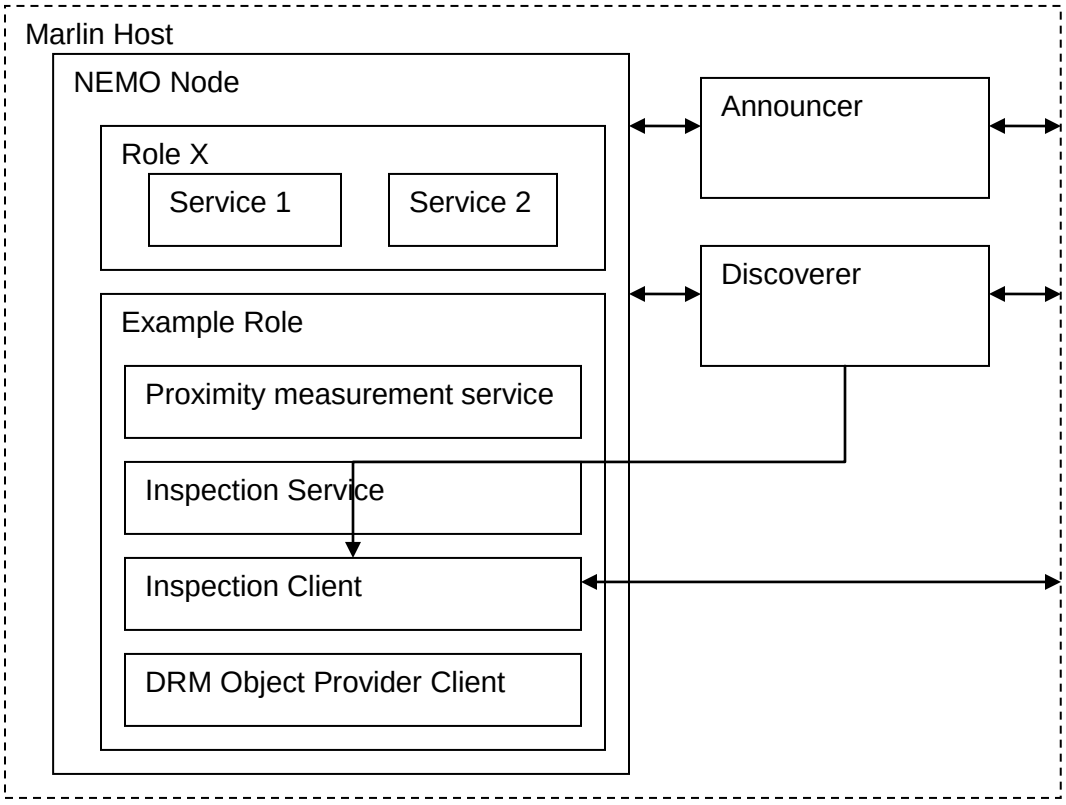


Figure 5-1 Example: NEMO Marlin Architecture

NOTE:
In Figure 5-1, the Announcer and the Discoverer are not part of a NEMO node but a given implementation could include them in the boundaries of a NEMO Node.

5.1.1.1 NEMO Nodes

NEMO nodes are logical entities that have two types of credentials:

- Authentication and Encryption keys: these are certified by an identity provider that binds the ID of the NEMO node and these keys.
- Certified attributes that are issued for the Node (bound to the node ID). Specifically, Marlin defines role and specification version attributes. A Node can have (or "implement") multiple roles. Roles are further described below.

Logically there can be multiple NEMO nodes on a Marlin Host. However, such an implementation decision would not impact the Discovery, Inspection and Service Access architecture.

5.1.1.2 NEMO Roles

As in [Coral], having a NEMO role implies that a set of services (and clients) are implemented by a NEMO node. The NEMO roles for Marlin are defined in Section 4.1 of this document.

5.1.1.3 Announcer

The announcer is the building block that implements the Discovery advertisements for the NEMO node. The announcer MAY be shared between multiple NEMO nodes. It is also responsible for answering Discovery search requests sent by other hosts discoverers.

5.1.1.4 Discoverer

The discoverer is responsible for sending the Discovery search requests and handling the Discovery search responses and advertisements. It SHOULD pass information to the Inspection client so that the service being searched for can be inspected. Like the announcer, it MAY be shared between multiple NEMO nodes.

5.1.1.5 Inspection Client

The inspection client utilizes [WS-MEX] for the inspection phase. It will get the necessary information (e.g., WSDL) so that the requesting NEMO node can invoke the service being searched for.

5.2 Message Security Policies

5.2.1 Overview

The NEMO Security Bindings specification ([NEMO] §3) defines the NEMO basic security protocol. This protocol SHALL be used in Marlin for message security.

Trusted time SHALL be used in order to check all the NEMO credentials ([X509] public key certificates and SAML Assertions). Trusted time SHALL also be used when a message timestamp is required (in the case of 'Integrity + Freshness' or 'Full Security').

The table below summarizes the different modes of this protocol:

Protocol Policy	Integrity	Nonce	Timestamp	Confidentia

				lity
No Security	NO	NO	NO	NO
Freshness Only	NO	YES	OPTIONAL	NO
Integrity Only	YES	NO	NO	NO
Integrity + Freshness	YES	YES	YES	NO
Confidentiality Only	NO	NO	NO	YES
Full Security	YES	YES	YES	YES

Table 5-1 Protocol Policy Characteristics

For a given message exchange, compatible message security policies SHALL be implemented. Compatible message security policies which will not hinder a receiver from implementing its policy. The following matrix outlines the compatible policies.

Receiver Protocol Policy	Compatible Sender Protocol Policy
No Security	No Security, Freshness Only, Integrity Only, Integrity + Freshness
Freshness Only	Freshness Only, Integrity + Freshness
Integrity Only	Integrity Only, Integrity + Freshness
Integrity + Freshness	Integrity + Freshness
Confidentiality Only	Confidentiality Only, Full Security
Full Security	Full Security

Table 5-2 Compatible Policies

5.2.2 Protocol Security Policy Identifiers

Each of the services defined by this specification informatively describes the protocol security policies of the service. This specification normatively defines the each of the desired policies using URIs.

We anticipate that the protocol security policies defined in this specification and in delivery system specifications will be added or change. With that in mind it is desirable to maintain a uniform identification scheme as outlined in Section1.3.3. However, we do deviate from this slightly in order to indicate the version of the specification the identifier is introduced and to support the possiblility that two or more policies for the same service could be introduced in that specification version.
For example;

urn:marlin:core:1-2:nemo:services:foobar:policy:0
urn:marlin:core:1-2:nemo:services:foobar:policy:1

reflect the policies of the “foobar” service. This identifier also indicates that the policy was introduces in version 1.2 (“1-2”) of the specification and that there are two distinct policies for the service (policy “0” and policy “1”).[3]

[3] Note that in previous specifications we used “1.x” rather than “1-x” but going forward we intend to use the dash form. Since this is merely an identifier to be evaluated as a whole this does not pose any incompatibilities.

5.2.3 Request Policies

The request policies (except the 'No Security') may mandate the inclusion of some or all the security credentials of the client. The following example depicts these credentials:

```
<!-- Client's NEMO Encryption Key -->
<wssp:SecurityToken
  nemosec:Usage="http://nemo.intertrust.com/2005/10/security/secure-
protocol/basic/1.0#response-encryptionKey">
   <wssp:TokenType>
       http://docs.oasis-open.org/wss/2004/01/oasis-200401-wss-x509-token-profile-
1.0#X509PKIPathv1
   </wssp:TokenType>
</wssp:SecurityToken>

<!-- Client's NEMO Authentication Key -->
<wssp:SecurityToken
  nemosec:Usage="http://nemo.intertrust.com/2005/10/security/secure-
protocol/basic/1.0#request-signingKey">
   <wssp:TokenType>
       http://docs.oasis-open.org/wss/2004/01/oasis-200401-wss-x509-token-profile-
1.0#X509PKIPathv1
   </wssp:TokenType>
</wssp:SecurityToken>

<!-- Client's Roles (application policy attribute tokens) -->
<wssp:SecurityToken> + <!-- one or more -->
  <wssp:TokenType>
    http://nemo.intertrust.com/2004/attribute/role</wssp:TokenType>
</wssp:SecurityToken>
```

To minimize repetition the above example will be referenced in the following sections as &client-credentials;

5.2.3.1 No Security

This policy does not add any protection to the message. Note that the use of this policy may be sub-optimal with respect to the requirements of the response policy. For example, if the response policy requires 'Freshness' then it is incumbent upon the requester to seed the protocol with the proper data.

5.2.3.2 Freshness Only

This policy enables seeding of the response policy when it requires 'Freshness'. This message exchange pattern assumes that the requestor retains the request nonce for comparison with the response message.

The Core devices in the home environment may have their clocks out of synchronization. Thus the timestamp on the messages SHOULD be considered informative.

The following [WS-POL] example expresses an 'Freshness' policy

```
  <!-- Request Policy ================================== -->
  <wsp:Policy>
    <!-- Protocol -->
    <nemop:ProtocolAssertion>
      <nemop:Reference URI="http://nemo.intertrust.com/2005/10/security/secure-
protocol/basic/1.0"/>
      <nemosec:Step type="request"/>
    </nemop:ProtocolAssertion>

    &client-credential;

    <!-- Nonce -->
    <nemop:Nonce
nemop:Usage="http://nemo.intertrust.com/2005/10/security/secure-
protocol/basic/1.0/policyAssertion#nonce"/>

    <!-- Message Age -->
    <wssp:MessageAge Age="3600"
nemop:Usage="http://nemo.intertrust.com/2005/10/security/secure-
protocol/basic/1.0/policyAssertion#timestamp"/>

  </wsp:Policy>
```

5.2.3.3 Integrity Only

The following [WS-POL] example expresses an 'Integrity Only' policy:

```
  <!-- Request Policy ================================== -->
  <wsp:Policy>
    <!-- Protocol -->
    <nemop:ProtocolAssertion>
      <nemop:Reference URI="http://nemo.intertrust.com/2005/10/security/secure-
protocol/basic/1.0"/>
      <nemosec:Step type="request"/>
    </nemop:ProtocolAssertion>

    &client-credentials;

    <!-- Integrity -->
    <wssp:Integrity
nemop:Usage="http://nemo.intertrust.com/2005/10/security/secure-
protocol/basic/1.0/policyAssertion#integrity">
      <wssp:MessageParts Dialect="http://nemo.intertrust.com/2004/policy#part">
                  wsp:Body()
                  nemop:Token(http://nemo.intertrust.com/2005/10/security/secure-
protocol/basic/1.0#request-toNode")
          </wssp:MessageParts>
    </wssp:Integrity>
  </wsp:Policy>
```

5.2.3.4 Integrity + Freshness

The Core devices in the home environment may have their clocks out of synchronization. Thus the timestamp on the messages SHOULD be considered informative.

The following [WS-POL] example expresses an ‘Integrity + Freshness’ policy

```
<!-- Request Policy ================================== -->
<wsp:Policy>
  <!-- Protocol -->
  <nemop:ProtocolAssertion>
    <nemop:Reference URI="http://nemo.intertrust.com/2005/10/security/secure-
protocol/basic/1.0"/>
    <nemosec:Step type="request"/>
  </nemop:ProtocolAssertion>

  &client-credential;

  <!-- Nonce -->
  <nemop:Nonce
nemop:Usage="http://nemo.intertrust.com/2005/10/security/secure-
protocol/basic/1.0/policyAssertion#nonce"/>

  <!-- Message Age -->
  <wssp:MessageAge Age="3600"
nemop:Usage="http://nemo.intertrust.com/2005/10/security/secure-
protocol/basic/1.0/policyAssertion#timestamp"/>

  <!-- Integrity -->
  <wssp:Integrity
nemop:Usage="http://nemo.intertrust.com/2005/10/security/secure-
protocol/basic/1.0/policyAssertion#integrity">
    <wssp:MessageParts Dialect="http://nemo.intertrust.com/2004/policy#part">
          wsp:Body()
          nemop:Token(http://nemo.intertrust.com/2005/10/security/secure-
protocol/basic/1.0#request-nonce")]
      nemop:Token(http://nemo.intertrust.com/2005/10/security/secure-
protocol/basic/1.0#request-timestamp")]
          nemop:Token(http://nemo.intertrust.com/2005/10/security/secure-
protocol/basic/1.0#request-toNode")]
        </wssp:MessageParts>
  </wssp:Integrity>
</wsp:Policy>
```

5.2.3.5 Confidentiality Only

The following [WS-POL] example expresses a ‘Confidentiality Only’ policy:

```
<!-- Request Policy ================================== -->
<wsp:Policy>
  <!-- Protocol -->
  <nemop:ProtocolAssertion>
```

```
    <nemop:Reference URI="http://nemo.intertrust.com/2005/10/security/secure-
protocol/basic/1.0"/>
    <nemosec:Step type="request"/>
  </nemop:ProtocolAssertion>

  &client-credentials;

  <!-- Confidentiality -->
  <wssp:Confidentiality
nemop:Usage="http://nemo.intertrust.com/2005/10/security/secure-
protocol/basic/1.0/policyAssertion#confidentiality">
    <wssp:MessageParts Dialect="http://nemo.intertrust.com/2004/policy#part">
          nemop:Token(http://nemo.intertrust.com/2005/10/security/secure-
protocol/basic/1.0#request-messageKey")]
          wsp:Body()
        </wssp:MessageParts>
  </wssp:Confidentiality>
 </wsp:Policy>
```

1326 **5.2.3.6 Full Security**

1327 The following [WS-POL] example expresses a 'Full Security' policy:

```
 <!-- Request Policy ================================== -->
 <wsp:Policy>
  <!-- Protocol -->
  <nemop:ProtocolAssertion>
    <nemop:Reference URI="http://nemo.intertrust.com/2005/10/security/secure-
protocol/basic/1.0"/>
    <nemosec:Step type="request"/>
  </nemop:ProtocolAssertion>

  &client-credential;

  <!-- Nonce -->
  <nemop:Nonce
nemop:Usage="http://nemo.intertrust.com/2005/10/security/secure-
protocol/basic/1.0/policyAssertion#nonce"/>

  <!-- Message Age -->
  <wssp:MessageAge Age="3600"
nemop:Usage="http://nemo.intertrust.com/2005/10/security/secure-
protocol/basic/1.0/policyAssertion#timestamp"/>

  <!-- Integrity -->
  <wssp:Integrity
nemop:Usage="http://nemo.intertrust.com/2005/10/security/secure-
protocol/basic/1.0/policyAssertion#integrity">
    <wssp:MessageParts Dialect="http://nemo.intertrust.com/2004/policy#part">
          wsp:Body()
       nemop:Token(http://nemo.intertrust.com/2005/10/security/secure-
```

```
protocol/basic/1.0#request-messageKey")]
            nemop:Token(http://nemo.intertrust.com/2005/10/security/secure-
protocol/basic/1.0#request-nonce")]
        nemop:Token(http://nemo.intertrust.com/2005/10/security/secure-
protocol/basic/1.0#request-timestamp")]
             nemop:Token(http://nemo.intertrust.com/2005/10/security/secure-
protocol/basic/1.0#request-toNode")]
          </wssp:MessageParts>
   </wssp:Integrity>

   <!-- Confidentiality -->
   <wssp:Confidentiality
nemop:Usage="http://nemo.intertrust.com/2005/10/security/secure-
protocol/basic/1.0/policyAssertion#confidentiality">
     <wssp:MessageParts Dialect="http://nemo.intertrust.com/2004/policy#part">
            nemop:Token(http://nemo.intertrust.com/2005/10/security/secure-
protocol/basic/1.0#request-messageKey")]
            wsp:Body()
          </wssp:MessageParts>
   </wssp:Confidentiality>

  </wsp:Policy>
```

5.2.4 Response Policies

5.2.4.1 No Security

This policy does not add any protection to the message. It just leaves the message as is.

5.2.4.2 Integrity Only

The following [WS-POL] example expresses an 'Integrity Only' policy:

```
  <!-- Response Policy ================================== -->
  <wsp:Policy>
    <!-- Protocol -->
    <nemop:ProtocolAssertion>
      <nemop:Reference URI="http://nemo.intertrust.com/2005/10/security/secure-
protocol/basic/1.0"/>
     <nemosec:Step type="response"/>
    </nemop:ProtocolAssertion>

    <!-- Integrity -->
    <wssp:Integrity
nemop:Usage="http://nemo.intertrust.com/2005/10/security/secure-
protocol/basic/1.0/policyAssertion#integrity">
     <wssp:MessageParts Dialect="http://nemo.intertrust.com/2004/policy#part">
       wsp:Body()
            nemop:Token(http://nemo.intertrust.com/2005/10/security/secure-
protocol/basic/1.0#response-toNode")]
          </wssp:MessageParts>
   </wssp:Integrity>
```

```
  </wsp:Policy>
```

5.2.4.3 Integrity + Freshness

Core devices in the home environment may have their clocks out of synchronization. Thus the timestamp on the messages SHOULD be considered informative.

The following [WS-POL] example expresses an ‘Integrity + Freshness’ policy

```
 <!-- Response Policy ================================== -->
 <wsp:Policy>
  <!-- Protocol -->
  <nemop:ProtocolAssertion>
   <nemop:Reference URI="http://nemo.intertrust.com/2005/10/security/secure-
protocol/basic/1.0"/>
    <nemosec:Step type="response"/>
  </nemop:ProtocolAssertion>

  <!-- Nonce -->
  <nemop:Nonce
nemop:Usage="http://nemo.intertrust.com/2005/10/security/secure-
protocol/basic/1.0/policyAssertion#nonce"/>

  <!-- Message Age -->
  <wssp:MessageAge Age="3600"
nemop:Usage="http://nemo.intertrust.com/2005/10/security/secure-
protocol/basic/1.0/policyAssertion#timestamp"/>

  <!-- Integrity -->
  <wssp:Integrity
nemop:Usage="http://nemo.intertrust.com/2005/10/security/secure-
protocol/basic/1.0/policyAssertion#integrity">
    <wssp:MessageParts Dialect="http://nemo.intertrust.com/2004/policy#part">
          wsp:Body()
          nemop:Token(http://nemo.intertrust.com/2005/10/security/secure-
protocol/basic/1.0#request-nonce")]
          nemop:Token(http://nemo.intertrust.com/2005/10/security/secure-
protocol/basic/1.0#response-nonce")]
      nemop:Token(http://nemo.intertrust.com/2005/10/security/secure-
protocol/basic/1.0#response-timestamp")]
          nemop:Token(http://nemo.intertrust.com/2005/10/security/secure-
protocol/basic/1.0#response-toNode")]
        </wssp:MessageParts>
  </wssp:Integrity>
 </wsp:Policy>
```

5.2.4.4 Confidentiality Only

The following [WS-POL] example expresses a ‘Confidentiality Only’ policy:

```
 <!-- Response Policy ================================== -->
 <wsp:Policy>
  <!-- Protocol -->
```

```
   <nemop:ProtocolAssertion>
     <nemop:Reference URI="http://nemo.intertrust.com/2005/10/security/secure-
protocol/basic/1.0"/>
     <nemosec:Step type="response"/>
   </nemop:ProtocolAssertion>

   <!-- Confidentiality -->
   <wssp:Confidentiality
nemop:Usage="http://nemo.intertrust.com/2005/10/security/secure-
protocol/basic/1.0/policyAssertion#confidentiality">
     <wssp:MessageParts Dialect="http://nemo.intertrust.com/2004/policy#part">
           nemop:Token(http://nemo.intertrust.com/2005/10/security/secure-
protocol/basic/1.0#response-messageKey")]
           wsp:Body()
         </wssp:MessageParts>
   </wssp:Confidentiality>
 </wsp:Policy>
```

1340 **5.2.4.5 Full Security**

1341 The following [WS-POL] example expresses a ‘Full Security’ policy:

```
 <!-- Response Policy ================================== -->
 <wsp:Policy>
   <!-- Protocol -->
   <nemop:ProtocolAssertion>
     <nemop:Reference URI="http://nemo.intertrust.com/2005/10/security/secure-
protocol/basic/1.0"/>
     <nemosec:Step type="response"/>
   </nemop:ProtocolAssertion>

   <!-- Nonce -->
   <nemop:Nonce
nemop:Usage="http://nemo.intertrust.com/2005/10/security/secure-
protocol/basic/1.0/policyAssertion#nonce"/>

   <!-- Message Age -->
   <wssp:MessageAge Age="3600"
nemop:Usage="http://nemo.intertrust.com/2005/10/security/secure-
protocol/basic/1.0/policyAssertion#timestamp"/>

   <!-- Integrity -->
   <wssp:Integrity
nemop:Usage="http://nemo.intertrust.com/2005/10/security/secure-
protocol/basic/1.0/policyAssertion#integrity">
     <wssp:MessageParts Dialect="http://nemo.intertrust.com/2004/policy#part">
           wsp:Body()
       nemop:Token(http://nemo.intertrust.com/2005/10/security/secure-
protocol/basic/1.0#response-messageKey")]
       nemop:Token(http://nemo.intertrust.com/2005/10/security/secure-
protocol/basic/1.0#request-nonce")]
```

```
            nemop:Token(http://nemo.intertrust.com/2005/10/security/secure-
protocol/basic/1.0#response-nonce")]
        nemop:Token(http://nemo.intertrust.com/2005/10/security/secure-
protocol/basic/1.0#response-timestamp")]
            nemop:Token(http://nemo.intertrust.com/2005/10/security/secure-
protocol/basic/1.0#response-toNode")]
          </wssp:MessageParts>
   </wssp:Integrity>

   <!-- Confidentiality -->
   <wssp:Confidentiality
nemop:Usage="http://nemo.intertrust.com/2005/10/security/secure-
protocol/basic/1.0/policyAssertion#confidentiality">
     <wssp:MessageParts Dialect="http://nemo.intertrust.com/2004/policy#part">
       nemop:Token(http://nemo.intertrust.com/2005/10/security/secure-
protocol/basic/1.0#response-messageKey")]
            wsp:Body()
          </wssp:MessageParts>
   </wssp:Confidentiality>

 </wsp:Policy>
```

5.3 Message Faults

Marlin utilizes SOAP for much of its messaging substrate. SOAP defines a mechanism to return fault indications to clients of a service. This section defines the general mechanisms by which a service can report a fault to a client of the service.

There are two types of fault responses in Marlin System Protocols. One is for errors pertaining to a SOAP header processing. Another is for errors pertaining to a SOAP body processing.

The schema which supports the fault mechanism is in Appendix A.7.

5.3.1 Faults for SOAP Header Processing

When an error occurs during the processing of the <S11:Header> element, the processor SHOULD return a fault code. In accordance with the mechanism defined in [SOAP11] the <S11:Fault> element MUST appear within the <S11:Body> element. Per this profile additional detail information MAY be returned by using the <nemoc:FaultDetails> element.

The detail information MUST be supplied as attributes of a <exc:ServiceException> element or conveyed as a child element of an <exc:ServiceException> element(s).

The syntax for <exc:ServiceException> element follows:
exc:ServiceException
A child element of the <nemoc:FaultDetails> element.

exc:ServiceException/@name

This attribute MUST either contain the following string identifier or a value defined in a delivery system specification:

- exc:NemoMessageProcessingException: This indicates that an exception occurred while processing the SOAP header.

exc:ServiceException/exc:RedirectURL
The <exc:RedirectURL> element is OPTIONAL. However when present, it MUST contain a valid URL. This URL is intended to give guidance to the user to help resolve the problem. The processor of the fault message should direct the user to resolve the URL.

exc:ServiceException/exc:Details

The following behaviors for <exc:Details> element are defined in Marlin Core.

- For exc:ServiceException/@name="exc:NemoMessageProcessingException": The <exc:Details> element is OPTIONAL.

When several distinct errors occur during a processing of a SOAP header, multiple <exc:ServiceException> elements MAY be returned in the <nemoc:FaultDetails> element.

Note: When the value of the name attribute in <exc:ServiceException> element is other than values defined in above, it MUST be defined a delivery system specification or be an organization specific URI defined for its proprietary purpose. If a client encounters an unknown value of the name attribute in <exc:ServiceException> element, the client SHOULD log the information in the <exc:RedirectURL> and <exc:Details> elements..

5.3.2 Faults for SOAP Body Processing

When an error occurs during processing of the <S11:Body> element, the processor SHOULD return a fault code. In accordance with the mechanism defined in [SOAP11] the <S11:Fault> element MUST appear within the <S11:Body> element. For body faults, detail information MAY be returned using <S11:detail> element.
The <exc:ServiceException> element MUST appear as a child element of the <S11:detail> element. The <exc:ServiceException> element is used to convey the actual detail information.

The syntax for <exc:ServiceException> element is as follows:
exc:ServiceException
The child element of <S11:detail> element.

exc:ServiceException/@name
This attribute MUST either contain the following string identifier or a value defined in a delivery system specification:

- exc:NemoMessageProcessingException: This indicates that an exception occurred while processing the SOAP body.

exc:ServiceException/exc:RedirectURL

The <exc:RedirectURL> element is OPTIONAL. However when present, it MUST contain a valid URL. This URL is intended to give guidance to the user to help resolve the problem. The processor of the fault message should direct the user to resolve the URL.

exc:ServiceException/exc:Details
The following behavior for <exc:Details> element are defined in Marlin Core.

- For exc:ServiceException/@name=”exc:NemoMessageProcessingException”: The <exc:Details> element is OPTIONAL.

Note: When the value of the name attribute in <exc:ServiceException> element is other than values defined in above, it MUST be defined a delivery system specification or be an organization specific URI defined for its proprietary purpose. If a client encounters an unknown value of the name attribute in <exc:ServiceException> element, the client SHOULD log the information in the <exc:RedirectURL> and <exc:Details> elements..

5.3.3 Fault Addressing

When the wsdl:operation includes a soapAction attribute and wsdl:input, wsdl:output, and wsdl:fault, the specified soapActiion attribute value is used in the <wsa:Action> element for request, response, and fault messages.
For example, in the following wsdl definition, the value "foo" is used for the value of the <wsa:Action> element for request, response, and fault messages:

```
    <wsdl:binding ...>
      ...
      <wsdl:operation ..>
      <wsdlsoap:operation soapAction="foo"/>
        <wsdl:input>...</wsdl:input>
        <wsdl:output>...</wsdl:output>
        <wsdl:fault ...>...</wsdl:fault>
      </wsdl:operation>
   </wsdl:binding>
```

When the wsdl:operation includes a soapAction attribute but does not include a <wsdl:fault> element, the default value is used in the <wsa:Action> element for fault messages.
For example, in the following wsdl definition, the default value "http://www.w3.org/2005/08/addressing/soap/fault " is used as the value of the <wsa:Action> element in a fault message:

```
   <wsdl:binding ...>
     ...
     <wsdl:operation ..>
     <wsdlsoap:operation soapAction="foo"/>
       <wsdl:input>...</wsdl:input>
       <wsdl:output>...</wsdl:output>
     </wsdl:operation>
   </wsdl:binding>
```

The default value is the string "http://www.w3.org/2005/08/addressing/soap/fault".

5.4 Discovery

5.4.1 Overview

The goal of the discovery layer is to enable Marlin services to be found.

This binding is based on the more general Nemo Discovery/Inspection Bindings specification ([NEMO] §6). This section further specifies discovery to ensure interoperability between two implementations of this specification.

The discovery mechanism is compatible with [UPnPDevArch1.0.1] §1 and §2. When a device is added to the network, the UPnP discovery protocol allows that device to advertise essential specifics about the device or one of its services, e.g., its type, its identifier, and a reference which can be used to acquire more detailed information.

Marlin Discovery leverages the UPnP discovery protocol. It compatibly extends it to advertise NEMO node specifics. When a NEMO node coexists within the context of an UPnP device, the Device Description document of an UPnP device is extended with information and end-point addresses for an UPnP Control Point (discoverer) to further inspect the NEMO node.

When no UPnP device can be leveraged, the Marlin device SHALL implement an UPnP Basic Device as defined in the [UPnPBasicDev1.0.1] document. In this case, the UPnP device simply extends it with UPnP compatible NEMO node Marlin extension.

Marlin devices SHALL conform to UPnP 1.0 [UPnPDevArch1.0.1] including the implementers guide errata [UPnPImplGuid]. Note that the UPnP 1.0.1 draft specification [UPnPDevArch1.0.1] is an update of UPnP 1.0 that includes all the latest errata.

5.4.2 Description Extension

In UPnP, after a control point has discovered a device, the control point has minimal information about the device. To obtain more device information and its capabilities, the control point retrieves the device's description from the URL supplied in the LOCATION header of the search response. The UPnP description for a device is expressed in XML and includes a list of embedded devices and services as well as vendor specific information.

The NEMO node specific information SHALL be listed as an extension to the UPnP root device description and includes URLs and parameters which are used to invoke an inspection service which implements [WS-MEX].

The description SHOULD be based on a standard UPnP Device Template. [UPnPBasicDev] specifies an extension mechanism so that vendors can differentiate their devices. The extension syntax enables extensibility by mandating devices and control points ignore all unknown elements and their sub elements or content and unknown attributes and their values.

To retrieve an UPnP Description a Control point (Discoverer) issues a HTTP GET request to the URL supplied in the LOCATION header of the search response message. The device returns the device description.

1510
1511 The following is an example of an UPnP Basic Device description with an extension
1512 defining a NEMO Node with two roles and sub-services (X_MarlinDeviceInfo):
1513

```
<?xml version="1.0"?>
<root xmlns="urn:schemas-upnp-org:device-1-0">
  <specVersion>
    <major>1</major>
    <minor>0</minor>
  </specVersion>
  <URLBase>/</URLBase>
  <device>
    <deviceType>urn:schemas-upnp-org:device:Basic:1</deviceType>
    <friendlyName>Marlin Rendering Device</friendlyName>
    <manufacturer>InterTrust Technologies</manufacturer>
    <manufacturerURL>http://www.intertrust.com</manufacturerURL>
    <modelDescription>Marlin Rendering Device 1.0</modelDescription>
    <modelName>MRD</modelName>
    <modelNumber>1.0</modelNumber>
    <modelURL>http://www.intertrust.com/MRD/</modelURL>
    <serialNumber>123123123123123</serialNumber>
    <UDN>uuid:UUID</UDN>
  </device>
  <X_MarlinDeviceInfo xmlns="urn:marlin:core:1-3:schemas"
     xmlns:wsp= "xmlns:wsp=http://schemas.xmlsoap.org/ws/2004/09/policy"
     xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance">
    <InspectionServiceLocation binding="urn:marlin:core:1-0:services:ws-
mex:binding:http:soap11" policyId="urn:marlin:core:1-0:services:ws-mex:policy:0">
    http://192.168.1.23:8080/ws-mex
    </InspectionServiceLocation>
    <NemoNodeInfo nodeID="urn:marlin:nemo:node:2344">
      <NemoRoleInfo roleURI="urn:marlin:core:role:drm-client">
        <PortType>proximity-check</PortType>
        <PortType>license-transfer</PortType>
        <PortType>drm-client-information</PortType>
      </NemoRoleInfo>
      <NemoRoleInfo roleURI="urn:marlin:core:role:security-data-provider">
        <PortType>provide-security-data</PortType>
        <PortType>EventSource</PortType>
        <PortType>SubscriptionManager</PortType>
      </NemoRoleInfo>
    </NemoNodeInfo>
  </X_MarlinDeviceInfo>
</root>
```

1514
1515 An UPnP Device description indicating a device that implements Marlin SHALL include a
1516 X_MarlinDeviceInfo element (see marlin-core.xsd) and SHALL contain at least the
1517 following elements: InspectionServiceLocation and NemoNodeInfo.
1518

The contents of X_MarlinDeviceInfo SHALL NOT contain <NemoRoleInfo> elements for services that are not yet operational. For example, a device which is functionally capable of acting fulfilling the domain information provider but has yet to be configured would not advertise this until provisioned. Once the device has been provisioned and ready to fulfill the role then it should advertise the availability of the service.

5.5 Inspection

5.5.1 Overview

Following the discovery phase, the client, SHALL have access to the X_MarlinDeviceInfo structure which SHALL supply all the information needed to invoke the inspection service.

This specification, defines static values for the protocol binding and the policyId attributes of the <InspectionServiceLocation> element:

Attribute	Identifier
binding	urn:marlin:core:1-0:services:ws-mex:binding:http:soap11
policyId	urn:marlin:core:1-0:services:ws-mex:policy:0

The URI, urn:marlin:core:1-0:services:ws-mex:policy:0, maps to “no security” (plain ws-metadata exchange request/response).

These parameters SHOULD be used to invoke an inspection service using [WS-MEX]. The inspection service allows the client to get more information about the discovered service.

5.5.2 Inspection Client and Service interaction

5.5.2.1 Supported Dialects

Marlin implementation of the inspection service SHOULD support the following dialect defined in Appendix I of the [WS-MEX]:

1. WSDL 1.1

Additionally, the NEMO Node Info dialect as defined in [NEMO] §6 SHALL be supported. The namespace URI is:

http://nemo.intertrust.com/2004/inspection/mex/nemonodeinfo

5.5.2.2 GetMetadata request

The GetMetadata request message SHALL conform to [WS-MEX] §3.1 with the following parameters:

/s:Header/wsa:ReplyTo
This field SHALL carry the endpoint of the inspection client of the requesting NEMO Node.

/s:Header/wsa:To
This field SHALL carry the endpoint address of the inspection service extracted from the <X_MarlinDeviceInfo> element.

/s:Body/wsx:GetMetadata/wsx:Identifier
This field SHALL carry the scope of the inspection request as defined below for each dialect.

5.5.2.3 GetMetadata response

Depending on the size of the metadata to return, the service MAY respond with a wsx:MetadataReference to the metadata. The client SHOULD follow-up this response with a second phase where it obtains the metadata. When using this option, the metadata SHALL be obtained using the HTTP-Get method [RFC2616].

The server MAY return a number of ws:Metadata Sections depending on the dialect and the wsa:To field of the request.

5.5.2.3.1 WSDL 1.1 dialect

wsx:Identifier	What should be returned
<NEMO Node ID>	All the WSDLs under the NEMO Node scope
<NEMO Node ID>::<NEMO Role>	All the WSDLs of the services belonging to this role under the NEMO Node scope
<NEMO Node ID>::<NEMO Role>::<WSDL port type>	The WSDL of the service uniquely identified within the scope of the NEMO Node and the indicated role.

5.5.2.3.2 NEMO Node Information dialect

All the wsx:Identifier contain a node ID so the response SHALL always contain the NEMO Node information for the specified NEMO Node ID.

This response SHALL carry all the security credentials in a <nemoc:NodeInfo> element. This includes:

- The signing and encryption NEMO keys
- All the asserted Role attributes for this NEMO Node
- The asserted Specification Version attributes as defined in Section 8.2.

The credentials delivered in the <nemoc:NodeInfo> element MUST adhere to the following:

1. The <wsse:SecruityTokenReference> elements SHALL signal the credential usage with a nemosec:TargetUsage attribute. The nemosec:Usage attribute SHALL NOT be present.
2. The credentials and values for the nemosec:TargetUsage attribute are as follows:
   - Signing key for response messages SHALL be signaled with: http://nemo.intertrust.com/2005/10/security/secure-protocol/basic/1.0#response-signingKey

     The <wsse:SecurityTokenReference> element SHALL contain an <wsse:Embeded> element. The <wsse:Embedded> element SHALL include a <wsse:BinarySecurityToken>. The <wsse:BinarySecurityToken> element SHALL include a ValueType attribute with the value: http://docs.oasis-open.org/wss/2004/01/oasis-200401-wss-x509-token-

profile-1.0#X509PKIPathv1

- Encryption key for request messages SHALL be signaled with: http://nemo.intertrust.com/2005/10/security/secure-protocol/basic/1.0#request-encryptionKey

  The <wsse:SecurityTokenReference> element SHALL contain an <wsse:Embeded> element. The <wsse:Embedded> element SHALL include a <wsse:BinarySecurityToken>. The <wsse:BinarySecurityToken> element SHALL include a ValueType attribute with the value: http://docs.oasis-open.org/wss/2004/01/oasis-200401-wss-x509-token-profile-1.0#X509PKIPathv1

- SAML Assertion with role attributes SHALL be signaled with: http://nemo.intertrust.com/2004/attribute/role

  The <wsse:SecurityTokenReference> element SHALL contain an <wsse:Embeded> element. The <wsse:Embedded> element SHALL include a <saml:Assertion> element (with role attributes.)

3. When one key is used for both signing response messages and encrypting request messages then there SHALL be two <wsse:SecurityTokenReference> elements, one indicating the response signing target usage and the other indicating the request encryption target usage. In other words, these two <wsse:SecurityTokenReference> elements include the same X.509 key with different target usages. Example 5-1 demonstrates this.

```
<nemoc:NodeInfo>
  <nemoc:NodeId>urn:marlin:nemo:node:2344</nemoc:NodeId>
  <!-- DRM Client's Encryption Key -->
  <wsse:SecurityTokenReference
       nemosec:TargetUsage="http://nemo.intertrust.com/2005/10/security/secure-
protocol/basic/1.0#request-encryptionKey">
    <wsse:Embedded>
      <wsse:BinarySecurityToken wsu:Id="PKIPath0"
           ValueType="http://docs.oasis-open.org/wss/2004/01/oasis-200401-wss-x509-
token-profile-1.0#X509PKIPathv1"
           EncodingType="http://docs.oasis-open.org/wss/2004/01/oasis-200401-wss-soap-
message-security-1.0#Base64Binary">
        MIAwggM0MIICHKADAgECAgFlMA0GCSqGSIb3DQEBBQUAMCExHzA...
      </wsse:BinarySecurityToken>
    </wsse:Embedded>
  </wsse:SecurityTokenReference>
  <!-- DRM Client's Signing Key -->
  <wsse:SecurityTokenReference
       nemosec:TargetUsage="http://nemo.intertrust.com/2005/10/security/secure-
protocol/basic/1.0#response-signingKey">
    <wsse:Embedded>
      <wsse:BinarySecurityToken wsu:Id="PKIPath1"
           ValueType="http://docs.oasis-open.org/wss/2004/01/oasis-200401-wss-x509-
token-profile-1.0#X509PKIPathv1"
```

```
          EncodingType="http://docs.oasis-open.org/wss/2004/01/oasis-200401-wss-soap-
message-security-1.0#Base64Binary">
        MIAwggM0MIICHKADAgECAgFlMA0GCSqGSIb3DQEBBQUAMCExHzA...
      </wsse:BinarySecurityToken>
    </wsse:Embedded>
  </wsse:SecurityTokenReference>
 ...
</nemoc:NodeInfo>
```

1625 **Example 5-1: NodeInfo indicating the same key is used for signing and encryption.**

1626

5.6 Subscription and Notification

5.6.1 Overview

Simple interaction such as request/response with web services is not enough and not very dynamic. There are some cases where a client wants to subscribe to an event such as “new security data available” and be notified when this event occurs. The solution we propose here is based on, and compatible with [WS-BASENOTE]. This is illustrated below:

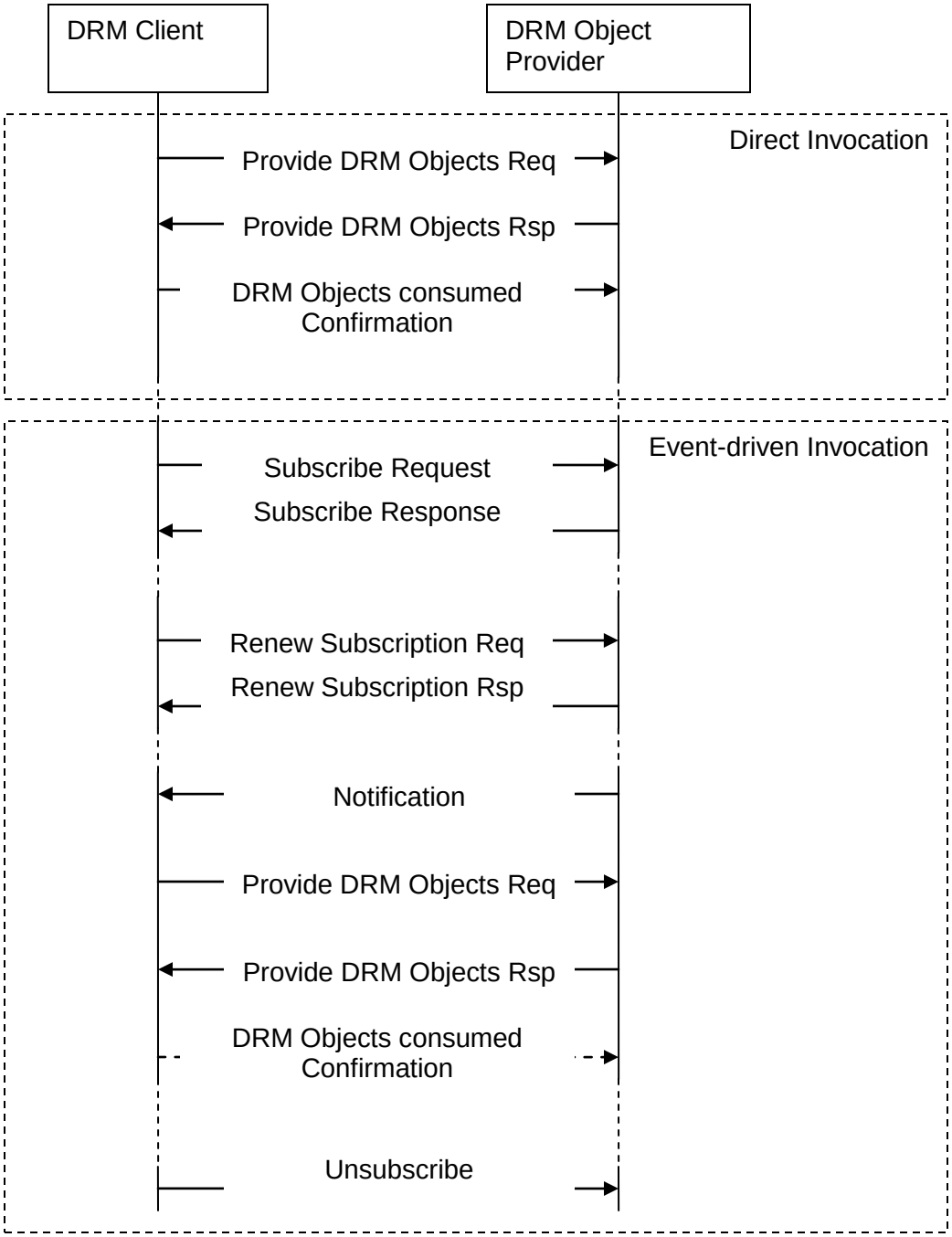


Figure 5-2 Subscription-based Notification Overview

In the first of the following sections, we explain how to subscribe to a topic and how to filter the data the client wants to receive notifications for.

5.6.2 Topics

In order to scope notification subscriptions, all subscription services MUST support the Simple Topic Expression Dialect specified in [WS-TOPICS] §7.1.This dialect is identified by the following URI:

http://docs.oasis-open.org/wsn/t-1/TopicExpression/Simple

This enables a subscriber to choose specific topics for notifications. Optionally, additional restrictions can be specified by qualifying the subscription with an expression which describes a particular system entity (e.g., Octopus Personality Node or Nemo Node.)

5.6.2.1 Provide Security Data Topics

This topic enables a subscriber to receive notifications for security metadata updates. The scope of the subscription MAY be narrowed by the root topic identifiers specified in the following table.

Topic Namespace Identifier	Root Topics	Semantic
urn:marlin:core:1-0:subscription:topic	security-data	Any security metadata
	crl	Newest CRL[4]
	broadcastkeyblock	Newest BKB[4]
	assertion	SAML Assertions
	trustedtime	Trusted Time

Table 5-3 Provide Security Data Root Topic Namespaces

The subscriber MUST specify the scope of the subscription by qualifying the <wsnt:TopicExpression> element with one of the root topics defined in Table 5-3.

Device implementations are REQUIRED to subscribe to the “crl” topic from an available SDP. Implementations with Internet connectivity are REQUIRED to resolve the CRL distribution point URI.

Device implementations that issues licenses are REQUIRED to subscribe to the “broadcastkeyblock” topic. Implementations with Internet connectivity are RECOMMENDED to acquire the BKB by resolving the distribution attributes in the BKB.

The following is an example of a request to subscribe to the “crl” root topic:

```
<wsnt:Subscribe>
  <wsnt:ConsumerReference>
    <wsa:Address>...</wsa:Address>
    <wsa:Metadata xsi:type=”mc:NemoNodeInfo”>
      <mc:NemoNodeInfo nodeID=”urn:marlin:nemo:node:2344”/>
    </wsa:Metadata>
  </wsnt:ConsumerReference>
  <wsnt:Filter>
    <wsnt:TopicExpression
        Dialect=” http://docs.oasis-open.org/wsn/t-1/TopicExpression/Simple”
```

[4] Certificate Revocation Lists and Broadcast Key Blocks are large data objects which update infrequently. The notification messages for these root topics include specific information to aid the Notification Consumer in determining whether the data object is more current than the object possessed by the Notification Consumer. See Section 5.6.3.1.1.

```
          xmlns:tns="urn:marlin:core:1-0:subscription:topic">
        tns:crl
      </wsnt:TopicExpression>
    </wsnt:Filter>
</wsnt:Subscribe>
```

The subscription request MAY be further qualified in the filter by including a <mc:SecurityMetadataTopicExpression> and including the <mc:NemoNodeId> element. In the absence of this element the subscription service SHOULD limit the information published to that which is safe to broadcast to all subscribers. Otherwise the subscription service must limit the distribution to the NEMO Node ID specified in the <wsa:Metadata> element within the <wsnt:ConsumerReference>.

5.6.2.2 Provide DRM Objects Topic

This topic enables a subscriber to receive notifications for Octopus DRM Objects. The supported root topic identifiers are specified in the following table.

Topic Namespace Identifier	Root Topics	Semantic
urn:marlin:core:1-0:subscription:topic	drm-objects	DRM Objects for the indicated Octopus personality

Table 5-4 Provide DRM Objects Root Topic Namespaces

The subscriber MUST specify the scope of the subscription by qualifying the <wsnt:TopicExpression> element with one of the root topics defined in Table 5-4. The following is an example of a request to subscribe to the "*drm-objects*" root topic:

```
<wsnt:Subscribe>
  <wsnt:ConsumerReference>
    <wsa:Address>...</wsa:Address>
    <wsa:Metadata xsi:type="mc:NemoNodeInfo">
      <mc:NemoNodeInfo nodeID="urn:marlin:nemo:node:2344"/>
    </wsa:Metadata>
  </wsnt:ConsumerReference>
  <wsnt:Filter>
    <wsnt:TopicExpression
        Dialect=" http://docs.oasis-open.org/wsn/t-1/TopicExpression/Simple"
        xmlns:tns=" urn:marlin:core:1-0:subscription:topic">
      tns:drm-objects
    </wsnt:TopicExpression>
    <mc:DRMObjectsTopicExpression>
      <mc:OctopusNodeID>…</mc:OctopusNodeID>
    </mc:DRMObjectsTopicExpression>
  </wsnt:Filter>
</wsnt:Subscribe>
```

The subscription request SHOULD further qualify the filter by including a <mc:DRMObjectsTopicExpression> which designates the Octopus Personality Node of interest in the <mc:OctopusNodeId> element. (Informative Note: This identifier is derived from the oct:uid attribute of the <oct:Node> when the Octopus personality node is encoded in XML.) In the absence of the <mc:DRMObjectsTopicExpression> element the subscription service may utilize the inspection services of the NEMO Node described

in the <wsa:Metadata> element within the <wsnt:ConsumerReference> in order to determine the Octopus node(s).

5.6.3 Notification Consumer

Notification Consumers SHALL implement the Notification Consumer interface as specified in [WS-BASENOTE] §3.

5.6.3.1 Notify

This message exchange MUST be implemented as specified in [WS-BASENOTE] §3. Notification consumers MUST understand Notify messages as specified in [WS-BASENOTE] §3.2.

The payload of the <wsnt:Notify> message is carried within a <wsnt:Message> element. It is RECOMMENDED that the payload carry a <mc:NotificationID> element. The value of this element is opaque to the Notification Consumer and serves as a simple identifier which the subscription resource may utilize to locate the specific data being published.

5.6.3.1.1 Qualified Notification Message

The <wsnt:Notify> generally carries sufficient context for the Notification Consumer to obtain the published data. However, the root topics “crl” and “broadcastkeyblock” defined in Section 5.6.2.1 supply updates to critical security metadata which change infrequently.

To avoid excessive consumption of bandwidth retrieving a large data object which is already known to the Notification Consumer a Notification Producer SHOULD insert information in the payload of the <wsnt:NotificationMessage> to identify the instance of these data objects.

For a CRL the payload should bear a <mc:CRLNumber> element. The value of this element is the integer value of the *id-ce-cRLNumber* extension field of a CRL as defined in [PKIX] §5.2.3. The following example depicts a notify message for a CRL.

```
<wsnt:Notify>
  <wsnt:NotificationMessage>
    <wsnt:SubscriptionReference>
      <wsa:Address>...</wsa:Address>
    </wsnt:SubscriptionReference>
    <wsnt:Topic
        Dialect=” http://docs.oasis-open.org/wsn/t-1/TopicExpression/Simple”
        xmlns:tns=”urn:marlin:core:1-0:subscription:topic”>
      tns:crl
    </wsnt:Topic>
    <wsnt:ProducerReference>
      <wsa:Address>...</wsa:Address>
    </wsnt:ProducerReference>
    <wsnt:Message>
      <mc:NotificationID>…</mc:NotificationID>
      <mc:CRLNumber>9</mc:CRLNumber>
    </wsnt:Message>
  <wsnt:NotificationMessage>
<wsnt:Notify>
```

For a Broadcast Key Block the payload SHOULD include a <mc:BKBRevocationVersion> element. The value of this element is the same as the sf:revocationVersion attribute defined in the [Starfish] schema.

5.6.4 Notification Producer

Marlin Notification producers MUST implement the Notification Producer interface as specified in [WS-BASENOTE] §4. Each Notification Producer endpoint MUST have a unique URI.

The following sections define the message exchanges utilized by Marlin entities to leverage a Notification Producer interface.

5.6.4.1 Subscribe

This message exchange MUST be implemented as specified in [WS-BASENOTE] §4.2. In addition to the above requirement the following components MUST be included in a subscription request message.

The mandatory <wsnt:ConsumerReference> element MUST indicate the NEMO Node ID of the notification consumer within a <wsa:Metadata> element. This MUST be accomplished by inserting a <mc:NemoNodeInfo> element as a child of the <wsa:Metadata> element. This element only need to supply the mc:nodeID attribute. Instance of a <wsa:Metadata> element indicating the NEMO Node ID information SHALL assert its element type by specifying the xsi:type attribute with a value of “mc:NemoNodeInfo” string identifier.

The subscription request MUST include a <wsnt:Filter> element. This element MUST carry a <wsnt:TopicExpression> element. The <wsnt:TopicExpression> element MUST specify the simple dialect attribute. The root topics supported by this specification are described in Section 5.6.2.1 and Section 5.6.2.2.

5.6.5 Subscription Manager Operations

Subscription management in Marlin MUST implement the Base Subscription Manager interface as specified in [WS-BASENOTE] §6 and §6.1.

The web services representing individual subscription resources SHALL reside at the same location as the Notification Producer service. That is, individual subscriptions SHALL be represented by endpoint references composed of the information matching the endpoint reference of the subscription manager service. The [WS-Addr] <wsa:Action> message address property helps to disambiguate the service consumer intentions.

The following sections define the interfaces that MUST be exposed by a Marlin Subscription Manager service.

5.6.5.1 Renew

This message exchange MUST be implemented as specified in [WS-BASENOTE] §6.1.1

5.6.5.2 Unsubscribe

This message exchange MUST be implemented as specified in [WS-BASENOTE] §6.1.2.

5.6.6 Faults

The SOAP faults used in context of the notification framework comply with the specifications of the respective message exchanges. As such, they extend <wsnt:BaseFault> as specified in [WSRF-BF] and have the <wsa:Action> URI:

http://docs.oasis-open.org/wsrf/fault

5.7 Service-specific Protocols

5.7.1 Proximity Check Protocol (HARPOON)

5.7.1.1 Overview

The proximity check SHALL allow an anchor (client) to check the proximity of a target (service).

This protocol is asymmetric as the anchor (client) generates a secret seed and is the only one that requires a secure timer. Moreover, the target (service) does not need to trust the anchor (client). It is also cryptographically efficient requiring only two public key operations.

The XML schema for this protocol is defined in the XML Namespace

urn:marlin:core:1-1:nemo:services:schemas

A copy of the XML schema and WSDL is in Appendix A.1 and B.1, respectively.

5.7.1.2 Generation of the set R of Q pairs from a Seed S

The set R is obtained from randomly generated seed using the following method:
$R_i = H^{2Q-i}(S)$

H(M) is the digest value of the hash function H over the message M.
$H^n(M) = H(H^{n-1}(M))$ for n>=1 and $H^0(M) = M$

The algorithm used for the hash function H() SHALL be [SHA1].

5.7.1.3 Sequence

a) A generates a set R of Q pairs of random numbers $\{R_0, R_1\}$, $\{R_2, R_3\}$… $\{R_{2Q-2}, R_{2Q-1}\}$.

b) A sends to B: E(PubB, {Q,S}).

c) B decrypts {Q,S} and precomputes R according to Section 5.7.1.2

d) B sends back an acknowledgement to indicate that it is ready to proceed

e) A sets the loop counter i=0

f) A measures T0 = now

g) A sends to B: {i, R_{2*i}}

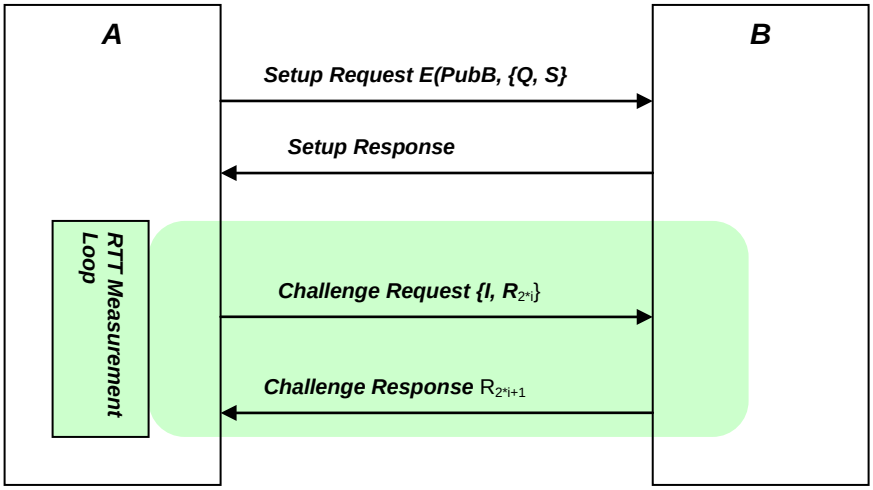
h) If the value of R_{2*I} is correct, B responds with R_{2*i+1}

i) A measures D = now-T0

j) If B is responded to with the correct value for $R_{2*i+1,}$ and D < Threshold: success, stop.

k) If i+1 < Q, A can retry a new measurement by incrementing i and going to step f)

If it is needed to perform more than Q measurements, A needs to start from step a) with a new set R.



5.7.1.4 Security Considerations[5]

When engaging in this protocol, care must be taken to follow some basic requirements:

- The loop f-i MUST NOT be repeated with the same value of i for any set R
- The protocol MUST be aborted if any unexpected message is received by either party. That includes:
  - If B receives an incorrect value for R_{2*i} in step g.
  - If Q is not within a specified range in step a
  - If i is repeated in the loop

[5] It is the purview of a compliance regime to define the values for Q and Threshold. Therefore, implementations should be flexible in configuring these values. Also it is assumed that implementation should support a minimum value of 64 for Q and 8ms for the Threshold.

- If i exceeds Q

5.7.1.5 Messages

5.7.1.5.1 Setup Request

- *E(PubB, {Q, S}):* The number of pairs Q as well as the secret pairs seed S encrypted with B's NEMO public encryption key

{Q, S} is the byte stream concatenation of Q (1 byte) and S (16 bytes) in network byte order.

The encryption is performed with [RSA-1_5]. PubB has been previously accessed through inspection and its certificate MUST have been verified.

5.7.1.5.2 Setup Response

None

5.7.1.5.3 Challenge Request

- *[i, R_{2*i}]*: The index i and the corresponding secret computed from the seed.

[i, R_{2*i}] is the byte stream concatenation of i (1 byte) and R_{2*i} (20 bytes) in network byte order, encoded in base64 for transport.

5.7.1.5.4 Challenge Response

- *R_{2*i+1}:* the corresponding secret from the Challenge Request

R_{2*i+1} is the byte stream of R_{2*i+1} (20 bytes) in network byte order, encoded in base64 for transport.

5.7.1.6 Protocol Security Policies

'No Security' policy is needed for this protocol. The identifier for this policy is:
urn:marlin:core:1.0:nemo:services:proximity-check:policy:0

5.7.2 DRM Client Information

5.7.2.1 Overview

This service can be used to get metadata on the DRM client (second level of inspection). This service includes four operations:

- get the Octopus Node of the DRM Client
- get the current Security Metadata of the client so that a decision can be made by a Security Data Provider to update it or not.
- get the active Octopus links of a DRM Client
- get the secret scuba sharing key of a client

The XML schema for this protocol is defined in the XML Namespace

urn:marlin:core:1-1:nemo:services:schemas

A copy of the XML schema and WSDL is in Appendix A.2 and B.2, respectively.

5.7.2.2 Get Octopus Node

5.7.2.2.1 Request Parameters

None

5.7.2.2.2 Response Data

- *Octopus Bundle:* the data structure containing the public Octopus Personality Node representing the DRM Client

5.7.2.2.3 Protocol Security Policies

The response must be signed by the NEMO authentication key of the DRM Client. This creates a secure binding between the NEMO Node ID of the DRM Client and the Octopus Node it hosts.

The request SHOULD obey 'No Security' policy (Section 5.2.3.1) and the response MUST obey the 'Integrity Only' policy (Section 5.2.4.2.)

The identifier for this policy is:
urn:marlin:core:1.0:nemo:services:drm-client-information:get-octopus-node:policy:0

5.7.2.3 Get Security Metadata

5.7.2.3.1 Request Parameters

None

5.7.2.3.2 Response Data

- *Security Metadata*: The opaque data structure representing the current security metadata of the DRM Client

5.7.2.3.3 Protocol Security Policies

The response must be signed by the NEMO authentication key of the DRM Client. This allows for the authenticity of the security metadata to be verified.

Because the response requires freshness the request SHOULD obey the 'Freshness' only policy (Section 5.2.3.2) and the response MUST obey the 'Integrity+Freshness' policy (Section 5.2.4.3.)

The identifier for this policy is:
urn:marlin:core:1-2:nemo:services:drm-client-information:get-security-metadata:policy:0

5.7.2.4 Get Domain Links

5.7.2.4.1 Request Parameters

- domain-policy: This OPTIONAL element specifies the URI used to identify the domain policy.

5.7.2.4.2 Response Data

- *Bundle*: structure carrying the current valid Octopus Links of the DRM Client

5.7.2.4.3 Protocol Security Policies

The response must be signed by the NEMO authentication key of the DRM Client. This allows for the authenticity of the links validity to be verified.

Because the response requires freshness the request SHOULD obey the 'Freshness' only policy (Section 5.2.3.2) and the response MUST obey the 'Integrity+Freshness' policy (Section 5.2.4.3.)

The identifier for this policy is:

urn:marlin:core:1-2:nemo:services:drm-client-information:get-domain-links:policy:0

5.7.2.5 Get Scuba Secret Sharing Key

5.7.2.5.1 Request Parameters

None

5.7.2.5.2 Response Data

- *Key Data*: structure carrying the base 64 encoded plain Scuba Sharing key of the DRM Client.

5.7.2.5.3 Protocol Security Policies

The request must be signed and the DRM Object Provider role of the requesting NEMO Node MUST be attached to the request.

The response must be signed by the NEMO authentication key and encrypted by the requestor's public key. This enables the recipient to verify the authenticity of the links validity and protects the confidentiality of the Scuba Secret Sharing key. It is OPTIONAL for a client to distribute its Scuba Secret Sharing key.

The request MUST obey the 'Integrity+Freshness' policy (Section 5.2.3.4) and the response MUST obey the response 'Full Security' policy (Section 5.2.3.5.)

The identifier for this policy is:

urn:marlin:core:1.0:nemo:services:drm-client-information:get-scuba-sharing-key:policy:0

5.7.3 Provide Domain Information

5.7.3.1 Overview

This service can be used to get metadata regarding the Domains that the Domain Information Provider has access to.
The XML schema for this protocol is defined in the XML Namespace:

urn:marlin:core:1-1:nemo:services:schemas

A copy of the XML schema and WSDL is in Appendix A.3 and B.3, respectively.

5.7.3.2 Request Parameters

None

5.7.3.3 Response Data

- *Domain Info (one or more):* Data structure encoding the ID of the Octopus Node representing the domain as well as the policy type for this domain and the Provide DRM Objects service endpoint that can be invoked to request a registration.
- *Domain Policy (one or more):* the data structure encoding the policy referred to in the Domain Info structure. This structure SHALL be used for User Interface purposes only.

5.7.3.4 Protocol Security Policies

The response must be signed by the NEMO Authentication key of the Domain Manager. This allows for the authenticity of the Domain Manager Information to be verified.

The request SHOULD obey the 'No Security' policy (Section 5.2.3.1) and the response MUST obey the 'Integrity Only' policy (Section 5.2.3.2.)

The identifier for this policy is:

urn:marlin:core:1.0:nemo:services:provide-domain-information:policy:0

5.7.4 Provide DRM Objects

5.7.4.1 Overview

This is a simple REQUEST/RESPONSE protocol. However, the exchange anticipates the response to bear an Agent. Agents are obligated to supply a confirmation. Confirming to the DRM Object providing service allows the service to determine that the object has been processed in a secure environment.

The XML schema for this protocol is defined in the XML Namespace

urn:marlin:core:1-1:nemo:services:schemas

A copy of the XML schema and WSDL is in Appendix A.4 and B.4, respectively.

5.7.4.2 Request parameters

- *Octopus Personality Node:* The Octopus Node representing the DRM Client.
- *Notification ID:* This OPTIONAL element is present if the request is sent after the DRM Client has been notified (event driven invocation), then this parameter MUST be present so that the service can correlate it with the notification it sent to the DRM Client.
- *DRMObjectContext:* This OPTIONAL element supplies the data structure that encodes the action context of the provisioning of this Octopus Object. It will indicate the action, if any, intended by the requester (i.e. registration or deregistration).

An example of a DRMObjectContext follows:

```
<mncs:DRMObjectContext>
  <mncs:DRMAction
name="urn:marlin:core:1.0:nemo:services:provide-drm-
objects:context:action:registration"/>
  <ml:ContextList>
    <ml:Context uid="urn:marlin:broadcast:1-0:domain:243523"/>
  </ml:ContextList>
</mncs:DRMObjectContext>
```

Therefore, when the service receives a Provide DRM Object request with this data structure, it should try to register the client with the domain urn:marlin:broadcast:1-0:domain:243523

The mncs:name attribute of the <mncs:DRMAction> element may have one of the following values:

urn:marlin:core:1.0:nemo:services:provide-drm-objects:context:action:registration
urn:marlin:core:1.0:nemo:services:provide-drm-objects:context:action:deregistration

5.7.4.3 Response Data

- *Octopus Bundle:* This OPTIONAL element supplies the data structure containing the Octopus objects. This bundle SHALL NOT bear Agents.
- *Agent Carrier:* This OPTIONAL element supplies the data structure containing the Agent as well as the input parameters and the context ID. The name and controlID attributes MUST be specified to unambiguously identify the Agent.
- *DRMObjectContext:* This OPTIONAL element supplies the data structure that encodes the action context for provisioning of the Octopus Object. This data structure also carries the context identifiers that the application SHOULD use in order to do the necessary cleanup. This element SHALL be present when the action context is related to domain registration or de-registration.

An example of a DRMObjectContext follows:

```
<mncs:DRMObjectContext>
  <mncs:DRMAction
name="urn:marlin:core:1.0:nemo:services:provide-drm-
objects:context:action:deregistration"/>
  <ml:ContextList>
    <ml:Context uid="urn:marlin:broadcast:1-0:domain:243523"/>
```

```
  </ml:ContextList>
</mncs:DRMObjectContext>
```

Therefore, when the client receives Provide DRM Object response with this data structure, it should cleanup all the Octopus links that have a domain-id attribute equal to "urn:marlin:broadcast:1-0:domain:243523".

5.7.4.4 Confirmation parameters

A confirmation MUST be sent. This is necessary regardless of whether the response contains an Agent Carrier or not. By making the confirmation mandatory we ensure that the communicating entities can be prepared for a multi-round message exchange pattern.

The DRM Client may also refuse the objects provided by the service. In this case, it will send a DRM Object Refusal in the confirmation.

- DRMObjectRefusal: This OPTIONAL element contains the DRMObjectContext that was sent in the response.
- *Agent Result Block:* This OPTIONAL element represents the data resulting from processing the Agent.

5.7.4.5 Notification

If a DRM Client has subscribed to the topic space defined in Section 5.6.2.2, the Provide DRM Objects service (event source) notifies the DRM Client when it has new DRM Objects for it. The payload of the notification SHALL be:

- *Notification ID:* The ID of the notification

The notification signals to the DRM Client that it MUST contact the Provide DRM Objects service to get the object(s).

5.7.4.6 Protocol Security Policies

The request and confirmation MUST obey the 'Integrity+Freshness' policy (Section 5.2.3.4) and the response MUST obey the 'Integrity+Freshness' policy (Section 5.2.4.3.)

Moreover, in order to correlate the request with the confirmation messages, the Message Correlation pattern described in [NEMO] §2.3 MUST be used. The specific information in the SOAP header guarantying the correlation MUST be covered by the message signature.

The identifier for this policy is:

```
urn:marlin:core:1.0:nemo:services:provide-drm-objects:obtain-objects:policy:0
```

5.7.5 Provide Security Data

5.7.5.1 Overview

This is a simple REQUEST/RESPONSE protocol.

The XML schema for this protocol is defined in the XML Namespace

2090
2091 urn:marlin:core:1-1:nemo:services:schemas
2092
2093 A copy of the XML schema and WSDL is in Appendix A.5 and B.5, respectively.
2094

2095 #### 5.7.5.2 Request parameters

2096 • *Security Metadata:* The security metadata possessed by the Device which the
2097 Service may inspect for currency.
2098 • *Notification ID:* This OPTIONAL element is present if the request is sent after the
2099 Device has been notified (event driven invocation), then this parameter MUST be
2100 present so that the service can correlate it with the notification it sent to the
2101 Device.

2102 #### 5.7.5.3 Response Data

2103 • *Security Metadata:* The data to be processed by the Device
2104

2105 #### 5.7.5.4 Notification

2106 If a Device has subscribed to the topic space defined in Section 5.6.2.1, the Provide
2107 Security Data service (event source) notifies the Device when it has new Security Data
2108 for it. The payload of the notification SHALL be:
2109 • N*otification ID:* The ID of the notification
2110
2111 The notification signals to the Device that it MUST contact the Provide Security Data
2112 service to get the update.

2113 #### 5.7.5.5 Protocol Security Policies

2114 Because the response requires freshness the request SHOULD obey the 'Freshness'
2115 only policy (Section 5.2.3.2) and the response MUST obey the 'Integrity+Freshness'
2116 policy (Section 5.2.4.3.)
2117
2118 The identifier for this policy is:

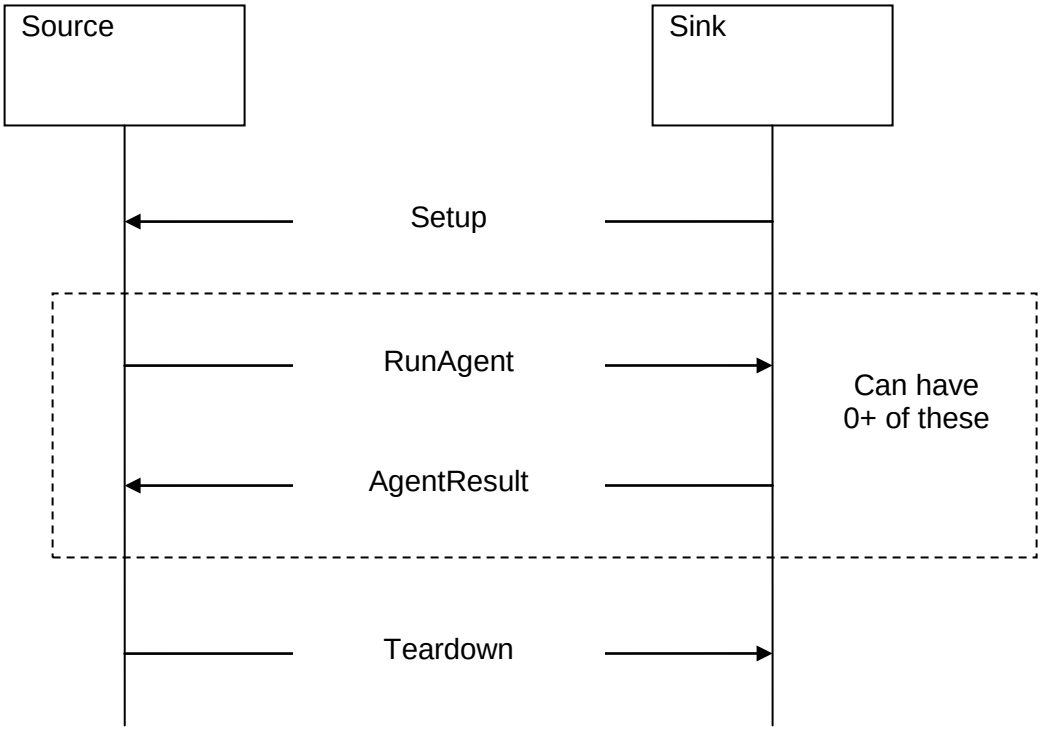
urn:marlin:core:1-2:nemo:services:security-data:obtain-data:policy:0

2119 5.7.6 License Transfer

2120 #### 5.7.6.1 Overview

2121 In order to ensure consistent behaviour among implementations some of the service
2122 interactions must uphold usage rules. For the License Transfer service the usage rule is
2123 defined in Section 11. This protocol specifies the service interaction used to support this
2124 usage rule.
2125
2126 The XML schema for this protocol is defined in the XML Namespace
2127
2128 urn:marlin:core:1-1:nemo:services:schemas
2129
2130 A copy of the XML schema and WSDL is in Appendix A.6 and B.6, respectively.

The figure below indicates the message flow for this protocol.



5.7.6.2 Setup Message Parameters

- *License:* Data structure that carries the license to be transfered.
- *Operation:* The type of transfer operation.
- *Octopus Bundle:* the data structure containing the public Octopus Personality Node representing the DRM Client.
- *SessionId:* OPTIONAL element. It SHALL NOT be present except if the operation is urn:marlin:core:1-3:service:license-transfer:release in which case it MAY be present if a SessionId parameter was received in the Extended Status Block fo the corresponding render or checkout action.
- *RequireContentKeys:* OPTIONAL element. It SHOULD be present with a value of false if the sink knows that it is already capable of decrypting the content keys. The absence of this element means that the Source has to re-encrypt the Content Keys of the sink in case of success of the protocol.

The types of operation are defined in the following table:

urn:marlin:core:1-0:service:license-transfer:move
urn:marlin:core:1-0:service:license-transfer:copy
urn:marlin:core:1-0:service:license-transfer:render
urn:marlin:core:1-0:service:license-transfer:checkout
urn:marlin:core:1-3:service:license-transfer:release

In the case of streaming, the identifier

urn:marlin:core:1-0:service:license-transfer:render

is used ([8pus] §3).

When the Setup Message is received, the Source SHALL do the following:

1. Collect all the contentId attributes found in the LicensePart elements (see XML schema).
    - When a contentId attribute in a LicensePart element is omitted, all of the contentId(s) in Bundle element (LicensePart/Bundle) are target for LTP transaction.
    - When a contentId attribute in a LicensePart element is specified, the corresponding contentId in Bundle element (LicensePart/Bundle) is a target for LTP transaction.
2. Process all the Bundle elements found in the LicensePart elements
3. Open the set of content IDs collected above
4. Verify the appropriate signatures on the relevant objects
5. Populate the relevant context for the Control.Actions.Transfer method
6. Optionally invoke the Control.Actions.Transfer.Check method on the processed Control object.
7. Invoke the Control.Actions.Transfer.Perform on the process Control object.
8. Read the Extended Status Block (ESB) returned by the Control.Actions.Transfer.Perform entry point.
    - If this ESB contains a RunAgentOnPeer obligation / OnAgentCompletion callback (see [8pus] §3), the RunAgentOnPeer obligation SHALL contain all the parameters that the Source application needs in order to build the RunAgent message. Note that this message will also be sent if the application encounters another RunAgentOnPeer/OnAgentCompletion callback/obligation pair in the Extended Status Block of the OnAgentCompletion callback (after one or more RunAgent/AgentResult message exchange).
    - If this ESB does not contain a RunAgentOnPeer obligation / OnAgentCompletion callback, it means that the Teardown message (see section 3.4) MUST be sent. Note that this ESB MAY contain a ProximityCheck obligation / OnSinkProximityChecked callback in which case the harpoon protocol SHALL be performed and the result will be read from the ESB of the OnSinkProximity checked callback before sending the Teardown message.

5.7.6.3 RunAgent Message Parameters

- *Agent Carrier:* The data structure containing the Agent as well as the input parameters and the context ID. The name and controlID attributes MUST be specified to unambiguously identify the Agent.

When the Sink receives the agent it MUST run it as specified in section 3.2.8 of the [8pus] specifications.

5.7.6.4 AgentResult Message Parameters

- *Agent Result Block:* The data resulting from the processing of the Agent.

5.7.6.5 Teardown Message Parameters

- *Extended Status Block:* This is the ESB of the last invocation of the Control that contains no agent related obligation / callback pair. It contains the protocol result so that the sink knows if the protocol has succeeded or not. In case of failure, this ESB MAY point to resources located in the ResourceList extension of the Control that was sent in the Setup message.
- *Content Key List:* OPTIONAL element. If the protocol is not successful as indicated by the Extended Status Block above, this element SHALL NOT be sent.

NOTE: In the case of the urn:marlin:core:1-0:service:license-transfer:render operation, the sink MUST use the ESB in the Teardown message as is and MUST NOT locally re-evaluate the Control for the Control.Actions.Play action. For all other operations, the Control.Actions.Play action MUST be performed before playing the content.

NOTE: in the case of the urn:marlin:core:1-0:service:license-transfer:render and urn:marlin:core:1-0:service:license-transfer:checkout actions, the ESB MAY contain a SessionId critical parameter (see [8pus] §3) that the sink MUST send back when doing a urn:marlin:core:1-3:service:license-transfer:release on the same piece of content.

5.7.6.6 Protocol Security Policies

The Setup and AgentResult messages MUST obey the 'Integrity+Freshness' policy (Section 5.2.3.4.) The RunAgent and Teardown messages MUST obey the 'Integrity+Freshness' policy (Section 5.2.4.3.)

Moreover, in order to correlate the messages, the Message Correlation pattern described in [NEMO] §2.3 MUST be used. The specific information in the SOAP header guarantying the correlation MUST be covered by the message signature.

The identifier for this policy is:

urn:marlin:core:1.3:nemo:services:license-transfer:policy:0

6 Marlin Protocol Bindings

Except where noted, Marlin Protocols SHALL use the Nemo SOAP/HTTP Message Bindings.

6.1 HTTP/OBEX Binding

The purpose of this section is to offer a binding of HTTP over a non-IP stack to support directly attached devices. When using OBEX as a transport over a non-IP stack this binding is REQUIRED.

This binding supports the conveyance of HTTP messages over an OBEX transport. The general notion behind this binding is to map the HTTP-message as defined in [RFC2616] §4 and §5 to objects in the OBEX Object Model [OBEX13] §2. In so doing, this binding is a generic HTTP binding and could be used to exchange any content that the HTTP protocol supports.

The following identifiers are defined to support this binding.

Identifier	Type	Value	Description
HOB_UUID	UUID	0ba38ad4-018b-3b48-b0ad-14946ef90493	Target header Connection Id to connect to the service supporting this binding
HOB_INSPECT	URI	/inspect.xml	An absolute-path reference form of a relativeURI [RFC2396] §5 to the device inspection document. For a device supporting the Marlin inspection service the result of dereferencing this URI would be an XML document containing a <mc:X_MarlinDeviceInfo> element.
HOB_SCHEME	URI scheme	httpobex	A protocol scheme identifier used as the prefix to URLs.

To further elaborate on the construction of URLs a client using this binding would make http-like requests to the OBEX connected server using the following static syntax:

httpobex://0ba38ad4-018b-3b48-b0ad-4946ef90493/

For example, to retrieve the well-known inspection service document (HOB_INSPECT) the URL would be:

httpobex://0ba38ad4-018b-3b48-b0ad-4946ef90493/inspect.xml

The following OBEX Operations are REQUIRED to support this binding.

OBEX Operation	Opcode
CONNECT	0x80
DISCONNECT	0x81

GET	0x03 (0x83)
ABORT	0xFF

6.1.1 Connection Establishment

This binding REQUIRES that a session between the client and server endpoints be established prior to exchanging HTTP messages over the transport. This is accomplished by sending a CONNECT request and the remote device indicates that the connection has been established by returning a successful response.

The OBEX CONNECT request MUST contain the following fields:

- CONNECT Opcode (0x80)
- Packet Length
- OBEX Version Number
- Flags
- Maximum OBEX Packet Length

In addition to the above fields the CONNECT message MUST include a Target header bearing the value of the HOB_UUID.

A successful OBEX CONNECT response MUST contain the following fields:

- OBEX Response code (0xA0)
- Packet Length
- OBEX Version Number
- Flags

In addition to the above fields a successful response message MUST include a Connection ID header. This header MUST be the first header.

The Connection ID is used to supply context in subsequent interactions between the client and server.

6.1.2 Connection Termination

Once a session has been established multiple message exchanges may occur over the connection. Once an application has completed its message exchanges it MUST tear down the session. This is accomplished by sending a DISCONNECT request, and the remote device indicates that the connection has been terminated, by returning a successful response.

The OBEX DISCONNECT request MUST contain the following fields:

- DISCONNECT Opcode (0x81)
- Packet Length

In addition to the above fields the DISCONNECT message MUST include a valid CONNECTION ID header to indicate which session to terminate.

A successful OBEX DISCONNECT response MUST contain the following fields:

- OBEX Response code (0xA0)
- Packet Length

6.1.3 Message Exchange

Following the successful establishment of a connection, HTTP message exchanges may commence.

This binding REQUIRES that the OBEX GET operation be used for all such message exchanges. This binding also REQUIRES that the connection id returned from the connection establishment phase of this binding be included in the Connection ID header of each GET request message.

6.1.4 Aborting a Message Exchange

In the advent that a multi-packet message exchange needs to be abandoned by the client application the OBEX ABORT operation may be used. See [OBEX13] for details.

6.1.5 Mapping HTTP Messages to OBEX

This binding leverages the many similarities between the OBEX protocol and HTTP to define mechanism by which to exchange HTTP messages over OBEX.

The general approach is to map HTTP methods to OBEX operations, HTTP status codes to OBEX response codes, HTTP message header to OBEX HTTP header, and the HTTP message body to OBEX Body (or End-of-Body) header objects.

The following table provides the mapping and the section number within the respective specifications where the mapped entity is defined.

HTTP 1.1 Tokens [RFC2616]	Section	OBEX Field/Object [OBEX13]	Section
Method	§5.1.1	Opcode	§3.3
Status-Code	§6.1	Response code	§3.2
message-header	§4.2	HTTP header	§2.2.8
message-body	§4.3	Body/End-of-Body header	§2.2.9
Content-Type entity-header	§7.2.1	Type header	§2.2.3

6.1.5.1 HTTP Method to OBEX Opcode Mapping

This binding supports all of the HTTP Methods defined in [RFC2616] §9.2-§9.9. All of these methods are mapped onto the OBEX GET operation. The rationale being that the OBEX GET operation is much more flexible than the OBEX PUT in handling variable sized HTTP request and response messages. That is unlike the OBEX GET operation the OBEX PUT operation does not support multi-packet responses.

6.1.5.2 HTTP Status Code to OBEX Response Code Mapping

OBEX natively supports a direct mapping between HTTP status codes and OBEX response codes. This binding REQUIRES the use of the native mapping.

6.1.5.3 HTTP Content-Type entity-header to OBEX Type Header Mapping

To aid in dispatching and processing protocol messages the OBEX Type header SHOULD include the equivalent value as the value conveyed in the HTTP Content-Type entity-header.

6.1.5.4 HTTP message-header to OBEX HTTP Header Mapping

The OBEX HTTP header MUST contain the contents of the HTTP 1.1 message-header.

6.1.5.5 HTTP message-body to OBEX Body or End-of-Body Mapping

The OBEX Body header (or End-of-Body) MUST deliver the payload of the HTTP message. The payload MUST only include the message-body as described in [HTTP] §4.3. For example, when exchanging SOAP messages the HTTP message-body would be consumed by the entire SOAP message (<Envelope>…</Envelope>). Note that the message may be split into smaller messages and transferred in several OBEX GET messages.

6.1.5.6 Example HTTP Message Exchange (Informative)

The following tables give a generalized example of the protocol message flows used to exchange an HTTP request and response, respectively. These examples assume that an OBEX session was previously established.

HTTP Client Request:	**Bytes**	**Description**
Opcode	0x03	GET, Final bit NOT set
	0xnnnn	Length of packet
	0xCB	Connection Id header
	0xnnnn	Connection Id
	0x42	Type header
	0x09	Length of type object
	text/xml	Type of object (NULL terminated ASCII)
	0x47	HTTP header
	0x…	HTTP 1.1 header
	0x48	Body header
	0xnnnn	Length of Body header
	0x…	Body (HTTP message-body)
HTTP Server Response:		
	0x90	CONTINUE, Final bit set
	0x0003	Length of response packet
HTTP Client Request:		
Opcode	0x03	GET, Final bit NOT set
	0xnnnn	Length of packet
	0x47	HTTP header
	0x…	HTTP 1.1 header
	0x48	Body header

	0xnnnn 0x…	Length of Body header Body (continuation of HTTP message-body)
HTTP Server Response:		
	0x90 0x0003	CONTINUE, Final bit set Length of response packet
HTTP Client Request:		
Opcode	0x83 0xnnnn 0x47 0x… 0x49 0xnnnn 0x…	GET, Final bit set Length of packet HTTP header HTTP 1.1 header End-of-Body header Length of End-of-Body header End-of-Body (last part of HTTP message-body)

2351

HTTP Server Response:	Bytes	Description
	0x90 0xnnnn 0x42 0x09 text/xml 0x47 0x… 0x48 0xnnnn 0x…	CONTINUE, Final bit NOT set Length of response packet Type header Length of type object Type of object (NULL terminated ASCII) HTTP header HTTP 1.1 header Body header Length of Body header Body (HTTP message-body)
HTTP Client Request:		
Opcode	0x83 0x0003	GET, Final bit set Length of packet
HTTP Server Response:		
	0xA0 0xnnnn 0x47 0x… 0x49 0xnnnn 0x…	SUCCESS, Final bit set Length of response packet HTTP header HTTP 1.1 header End-of-Body header Length of End-of-Body header End-of-Body (last part of HTTP message-body)

2352

2353 *6.2 SOAP 1.1/HTTP 1.1 Binding (Informative)*

2354 This binding is normatively defined in [SOAP11] 6§. However to aid the reader germane
2355 aspects of the binding are informatively described here.

2356 6.2.1 HTTP Headers

2357 6.2.1.1 HTTP Method

2358 The HTTP request method must be POST as defined in the SOAP HTTP Binding
2359 [SOAP11].

2360 6.2.1.2 Content-Type

2361 The Content-Type header must be ‘text/xml; charset=”utf-8”’ as defined in the SOAP
2362 HTTP Binding [SOAP11].

2363 6.2.1.3 Content-Length

2364 The Content-Length header must be supplied in accordance with HTTP 1.1 [RFC2616].

6.2.1.4 SOAPAction

The SOAP Action header must be supplied to give the intent of the SOAP HTTP request [SOAP11] §6.1.1. However, this is an application specific value and therefore its definition is out of scope for this specification.

6.2.1.5 Caching Policy

It is the intent of this binding to not preclude the use of HTTP proxies. However, a SOAP based application typically does not desire an HTTP intermediary to cache requests and responses.

The following HTTP headers should be supplied to inhibit HTTP proxies from caching protocol messages.

Requesters should include the following HTTP headers:

- Cache-Control: no-cache, no-store
- Pragma: no-cache

Responders should include the following HTTP headers:

- Cache-Control: no-cache, no-store, must-revalidate, private
- Pragma: no-cache

Additionally, it is not recommended to use the HTTP headers defined to support the HTTP 1.1 Validation Model.

6.3 NEMO Message Binding

This binding shall comply with the SOAP Message Binding as defined in [NEMO] §2.

Additionally, the security of message exchanges MUST conform to the NEMO Security Bindings specification [NEMO] §3. Section 9.1.4.4 and section 9.1.4.5 describe signature verification and confidentiality protection of various target data, respectively. The proper application of the verification or encryption MUST align with NEMO usages as defined by [NEMO] §3. Thus, the signing/encryption targets for secure NEMO messaging MUST adhere to the following rules:

1. A Marlin certificate containing a key used in a NEMO message with either of the NEMO usages;

   - http://nemo.intertrust.com/2005/10/security/secure-protocol/basic/1.0#response-signingKey
   - http://nemo.intertrust.com/2005/10/security/secure-protocol/basic/1.0#request-encryptionKey

   MUST contain the certificate policy;
   - id-cp-nemo-marlin-service-key

2. A Marlin certificate containing a key used in a NEMO message with either of the NEMO usages;

   - http://nemo.intertrust.com/2005/10/security/secure-protocol/basic/1.0#request-signingKey
   - http://nemo.intertrust.com/2005/10/security/secure-protocol/basic/1.0#response-encryptionKey

   MUST contain the certificate policy;
   - id-cp-nemo-marlin-client-drm-key

7 Marlin Key Management

7.1 Introduction (Informative)

In the Octopus DRM scheme an encrypted Content Key is delivered in the Octopus License Object. In the Marlin Key Management System, the Content Key may be encrypted with two different sets of keys:

- **Scuba Keys**
  A key management system in which Octopus objects, such as Nodes and Links, carry keys needed to allow users and devices to derive the Content Key to access the protected content.
- **[Starfish] Revocation Keys**
  A type of broadcast encryption that allows a Content Key to be encrypted in such a way that selected clients (for example, those that have been compromised in some way) are no longer able to access them. [Starfish] defines and uses the HBES (Hierarchical Hash-Chain Broadcast Encryption Scheme) key tree for creating and assigning revocation keys.

If no Nodes have yet been revoked, the Content Key is only encrypted with the Scuba Sharing Key. After one or more revocations have taken place, the Content Key is additionally encrypted using the [Starfish] revocation mechanism.

7.2 HBES Broadcast Key Block Validity

[Starfish] defines a binary and XML encoding of the HBES Broadcast Key Block (BKB). Conformant implementations of this specification MUST support the XML BKB encoding defined in [Starfish] §4.3. This specification requires that the BKB revocation information be signed by a trusted authority. The license issuing entity MUST verify the validity and integrity of the distributed BKB. However, a license consuming entity, such as a Octopus DRM Engine, SHOULD NOT verify the signature over the BKB revocation information.

7.3 Content Key Object before Exclusion

The ContentKey Object (<oct:ContentKey> element) contains a <oct:SecretKey> element which represents the actual key used to encrypt the content.

Before the first revocation of a device (or application), the Content Key CK (represented by the <oct:SecretKey> element) SHALL only be encrypted by the Scuba key (public or secret) of the entity the license is targeted to (the user for example) [8pus] §6. This results in the following algorithm:

$$K_{Marlin} = E(K_{scuba}, CK)$$

In this algorithm the following keys are introduced:

K_{scuba} The key distributed by Scuba [SCUBA].
K_{Marlin} The key to be stored in the Content Key Object
CK The key used to encrypt the content.

K_{scuba} is calculated according to [8pus] §6. K_{Marlin} is generated from K_{scuba} and CK.

After encryption with K_{scuba} (public or secret) of the entity the license is targeted to, the resulting encrypted data K_{Marlin} MUST be represented by <oct:SecretKey> in the <oct:ContentKey> element.

7.4 Content Key Object after Exclusion

After the first exclusion, the CK SHALL be encrypted using the [Starfish] revocation mechanism. All DRM Clients SHALL implement the [Starfish] revocation mechanism. The resulting data SHALL then be encrypted with the Scuba key (public or secret) of the entity the license is targeted to. This results in the following algorithm:

$$K_{Marlin} = E(K_{scuba}, (E(K_{root}, CK))$$

In this algorithm the following keys are introduced:

K_{root} The Broadcast Key extracted from the BKB. See [Starfish] for a detailed description of how the Broadcast Key is recovered for a given receiver.

K_{scuba} The key distributed by Scuba Key Distribution ([8pus] §6).

K_{Marlin} The key to be stored in the Content Key Object

CK The key used to encrypt the content.

When exclusion is used, K_{root} is recovered from the BKB. The BKB MUST be carried in the same Bundle as the Content Key Object per Section 3.3.2.

After encryption with K_{root}, the byte sequence of <xenc:EncryptedData> is encrypted with K_{scuba} (public or secret) of the entity the license is targeted to. The resulting super-encrypted data K_{Marlin} MUST be represented by <oct:SecretKey> in <oct:ContentKey>.

Since CK is encrypted by both K_{root} and K_{scuba}, the ContentKey object SHALL follow the super-encryption guidance given in [xmlenc] §2.1.5. The algorithm used to encrypt the CK with K_{root} SHALL be 128 bit AES in CBC mode and the padding SHALL follow the guidance given in [xmlenc] §5.2. The [xmlenc] representation of the CK encrypted with K_{root} is signaled with an <xenc:EncryptionMethod> element Algorthm attribute with a value of:

http://marlin-drm.com/starfish/algorithmID/1.0

2496 The following example depicts the contents of the Octopus Bundle including the Content
2497 Key Object and the exclusion information supplied by the BKB:
2498

```
[01] <oct:Bundle xmlns:oct="http://www.octopus-drm.com/profiles/base/1.0">
[02]
[03]  <!-- ContentKey -->
[04]  <oct:ContentKey uid="urn:foo:bar:content-key:551881195">
[05]   <oct:SecretKey uid="urn:foo:bar:content-key:551881195:secret-key">
[06]    <KeyData encoding="xmlenc" format="RAW">
[07]     <EncryptedData xmlns="http://www.w3.org/2001/04/xmlenc#"
             Type="http://www.w3.org/2001/04/xmlenc#Element">
[08]      <EncryptionMethod
            Algorithm="http://www.w3.org/2001/04/xmlenc#aes128-cbc"/>
[09]      <KeyInfo xmlns="http://www.w3.org/2000/09/xmldsig#">
[10]       <KeyName>urn:marlin:organization:phony:01:secret-sharing</KeyName>
[11]      </KeyInfo>
[12]      <CipherData>
[13]       <CipherValue>K3K9Idxk...MjoyW+w2S9i…=</CipherValue>
[14]      </CipherData>
[15]     </EncryptedData>
[16]    </KeyData>
[17]   </oct:SecretKey>
[18]  </oct:ContentKey>
[19]
[20]  <!-- Other Octopus Objects -->
[21]  <oct:Protector>
[22]   <oct:ContentKeyReference>
[23]    <oct:Uid>urn:foo:bar:content-key:551881195</oct:Uid>
[24]   </oct:ContentKeyReference>
[25]   <oct:ProtectedTargets>
[26]    <oct:ContentReference>
[27]     <oct:Uid>urn:x-octopus.intertrust.com:content:0012</oct:Uid>
[28]    </oct:ContentReference>
[29]   </oct:ProtectedTargets>
[30]  </oct:Protector>
[31]  <oct:Controller Id="controller"
[32]              uid="urn:foo:bar:controller:2031011164">
[33]   <oct:ControlReference>
[34]    <oct:Uid>urn:foo:bar:control:874685522</oct:Uid>
[35]    <oct:Digest>
[36]     <DigestMethod xmlns="http://www.w3.org/2000/09/xmldsig#"
[37]           Algorithm="http://www.w3.org/2000/09/xmldsig#sha1"/>
[38]     <DigestValue xmlns="http://www.w3.org/2000/09/xmldsig#">
[39]      fj9yTLt4b90/SsYQyj8wB2dzRBU=</DigestValue>
[40]    </oct:Digest>
[41]   </oct:ControlReference>
[42]   <oct:ControlledTargets>
[43]    <oct:ContentKeyReference>
[44]     <oct:Uid>urn:foo:bar:content-key:551881195</oct:Uid>
[45]     <oct:Digest>
[46]      <DigestMethod xmlns="http://www.w3.org/2000/09/xmldsig#"
[47]           Algorithm="http://www.w3.org/2000/09/xmldsig#sha1"/>
[48]      <DigestValue xmlns="http://www.w3.org/2000/09/xmldsig#">
[49]       yDOqz1Bs1IqstZpbp94jig3h5kY=</DigestValue>
```

```
[50]        </oct:Digest>
[51]      </oct:ContentKeyReference>
[52]    </oct:ControlledTargets>
[53]  </oct:Controller>
[54]  <oct:Control uid="urn:foo:bar:control:874685522">
[55]    <oct:ControlProgram
[56]        protocol="http://www.octopus-drm.com/specs/scp-1_0">
[57]      <oct:CodeModule
[58]        type="http://www.octopus-drm.com/specs/pkcm-1_0">
[59]       AAABsHB...LkdldExvY2FsVGltZQAAAAAA</oct:CodeModule>
[60]    </oct:ControlProgram>
[61]  </oct:Control>
[62]  ...
[63]  <!-- BroadcastKeyBlock -->
[64]  <sf:BroadcastKeyBlock xmlns:sf="http://marlin-drm.com/starfish/1.2"
[65]                      keyTreeName="urn:marlin:starfish:keytree:1">
[66]    <sf:RevocationInformation structureVersion="0"
[67]       revocationVersion="0"
[68]       distributionURIs="http://marlin-tmo/bkb/bkb.xml"
           issuedOn="..."    nextUpdate="..."
[69]       ID="sYdkN33vvLG9sAN9bKp/8q0NKU=">
[70]      mQEMAzR ... S5qp =q9mn</sf:RevocationInformation>
[71]    <sf:EncryptedBroadcastKeys>
[72]      QvAsufa ... W3zu =a4bq</sf:EncryptedBroadcastKeys>
[73]    <!-- Signature -->
[74]    <Signature xmlns:ds="http://www.w3.org/2000/09/xmldsig#">
[75]      <SignedInfo>
[76]        ...
[77]       <ds:Reference URI="#sYdkN33vvLG9sAN9bKp/8q0NKU=">
[78]         <ds:DigestMethod
[79]             Algorithm="http://www.w3.org/2000/09/xmldsig#sha1"/>
[80]         <ds:DigestValue>GyGsF0Pi4xPU.. .</ds:DigestValue>
[81]       </ds:Reference>
[82]      </SignedInfo>
[83]    </Signature>
[84]  </sf:BroadcastKeyBlock>
[85] </oct:Bundle>
```

2499 **Example 7-1 XML Depicting super-encrypted content key**

2500 Line [04] begins the <oct:ContentKey> element. Line [08] indicates that the symmetric
2501 encryption method is in use.
2502
2503 Line [10] signals the name of the secret key with the <ds:KeyName> element.
2504
2505 On line [13] the <xenc:CipherValue> element carries the base64 encoded representation
2506 of the super-encrypted CK. Decrypting this would reveal the XML representation of the
2507 encrypted content key as exemplified below:
2508

```
<EncryptedData xmlns="http://www.w3.org/2001/04/xmlenc#">
 <EncryptionMethod Algorithm="http://marlin-drm.com/starfish/algorithmID/1.0"/>
 <KeyInfo xmlns="http://www.w3.org/2000/09/xmldsig#">
   <KeyName>urn:marlin:starfish:keytree:1</KeyName>
 </KeyInfo>
```

```
        <CipherData>
          <CipherValue>YOUENCRYPTEDRCONTENTKEYCOMESHERE</CipherValue>
        </CipherData>
      </EncryptedData>
```

On line [64] of Example 7-1 the <sf:BroadcastKeyBlock> element is included in an <oct:Bundle> element.

The integrity of the BKB revocation information must be protected. Line [66] depicts the <sf:RevocationInformation> element and line [74] shows the <ds:Signature> element which carries the integrity protection information.

Line [71] illustrates the <sf:EncryptedBroadcastKeys> element which carries the base64 encoding of the encrypted (with interval keys derived from the device key set [Starfish] §3.3.1) broadcast key.

8 Renewability

8.1 Overview

Renewability in Marlin Core System implementations is supported in several ways:

- Via implementation of upgrade mechanisms – devices that implement Marlin Core System Specifications MUST support a mechanism for updating the version of their implementation.
- Licenses MAY encode a minimum security specification version requirement in control programs (see Section 8.2 and Section 12.5.4.3).
- Via distribution and checking of Certificate Revocation Lists. Certificates that have been revoked MUST be published in a Certificate Revocation List (CRL). The distribution of the CRL MAY be accomplished via the Marlin Security Data Provider or some other mechanism. Additionally, certificates MAY be issued with the CRL Distribution Point (CDP) extension which can be used to aid in retrieving the freshest CRL. If revocation lists are used, only the serial number of revoked certificates SHOULD be include. That is, expired certificates SHOULD NOT remain in a CRL subsequent to their expiration. This practice may significantly reduce the size of the CRL.
- Via denial of service to devices whose certificates have been revoked or that do not implement an acceptable version of the specification
    - Marlin protocols MUST include assertions that certify the version of their implementation. Service policies MUST require these assertions be present in the request messages.
    - Licenses MAY encode a minimum specifications version requirement.
- As a last resort, [Starfish] Broadcast Encryption may be used to permanently prevent a device from accessing further content in a Marlin ecosystem. However, once revoked, version upgrade of the DRM Client will not suffice to restore its ability to participate in the ecosystem, as it would need to re-acquire a personality node and all content previously bound to that node would be invalid. The distribution of the Broadcast Key Block MAY be accomplished via the Marlin Security Data Provider or some other mechanism. Additionally, a Broadcast Key Block MAY be issued with a list of Distribution URIs. The Distribution URIs MAY be used to retrieve the current Broadcast Key Block in a manner specific to the URI.

8.2 Specification Version Attributes

A client MUST be provisioned with (at manufacturing time or personalization time) a trusted representation of the Security Specification Version attributes. The Security Specification Version Attributes convey trusted attributes about a system entity (a Nemo node). The lifecycle and security properties of these asserted attributes signal the security facilities of the underlying specification. In practice Security Specification Version attributes would only change if a security issue required some fundamental change in security components of the specification. For example, if one of the requisite encryption algorithms were found to have a design flaw then a new algorithm would be adopted. In this example, that would require the Security Specification Version attributes of the affected specification to change so as to reflect the new security component.

The Security Specification Version attribute information is decoupled from a particular product implementation so as to independently manage and renew the security infrastructure. Specifically the lifecycle of, and the type of information signaled with Security Specification Version attributes are generally bound to the lifecycle of specified security mechanisms.

The Security Specification Version and Role attributes are conveyed within a SAML assertion. As an optimization, Security Specification Version attributes SHOULD be asserted in the same SAML assertion as the Role attributes. The authority to generate this assertion may be delegated to Marlin Services however this operational detail is not in the scope of this specification.

The asserted attributes apply to the entire Nemo node. The assertion MUST be bound to the Nemo Node of the DRM Client or other system entities and MUST be bound to the Security Specification Version attributes of the implementation itself. An implementation MUST ensure that the actual Security Specification Version attributes of the implementation are greater or equal to that of the assertion it possesses.

The assertion(s) bearing Security Specification Version attributes MUST be included in appropriate Nemo protocols along with the asserted Role attributes to allow services to evaluate the attributes when deciding to provide service.

- When the client assertion(s) is rejected by the service, the client SHOULD attempt to update its Security Specification Version attributes to an acceptable level. This may require a software update (or simply an upgrade to an existing assertion if the implementation was already updated).
- The above allows software update and Security Specification Version attribute assertion acquisition to occur independently. If the versions in the assertion are less than what the current client software supports, it may be sufficient to acquire a new assertion that reflects the current software capabilities. Only if the client does not support an acceptable version of the specifications are software update and assertion renewal required.

The Security Specificatin Version attributes referenced in the assertion MUST be made visible to the Octopus DRM Engine to be resolved at license evaluation time. Refer to the Section 12.5.4.3 of the DRM Usage Profiles for the specific attributes and arguments to the Octopus System.Host.GetObject() that allow an Octopus engine to access the value of the Security Specification Version certified in the assertion.

9 Marlin Trust Management

9.1 Certificates

Certificates assert a binding between an identity and a public key. The format of the certificates used in Marlin is X.509 v3 [X509]. Additionally, the certificates used in Marlin rely upon [PKIX] as a certificate profile. Except where otherwise noted the certificate fields SHALL comply with the X.509 specification [X509] and the IETF PKIX profile [PKIX].

Certificates for NEMO Nodes are subject to the NEMO Trust Management Bindings ([NEMO] §4) specification.

The certified public keys used in Marlin have specific policies attached to them. The various Certification Authorities manage the lifecycle of the keys and their usage.

Certificate Contents

Typical contents of X.509 certificates used in Marlin consist of the following fields:

- Version.
- Serial Number.
- Signature.
- Issuer.
- Validity.
- Subject.
- Subject Public Key Information.
- Extensions:
    - Authority Key Identifier.
    - Subject Key Identifier.
    - Key Usage.
    - Basic Constraints.
    - Certificate Policies
    - CRL Distribution Points

9.1.1.1 Version

The value of this field MUST be 2, which corresponds to X.509 version 3 Certificates.

```
Version  ::=  INTEGER  {  v3(2)  }
```

9.1.1.2 Signature

The value of this field SHALL be either *sha-1WithRSAEncryption*[6] [RFC3279] or *sha256WithRSAEncryption* [PKIXALGS].

```
sha-1WithRSAEncryption OBJECT IDENTIFIER  ::=  {
       iso(1) member-body(2) us(840) rsadsi(113549) pkcs(1) pkcs-1(1) 5  }

sha256WithRSAEncryption OBJECT IDENTIFIER ::= {
       iso(1) member-body(2) us(840) rsadsi(113549) pkcs(1) pkcs-1(1) 11 }
```

9.1.1.3 Issuer

The distinguished name of the Issuer MUST be represented with a single directory name attribute. The attribute type MUST be either a X.500 commonName or a directory name attribute whose syntax adheres to a URN and is identified by the object identifier is *id-nat-uri*. The latter is the preferred attribute type and it MUST be used for all CA or end-entity certificates managed outside of the trust authority (see Sections 9.4.) This attribute SHALL be encoded using UTF-8.

```
 cf.
   id-marlin   OBJECT IDENTIFIER ::= {iso(1) identified-organization(3) dod(6)
                        internet(1) private(4) enterprise(1) marlin(23727)}
   id-nemo     OBJECT IDENTIFIER ::= {id-marlin nemo(1)}
   id-nemo-nat OBJECT IDENTIFIER ::= {id-nemo nameAttribute(1)}
   id-nat-uri  OBJECT IDENTIFIER ::= {id-nemo-nat 1}
```

See [NEMO] §4 for more information.

9.1.1.4 Subject

The distinguished name of the Subject field MUST be represented with a single attribute. The attribute type MUST be either a X.500 commonName or a URI attribute whose object identifier is *id-nat-uri*. The latter is the preferred attribute type and it MUST be used for all CA or end-entity certificates managed outside of the trust authority (see Sections 9.4.) This attribute SHALL be encoded using UTF-8.

9.1.1.5 Subject Public Key Info

This field carries the public key of the subject and identifies the algorithm with which the key is used. Presently the only supported algorithm is *rsaEncryption*.

```
pkcs-1 OBJECT IDENTIFIER ::= { iso(1) member-body(2) us(840) rsadsi(113549) pkcs(1) 1 }

rsaEncryption OBJECT IDENTIFIER ::=  { pkcs-1 1}
```

9.1.2 Excluded Certificate Extensions

Generally we are silent regarding optional certificate extensions. However, in some cases the use of the optional extension may have an adverse impact on interoperability, performance or implementation complexity. Therefore, the remainder of this sub-section indicates which of these extensions SHOULD NOT be included in certificates adhering to this profile.

[6] Note that the key size is a matter to be determined by a compliance body. Also note that current best practices are for the trust anchors to have larger key sizes, which, as of this writing, is typically 2048 bits.

9.1.2.1 Policy Mappings

The Policy Mappings extension SHOULD NOT be included in certificates adhering to this profile. Doing so may impact interoperability and performance.

9.1.2.2 Policy Constraints

The Policy Constraint extension SHOULD NOT be included in certificates adhering to this profile. The Policy Constraint extension could be used as a technical mechanism by which to enforce the above recommendation to prevent Policy Mappings. However, using Policy Constraints may have an adverse effect on implementation complexity and interoperability.

9.1.2.3 Subject Alternative Name

The Subject Alternative Name extension SHOULD NOT be included in certificates adhering to this profile.

9.1.2.4 Issuer Alternative Name

The Issuer Alternative Name extension SHOULD NOT be included in certificates adhering to this profile.

9.1.3 Certificate Extensions

Marlin Core System implementation certificate extension fields include Authority Key Identifier, Subject Key Identifier, Basic Constraints, Certificate policy, and CRL Distribution Points.

9.1.3.1 Authority Key Identifier

This field contains a hash of the issuer's public key.

```
extnID   : id-ce-authorityKeyIdentifier OBJECT IDENTIFIER ::=  { id-ce 35 }
critical : FALSE
value    : hash(PublicKey)
```

The key identifier SHOULD be composed of the 160-bit SHA-1 hash (as defined in [PKIX] §4.2.1.2 method 1) of the value of the bit string *issuerPublicKey* (excluding the tag, length, and number of unused bits). This field is used to enable key changeover.

9.1.3.2 Subject Key Identifier

This field contains a hash of the subject's public key.

```
extnID   : id-ce-subjectKeyIdentifier OBJECT IDENTIFIER ::=  { id-ce 14 }
critical : FALSE
value    : hash(PublicKey)
```

The key identifier is composed of the 160-bit SHA-1 hash (as defined in [PKIX] §4.2.1.2 method 1) of the value of the bit string *subjectPublicKey* (excluding the tag, length, and number of unused bits).

9.1.3.3 Key Usage

The key usage extension defines the purpose for which the key has been certified. For example, it specifies whether a key can be used for signature, certificate signing and key or data encipherment. The key usage field contains a bit string consisting of a series of

flags, as indicated in [PKIX] §4.2.1.3. The particular key usages utilized by this profile are specified in Section 9.4.

```
extnID   : id-ce-keyUsage OBJECT IDENTIFIER ::=  { id-ce 15 }
critical : TRUE
```

9.1.3.4 Basic Constraints

This field contains the value of the certificate's basic constraints. The basic constraints extension specifies whether the subject of the certificate is a Certificate Authority (CA) and in that case the maximum number of CA certificates that can follow this certificate in a certification path. This profile MUST adhere to the definition provided in [PKIX] §4.2.1.10.

```
extnID   : id-ce-basicConstraints OBJECT IDENTIFIER ::=  { id-ce 19 }
critical : TRUE
```

9.1.3.5 CRL Distribution Points Field

This field identifies how CRL information is obtained. This profile relies upon an indirect CRL as described in [PKIX] §5. The CRL Distribution Points field MUST contain a *DistributionPointName*. This name MUST contain a general name of type URI. This URI is a pointer to the current CRL and is issued by the entity identified in *cRLIssuer*.

```
extnID   : id-ce-cRLDistributionPoints OBJECT IDENTIFIER ::=  { id-ce 31 }
```

All implementations SHALL be prepared to resolve the URI using the HTTP GET method. The URI MUST point to a file that has an extension of ".crl". The file MUST contain the DER encoded CRL [RFC 2585]. The mime-type of the returned resource SHALL be “application/pkix-crl”.

The following is an (non-normative) example of how the cRLDistributionPoints field is populated:

```
cRLDistributionPoints:
    DistributionPoint:
        distributionPoint: fullName: uniformResourceIdentifier: http://marlin-
tmo.com/crl/mtmocrls.crl
        cRLIssuer: directoryName: URI=urn:marlin:datacertification:revocation
```

9.1.3.6 Certificate Policies

As specified in [PKIX] §4.2.1.5, this field indicates the policies under which the certificate has been issued and the purposes for which the certificate may be used. It plays a critical role in the construction of certification paths and the validation there of.

```
extnID   : id-ce-certificatePolicies OBJECT IDENTIFIER ::= {id-ce 32}
critical : TRUE
```

The field contains *policyIdentifier* parameters within the *PolicyInformation* to indicate the OIDs of the policies. Certificates SHALL NOT include *policyQualifiers* other than those defined in [PKIX] §4.2.1.5, the CPS Pointer and the User Notice qualifiers.
The CA MAY include the special policy *anyPolicy* as prescribed in [PKIX] §4.2.1.5.

```
anyPolicy OBJECT IDENTIFIER ::= { id-ce-certificatePolicies 0 }
```

The application specific policy OIDs are defined in Sections 9.1.3.6.1 and 9.1.3.6.2. The certificate policies and end-entity purposes for these different policy identifiers are further described in Sections 9.1.4.4, 9.1.4.5 and 9.4.

9.1.3.6.1 Certificate Policy OIDs for Octopus

The certificate policy OIDs defined for Octopus are shown below.

```
id-octopus           OBJECT IDENTIFIER ::= {id-marlin octopus(2)}
id-octopus-cp        OBJECT IDENTIFIER ::= {id-octopus certificatePolicies(2)}
id-octopus-marlin-cp OBJECT IDENTIFIER ::= {id-octopus-cp marlin(10)}
```

The following Marlin policy identifier OIDs are defined for the Octopus DRM.

```
id-cp-octopus-marlin-scuba-sharing-key              ::= {id-octopus-marlin-cp 1}
id-cp-octopus-marlin-scuba-confidentiality-key      ::= {id-octopus-marlin-cp 2}
id-cp-octopus-marlin-signing-personality            ::= {id-octopus-marlin-cp 3}
id-cp-octopus-marlin-signing-node                   ::= {id-octopus-marlin-cp 4}
id-cp-octopus-marlin-signing-link                   ::= {id-octopus-marlin-cp 5}
id-cp-octopus-marlin-signing-controller             ::= {id-octopus-marlin-cp 6}
id-cp-octopus-marlin-signing-control                ::= {id-octopus-marlin-cp 7}
```

Note: The OIDs represented by the identifiers id-cp-octopus-marlin-scuba-sharing-key and id-cp-octopus-marlin-scuba-confidentiality-key are reserved for future use.

9.1.3.6.2 Certificate Policy OIDs for NEMO

The Certificate policy OIDs defined for NEMO are shown below.

```
id-nemo           OBJECT IDENTIFIER ::= {id-marlin nemo(1)}
id-nemo-cp        OBJECT IDENTIFIER ::= {id-nemo certificatePolicies(2)}
id-nemo-marlin-cp OBJECT IDENTIFIER ::= {id-nemo-cp marlin(10)}
```

The following Marlin policy identifier OIDs are defined for the NEMO protocol and services.

```
id-cp-nemo-marlin-client-drm-key                    ::= {id-nemo-marlin-cp 1}
id-cp-nemo-marlin-service-key                       ::= {id-nemo-marlin-cp 3}
id-cp-nemo-marlin-signing-data-certification        ::= {id-nemo-marlin-cp 4}
id-cp-nemo-marlin-signing-service-role              ::= {id-nemo-marlin-cp 5}
id-cp-nemo-marlin-signing-client-role               ::= {id-nemo-marlin-cp 6}
id-cp-nemo-marlin-signing-security-metadata         ::= {id-nemo-marlin-cp 7}
id-cp-nemo-marlin-signing-bootstrap                 ::= {id-nemo-marlin-cp 8}
id-cp-nemo-marlin-signing-content-metadata          ::= {id-nemo-marlin-cp 9}
```

Note: OID `{id-nemo-marlin-cp 2}` is reserved.

9.1.4 Certificate Validation

The certificate user MUST check a sequence of certificates until reaching a trust anchor to verify the validity of the certified public key.
Except where otherwise noted, the process of validating a certificate SHALL produce the same results and exhibit the same behavior as the path validation process defined in [PKIX] §6.1.

The certificate validation process consists of the following steps:

- Certificate Verification Process.

- Certificate Checking Process.
- Signature Verification Process.
- Key/Data Encipherment Process.

9.1.4.1 Trust Anchors

To validate a certificate, a certificate user MUST have the proper trust anchor or utilize an online certificate validation service. If the certificate user performs path validation it MUST manage trust anchor information securely to prevent unauthorized changes. The secure distribution of trust anchors is out of scope for this specification. However, a description of a trust hierarchy which supports these best practices and enables certain operational efficiencies is discussed in Section 9.4.

9.1.4.2 Certificate Path Validation

Each certificate MUST be verified by checking the certificate against its issuer. The issuer certificate MUST be checked against its issuer, in turn, until the trust anchor is reached. It is RECOMMENDED that the algorithm defined in [PKIX] §6.1 is implemented.

Input data to the certificate path validation process include the following items:

- A certificate (target).
- Issuer certificate information, including the following:
  - Trust Anchor.
  - Public key.
  - Subject.
  - Basic constraints.
  - Key usage.
  - Certificate Policies.
- List of acceptable certificate policies
- The current date and time.

The validation algorithm defined in [PKIX] §6.1 takes into consideration Certificate Policies. Thus the validation algorithm implemented MUST also meet the following conditions:

- **Check Certificate Policies**
  The policy information terms of the issuing certificate MUST be either *anyPolicy* or one or more of the intended certificate policy terms. See [PKIX] §4.2.1.5 and §6.1.3. The policy terms MUST be checked against the acceptable policies. See Section 9.1.3.6.1 and 9.1.3.6.2 for the policy identifiers. The acceptability of the policy terms are described in Sections 9.3 and 9.4.

9.1.4.3 Certificate Checking Process

The certificate checking process determines if the certificate can act on the specified target type. Input data to the certificate checking process includes a valid end-entity certificate from the path validation process and the key usage required to act on the target. To check the validity of the certificate, the system MUST check the following values:

- **Key Usage**
  The key usage MUST be appropriate for the target type.

9.1.4.4 Signature Verification of Signed Data

Input data to the signature verification process includes the following items:

- Trust Anchor.
- Target data.
- Target signature.
- Target type.
- Certificate information (verified, as in Section 9.1.4.2), including the following:
  - Trust Anchor
  - Public Key
  - Subject
  - Basic Constraint
  - Key Usage

The output from the signature verification process is either a positive or negative result. To validate a signature, the system MUST check the following items:

- **Target Signature**
  The target data MUST be signed by the public key of the certificate.
- **Key Usage**
  The *digitalSignature* bit MUST be set in the key usage field of the certificate.

For the signing keys used to secure NEMO messages refer to Section 6.3 for processing rules regarding the NEMO security bindings as they relate to the NEMO signing targets.

9.1.4.5 Key/Data Encipherment Process

Input to the key or data encipherment process includes the following information:

- Target information includes the following:
  - Target data.
  - Target type.
- Certificate information (verified, as in Section 9.1.4.2) includes the following:
  - Trust Anchor
  - Public Key
  - Subject
  - Key Usage

Output from the key or data encipherment process consists of the encrypted data.
To encrypt data or keys, the system must check the following items:

- **Key Usage**
  If target data is a key, the KeyEncipherment bit must be set in key_usage of the certificate. If target data is not a key, the DataEncipherment bit must be set in key_usage of the certificate.

For processing rules regarding the NEMO security bindings as they relate to the NEMO encryption targets refer to Section 6.3.

9.2 Certificate Revocation List

A Certificate Revocation List (CRL) is used to convey to a certificate user the set of revoked certificates. The format of the CRL used in Marlin is X.509 v2 CRL [X509]. Additionally, the CRL used in Marlin rely upon [PKIX] as a CRL profile. This document specifies the use of indirect CRLs and their contents. The balance of this section describes the necessary fields to support an indirect CRL. Except where otherwise noted the CRL fields SHALL comply with the X.509 specification [X509] and the IETF PKIX profile [PKIX].

CRL Contents

Contents of X.509 CRLs used in Marlin consist of the following fields:

- Version.
- Signature.
- Issuer.
- This Update.
- Next Update.

- Revoked Certificates.
  - User Certificate.
  - Revocation Date.
  - CRL Entry Extension:
    - Certificate Issuer.
- CRL Extensions:
  - Authority Key Identifier
  - CRL Number.
  - Issuing Distribution Point.

9.2.1.1 Version

The value of this field MUST be 1, which corresponds to X.509 version 2 CRL.

```
Version  ::=  INTEGER  {  v2(1)  }
```

9.2.1.2 Signature

The value of this field SHALL be either *sha-1WithRSAEncryption*[7] [RFC3279] or *sha256WithRSAEncryption* [PKIXALGS].

```
sha-1WithRSAEncryption OBJECT IDENTIFIER  ::=  {
       iso(1) member-body(2) us(840) rsadsi(113549) pkcs(1) pkcs-1(1) 5  }

sha256WithRSAEncryption OBJECT IDENTIFIER ::= {
       iso(1) member-body(2) us(840) rsadsi(113549) pkcs(1) pkcs-1(1) 11 }
```

9.2.1.3 Issuer

The distinguished name of the CRL Issuer MUST be represented with a single directory name attribute. The attribute type MUST be either a X.500 commonName or a directory name attribute whose syntax adheres to a URN and is identified by the object identifier *id-nat-uri* as defined in Section 9.1.1.3.

9.2.1.4 CRL Entry Extension

All mandatory fields for this extension must be present and follow the guidance given in [PKIX] §5.3. Of particular importance is the presence of the *certificateIssuer* field which identifies the authority which issued the revoked certificate.

9.2.1.5 CRL Extensions

As previously mentioned, the specification adheres to the [PKIX] CRL profile which mandates which fields must be present in the CRL Extensions. Specifically the *AuthorityKeyIdentifier* and the *CRLNumber* MUST be present. Additionally, since this specification relies upon an indirect CRL the *issuingDistributionPoint* extension MUST be present.

[7] Note that the key size is a matter to be determined by a compliance body. Also note that current best practices are for the trust anchors to have larger key sizes, which, as of this writing, is typically 2048 bits.

9.2.1.5.1 Issuing Distribution Point

This field MUST follow the guidance given in [PKIX] §5.2.5. Specifically since the CRL is an indirect CRL the *indirectCRL* field MUST be present and MUST have a value of TRUE.

9.3 Trust Management of Marlin Services (Informative)

An implementation of a Marlin Core System may consist of a number of services. These services rely upon the trust management mechanisms defined by this specification. Additionally, delivery system specifications which build upon the Marlin Core System may utilize the same mechanisms defined here. This section gives a general overview of the types of services and the manner in which they may leverage the trust management mechanisms defined elsewhere in this specification.

9.3.1 Secure Peer Interactions

9.3.1.1 Keys Used in NEMO Secure Communications

A short summary of the keys used to secure communications using the NEMO stack is provided below. The functions of the keys issued to a service are typically common for all such entities. They are as follows:

- **Keys for signing and encrypting protocol data**
  All NEMO-based services have these keys.
- **Keys for signing the service role attribute assertions**
  An entity must have a valid, signed role attribute assertion to be accepted as a NEMO service. These signing keys are issued under the authority of a service specific CA.
- **Key for signing the client role attribute assertions**
  An entity must have a valid, signed client role attribute assertion to be accepted as a NEMO client. The Personalization Service signs client role attribute assertions.
- **Keys for signing and encrypting data at the client**
  All clients have these keys. There may be separate key sets for use by DRM-related applications and non-DRM related applications.

9.3.2 DRM Services

A variety of services which are either defined or enabled by this specification fall into the catagory of DRM Services. In general, these services are associated with the provisioning, management and realization of content sharing and distribution systems. Examples of the services which are specifically defined by this specification are called out in the subsequent sections. Examples of services which are enabled by this specification are those services which directly support a specific delivery system technology and the business models which they enable. For example a service which collects content usage information would fall into this catagory.

9.3.2.1 Registration Services

Registration Sevices are those services which are responsible for the provisioning and relationship management of Octopus Objects used to influence the governance model. One such service defined by this specification is the DRM Object Provider service.

Trust Management supports these types of services by defining the certified keys used to secure the Octopus Objects. These keys are certified by a Registration Services CA (see 9.4.4.) Specifically, these certified keys enable a service to sign User Nodes, Links, and Controls.

- **User Nodes**
  User Nodes contain a set of Scuba Sharing Keys.
- **Links**
  Links are issued by the Registration Service.
- **Controls**
  Controls may be issued by the Registration Service.

9.3.2.2 License Issuing Services

License Issuing Services are those services which are responsible for the creation of Octopus Objects used to govern access to content.

Trust Management supports these types of services by defining the certified keys used to secure some of these Octopus Objects. These keys are certified by a Content License Services CA (see 9.4.5.) Specifically, these certified keys enable a service to sign Controller and Control objects.

9.3.2.3 Marlin Personalization Services

Personalization Services are those services which are responsible for the provisioning of Octopus Objects used to represent the personality of a device.

Trust Management supports these types of services by defining the certified keys used to secure these Octopus Objects. These keys are certified by a DRM Personalization CA. These certified keys enable the certificate user to sign Personality Nodes.

- **Personality Nodes**
  A DRM Client Personality Node contains a set of Scuba Sharing Keys. These nodes and the keys they carry are issued by a personalization service or device manufacturer.

9.3.3 Data Certification Services

A variety of services which are either defined or enabled by this specification fall into the catagory of Data Certification Services. In general, these services are associated with the provisioning, management and renewal of security and integrity services. Examples of the services which are specifically defined by this specification are called out in the subsequent sections.

9.3.3.1 Security Metadata Certification Services

Security Metadata Certification Services are those services which are responsible for the attestation and issuance of security metadata used in the renewal and remediation of trust objects. Some examples of security metadata items are trusted time and certificate revocation lists (CRLs). One such service defined by this specification is the Security Data Provider service (Section 4.2.3.)

Trust Management supports these types of services by defining the certified keys used to secure the metadata. These keys are certified by a Security Metadata Certification Services CA. Specifically, these certified keys enable the service to sign tokens attesting to the freshness or qualities of security metadata.

9.3.3.2 Content Metadata Certification Services (aka Content Packager)

Content Metadata Certification Services are those services which are responsible for the integrity protection of the content metadata.

Trust Management supports these types of services by defining the certified keys used integrity protect the content metadata.

9.4 Trust Hierarchies and Policies

The Marlin Core System implementation employs X.509 Version 3 certificates for binding an identity to a public key, and constraining the usage of the certified key to a specific purpose. The following sections define the certificate profile, the certificate contents of defined fields, and the trust hierarchies for the management and validation of Marlin Core System implementation specification certificates. Except where otherwise noted certificates for Marlin Core System implementation SHALL comply with the X.509 Version 3 specification [X509] and the IETF PKIX profile [PKIX].

The symbols in Figure 9-1 are used to depict the relationship of the entities in the trust hierarchy including the entities which participate and the certificates and keys issued.

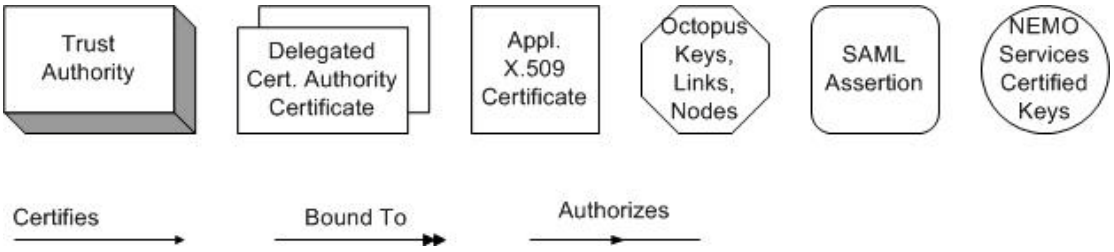


Figure 9-1 Symbol Legend

A single ended arrow between objects in the diagram describes a “Certifies” relationship. In essence, this suggests that the certificate from which the arrow sources is used to sign the object that the arrow points to. A double ended arrow between objects describes a binding of and identity to a role. A single midline arrow indicates an ‘authorizes’ relationship.

The distinction between the "Trust Authority” and the "Delegated CA" is that a "Delegated CA" is operated by Marlin participating companies (or on their behalf)

whereas the Trust Authority is operated by a Marlin-wide entity.

The following diagram depicts the top level trust topology of the Marlin Core System.

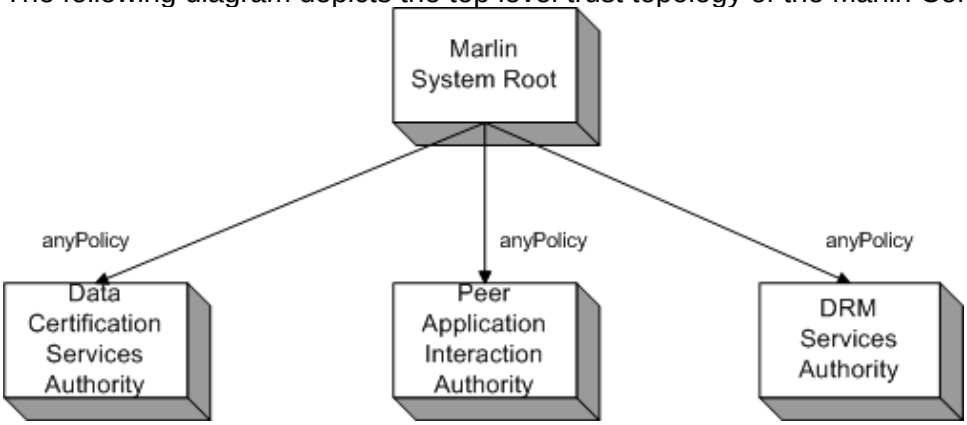


Figure 9-2 Top-level Trust Topology

The trust topology is comprised of a self-signed system root, the Marlin System Root, and three functional certificate hierarchies, the DRM Services Certificate Hierarchy, the Data Certification Services Certificate Hierarchy and the Peer Application Interaction Hierarchy[8]. These trust authorities delegate to subordinate Certification Authorities the responsibility of issuing and managing the lifecycle of CA and end-entity certificates. These subordinate CAs may have a certificate policy of “anyPolicy” or a specific set of certificate policy terms may be bound to their certificates. It is REQUIRED that downstream subordinate CAs will have specific certificate policy terms in their CA certificates and that they will propagate those terms into the end-entity certificates they issue. That said, the certificates issued by the subordinate CAs must adhere to the policy information terms conveyed in their certificate. Additionally, the subordinate CAs further constrains their subordinates and end-entities with certificate policies, and key[9].

9.4.1 Peer Application Interaction Trust Hierarchy

The Marlin System Peer Application Interaction Trust Hierarchy is responsible for the lifecycle and management of end-entity certificates used to secure participant communication protocol interactions. Generally the certified keys bind a key to an identity of either a client or service system entity.

The following diagram depicts the certificates managed under this authority.

[8] To optimize path validation and to minimize the reliance on self-signed certificates a best practice would be to distribute and rely upon these functional root certificates as trust anchors.

[9] A compliance regime may dictate otherwise but it may be operationally simpler for the Marlin system root authority to not constrain in any way the functional roots and for the functional roots to minimize any constraints it puts on the operational certification authorities. With that in mind the root would not specify any certificate policies and the functional roots would minimally constrain the operational CAs to “anyPolicy”.

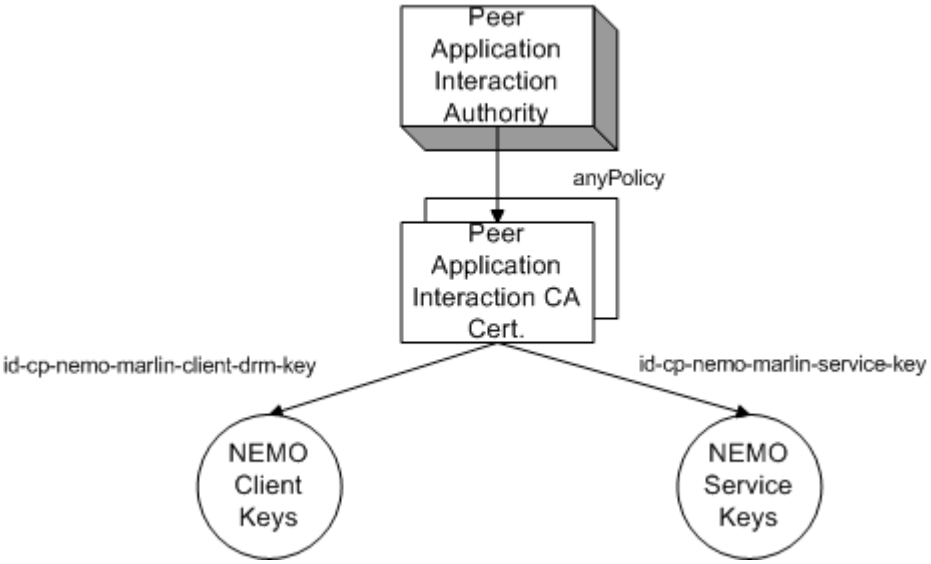


Figure 9-3 Peer Application Interaction Trust Hierarchy

9.4.1.1 Peer Application Interaction Authority

The Peer Application Interaction Authority is used to authorize instances of Peer Application Interaction Certification Authorities. This authority is used to issue certificates to Marlin participants so that they may operate a CA which certifies end-entity certificates for Devices or Client applications and Services such that they may communicate application or DRM data in a secure and interoperable manner.

The following table describes the Certificates issued by this authority and the intended usage of the keys certified by CA subordinates.

Certificates that Authorize a Subordinate CA to Certify …	**Basic Constraint**	**Key Usage**	**Certificate Policy Identifiers**
Signing Certificates for Devices and Client Applications Protocol and Service Interaction			
DRM keys of a NEMO Node	CA=TRUE pathLenConstraint>0	keyCertSign	id-cp-nemo-marlin-client-drm-key
Signing Certificates for NEMO Protocol and Service Interaction			
Service keys	CA=TRUE pathLenConstraint>0	keyCertSign	id-cp-nemo-marlin-service-key

Table 9-1 Peer Application Interaction Certifying Keys

9.4.1.2 Peer Application Interaction Certification Authority

This Certification Authority certifies the keys with the following usage and policies.

Key	Signing/ Encryption Targets	Basic Constraint	Key Usage	Certificate Policy Identifiers
Signing and/or Encrypting DRM or Application Payloads				
Keys for signing DRM/Application data	Data for use by DRM applications	CA=FALSE	digitalSignature	id-cp-nemo-marlin-client-drm-key
Keys for encrypting DRM/Application data	Data for use by DRM applications	CA=FALSE	dataEncipherment	id-cp-nemo-marlin-client-drm-key
Keys for encrypting DRM/Application keys	Keys for use by DRM applications	CA=FALSE	keyEncipherment	id-cp-nemo-marlin-client-drm-key

3133 **Table 9-2 Client End-Entity Certificates**

3134

Key	Basic Constraint	Key Usage	Certificate Policy Identifiers
NEMO Protocol Services Interactions			
Sign protocol data	CA=FALSE	digitalSignature	id-cp-nemo-marlin-service-key
Encrypt protocol data	CA=FALSE	keyEncipherment	id-cp-nemo-marlin-service-key

3135 **Table 9-3 Service End-Entity Certificates**

3136 9.4.2 DRM Services Trust Hierarchy

3137 The following diagram depicts the entities which fall under this portion of the trust
3138 topology.

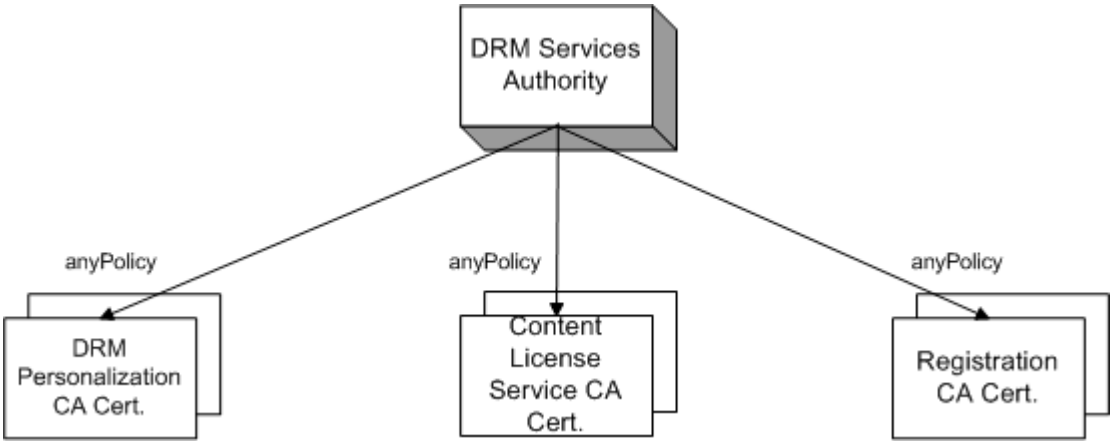


Figure 9-4 DRM Services Trust Hierarchy

9.4.2.1 DRM Services Authority

The DRM Services Authority may be used to authorize instances of a three distinct Certification Authorities. These Certification Authorities are described separately so as to highlight their individual functions and characteristics. However, other factors which are out of scope for this document may influence whether this degree of separation is necessary. For example, compliance criteria or operational efficiencies may be best served by combining the Content License Service CA with the Registration CA. This specification does not mandate separation or combination of the CAs but merely supplies an informative set of guidelines for constructing a trust topology.

The DRM Services Authority may issue certificates to Marlin participants so that they may operate a CA which certifies end-entity certificates for DRM Device or DRM Client application Personalization, Content License issuance and Registration services. This authority may also enable the subordinate CAs to issue certificates used to bind identities to specific roles.

The following subsections describe the three subordinate CAs independently as if they had a separate trust authority.

9.4.3 DRM Client Personalization Trust Hierarchy

The DRM Services Authority enables a trust hierarchy focused on the personalizing DRM Devices and Client Applications.

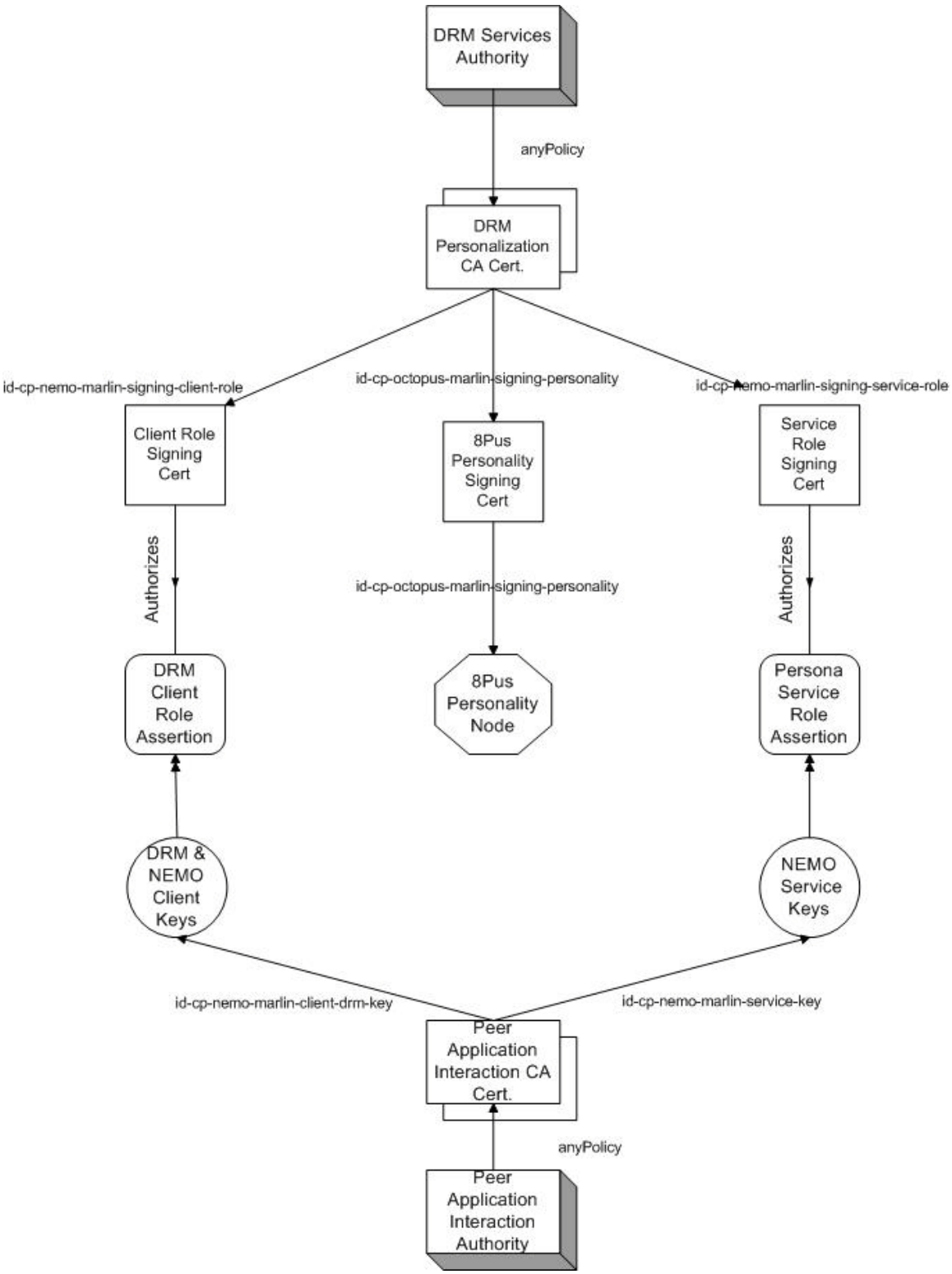


Figure 9-5 DRM Client Personalization

9.4.3.1 Trust Authority

3166 The DRM Services Authority is used to authorize instances of a DRM Personalization
3167 Certification Authority. A DRM Personalization Certification Authority issues end-entity
3168 certificates used to personalize a DRM Client.
3169

Certificates that Authorize a Subordinate CA to Certify …	**Basic Constraint**	**Key Usage**	**Certificate Policy Identifiers**
Marlin Octopus Personalization Service or Device Manufacturer			
Keys for signing personality nodes	CA=TRUE pathLenConstraint>0	keyCertSign	id-cp-octopus-marlin-signing-personality
NEMO Protocol and Service Interaction			
Keys for signing the service role	CA=TRUE pathLenConstraint>0	keyCertSign	id-cp-nemo-marlin-signing-service-role
Keys for signing Client Role information	CA=TRUE pathLenConstraint>0	keyCertSign	id-cp-nemo-marlin-signing-client-role

3170 **Table 9-4 Personalization Certifying Keys**

3171

3172

3173 #### 9.4.3.2 DRM Personalization Certification Authority

3174 In Marlin, a client application hosts both an Octopus Personality Node, which provides a
3175 secure environment for DRM operations, and a NEMO Node, which is used to establish
3176 secure, interoperable communications with various services.
3177
3178 DRM Devices and DRM client applications require a variety of certificates and keys to
3179 interoperate with services and to participate in the consumption of protected content. An
3180 instance of a subordinate DRM Personalization Certification Authority is responsible for
3181 issuing the certificates which are used to personalize a device or client application. The
3182 certificates issued by this authority should be constrained to specific purposes. The
3183 keys, certificates and assertions which personalize a device or client application may be
3184 provisioned at manufacture time or delivered via a personalization bootstrap protocol.
3185
3186 The keys used to sign a Personality node MUST have key usages, and certificate
3187 policies that allow them to sign Octopus Personality Nodes *(*digitalSignature and id-cp-
3188 octopus-marlin-signing-personality, respectively.)
3189

Key Used to Sign…	Basic Constraint	Key Usage	Certificate Policy Identifiers
Marlin Octopus Personalization Service or Device Manufacturer			
Personality nodes	CA=FALSE	digitalSignature	id-cp-octopus-marlin-signing-personality

3190 **Table 9-5 Signing Keys and Certificates for Personalization Service or Device Mfg.**

3191 This subordinate CA also issues certificates which may be used for protocol interactions
3192 and to sign Client role attribute assertions.
3193

Key	Basic Constraint	Key Usage	Certificate Policy Identifiers
Personalization Services			
Keys for signing the client role	CA=FALSE	digitalSignature	id-cp-nemo-marlin-signing-client-role
Keys for signing the service role	CA=FALSE	digitalSignature	id-cp-nemo-marlin-signing-service-role

3194 **Table 9-6 Role Signing Certificates**

3195

9.4.4 Registration Services Trust Hierarchy

The DRM Services Authority enables a trust hierarchy focused on a domain management model.

The following diagram depicts the entities which fall under this portion of the trust topology.

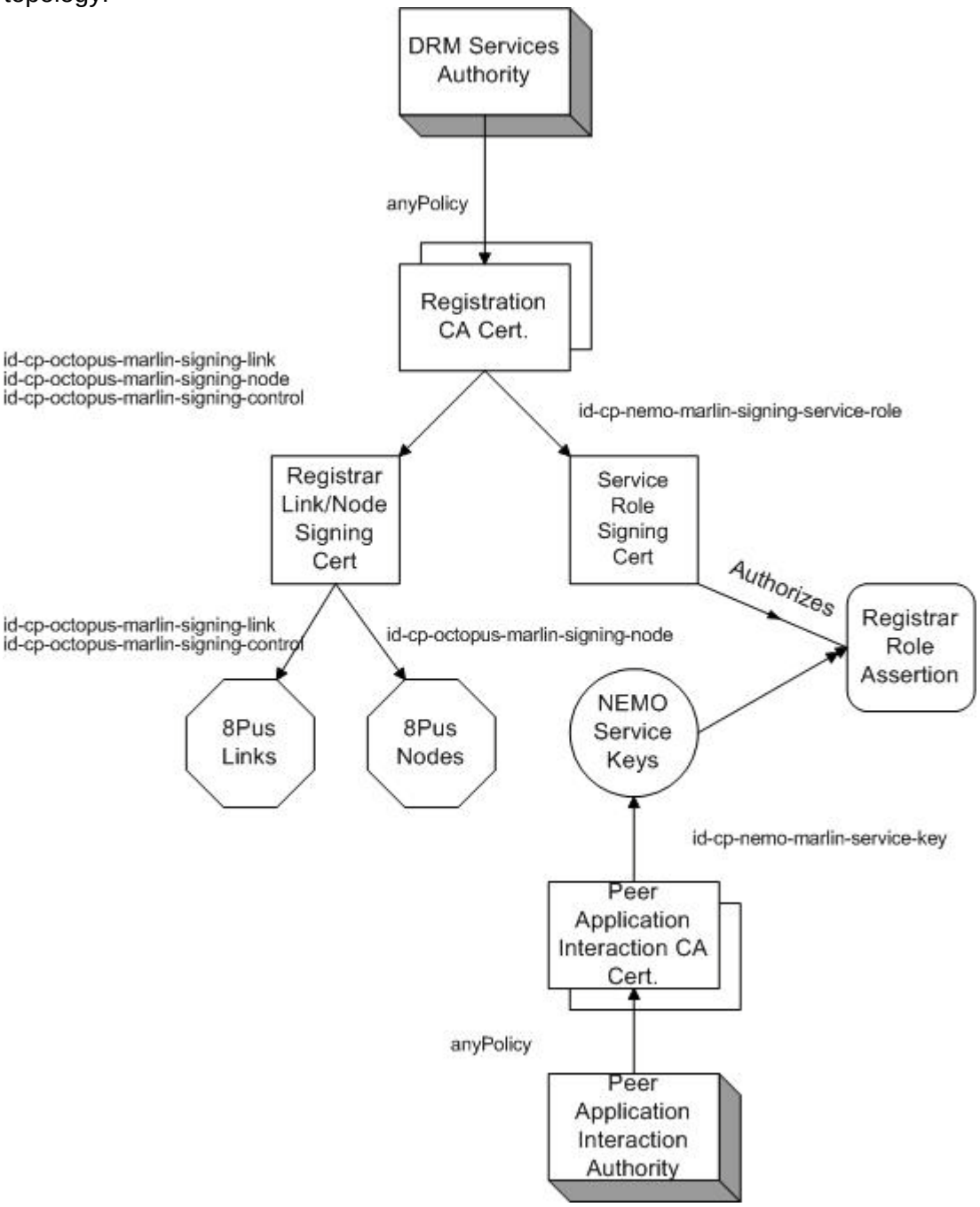


Figure 9-6 Registration Trust Hierarchy

9.4.4.1 Trust Authority

The DRM Services Authority is used to authorize instances of a Marlin User/Device Registration Certification Authority. A Marlin User/Device Registration Certification Authority issues end-entity certificates used to operate a domain registration service. That is, this authority is used to delegate to authorized entities the right to issue certificates which may be used to operate a registration service.

Certificates that Authorize a Subordinate CA to Certify …	**Basic Constraint**	**Key Usage**	**Certificate Policy Identifiers**
NEMO Protocol and Service Interaction			
Keys for signing the service role	CA=TRUE pathLenConstraint>0	keyCertSign	id-cp-nemo-marlin-signing-service-role
Registration Service			
Keys for signing Octopus objects	CA=TRUE pathLenConstraint>0	keyCertSign	id-cp-octopus-marlin-signing-node
			id-cp-octopus-marlin-signing-link
			id-cp-octopus-marlin-signing-control

Table 9-7 Registration Service Authority Subordinate Certifying Keys

9.4.4.2 Registration Certification Authority

A Registration Certification Authority issues certified keys to a registration service (a Registrar) which enable the service to engage in protocol interactions with Nemo clients, sign Octopus User/Domain Nodes, Octopus Links, Octopus Controls and to sign Registrar role attribute assertions.

Marlin supports a model by which a domain of devices or client applications can be formulated to enable a rich user experience. In some circumstances this is accomplished by associating a content rendering application (or device) with a user account. The subordinate Registration Certification Authorities issues end-entity certificates to operational entities which manage these linkages. The certificates issued constrain the operational entities to use the issued certificate for the purpose of signing Octopus Nodes, Links and Controls.

The end-entity signing certificates issued MUST have a certificate policy term that indicates the certified key is intended to be used to sign Controls and Links (*id-cp-octopus-marlin-signing-control* and *id-cp-octopus-marlin-signing-link*, respectively).

Section 3.3.4 describes the mechanism by which Octopus objects are signed. As noted, Controls may be indirectly signed either by reference or as a result of being contained by the object being directly signed. Implementations SHALL take this into consideration when selecting acceptable policy terms. For example, when a Control is covered by the

signature over a Link object, the acceptable policy terms (for the Control) would be *id-cp-octopus-marlin-signing-link* AND *id-cp-octopus-marlin-signing-control*. Because the signed Link object includes the Control, the identity of the Link signer (extracted from the Subject distinguished name of the signers X.509 certificate) is also the signer of the Control.

Key Used to Sign…	**Basic Constraint**	**Key Usage**	**Certificate Policy Identifiers**
Registration Service			
Non-Personality Nodes, Links and Controls	CA=FALSE	digitalSignature	id-cp-octopus-marlin-signing-node
			id-cp-octopus-marlin-signing-link
			id-cp-octopus-marlin-signing-control

Table 9-8 Registration Service Certifying Keys

This subordinate CA also issues certificates which may be used for protocol interactions and to sign Registration Service (aka Registrar) role attribute assertions. The DRM Object Provider and Domain Information Provider roles are asserted through a key authorized under this Certification Authority.

Key Used to Sign…	**Basic Constraint**	**Key Usage**	**Certificate Policy Identifiers**
Registration Service			
Keys for signing the service role	CA=FALSE	digitalSignature	id-cp-nemo-marlin-signing-service-role

Table 9-9 Role Signing Certificate

9.4.5 Content Licensing Trust Hierarchy

The DRM Services Authority enables a trust hierarchy focused on content license issuance. The following diagram depicts the entities which fall under this portion of the trust topology.

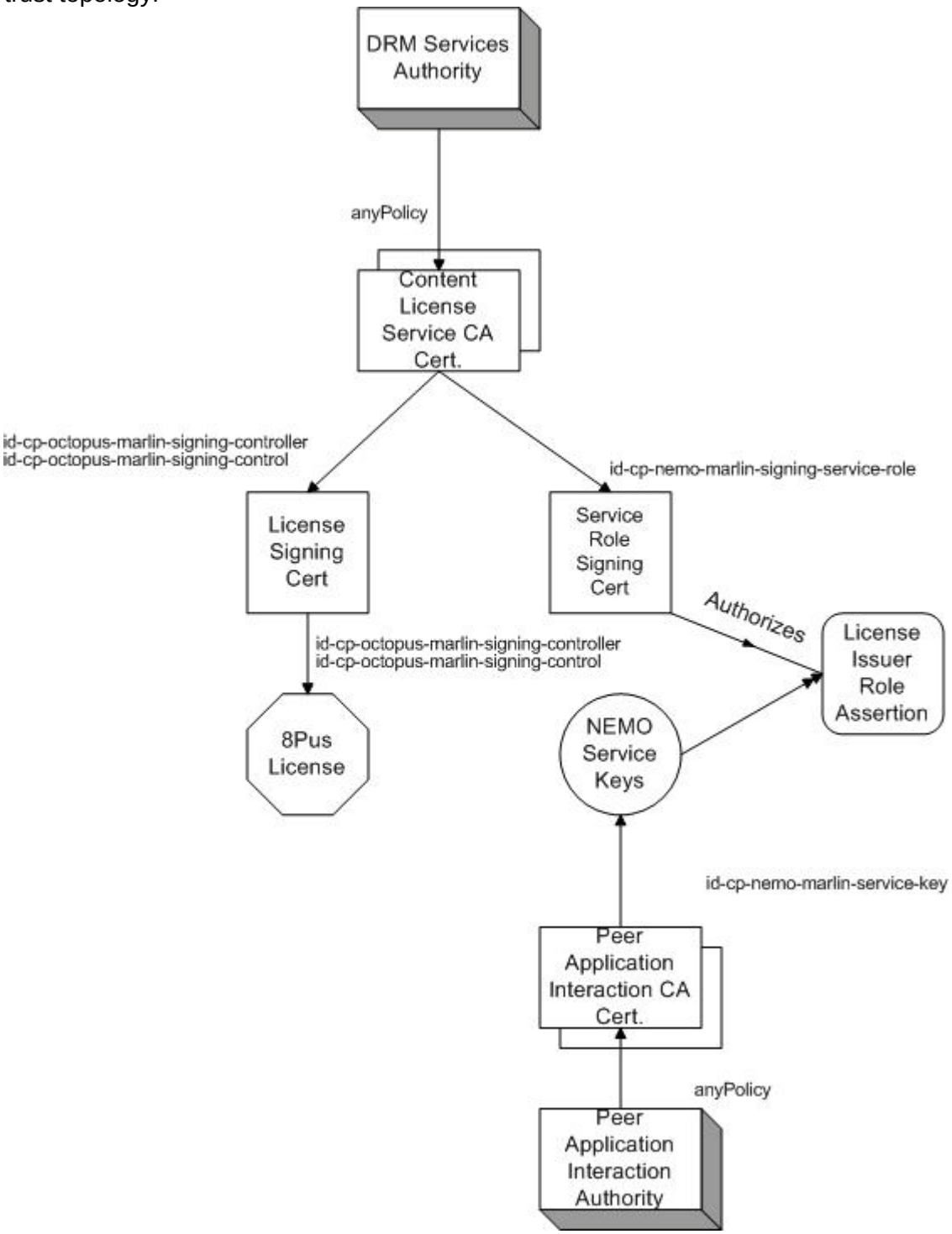


Figure 9-7 Content Licensing Trust Hierarchy

9.4.5.1 Content License Services Authority

The DRM Services Authority is used to authorize instances of a Marlin Content License Certification Authority. A Marlin Content License Certification Authority issues end-entity certificates used to operate a License Issuer service. That is, this authority is used to delegate to authorized entities the right to issue certificates which may be used to operate a License Issuer service.

Certificates that Authorize a Subordinate CA to Certify …	Basic Constraint	Key Usage	Certificate Policy Identifiers
NEMO Protocol and Service Interaction			
Keys for signing the service role	CA=TRUE pathLenConstraint>0	keyCertSign	Id-cp-nemo-marlin-signing-service-role
License Service			
License Service/Keys for signing License	CA=TRUE pathLenConstraint>0	keyCertSign	id-cp-octopus-marlin-signing-controller
			id-cp-octopus-marlin-signing-control

Table 9-10 Content License Services Authority Subordinate Certifying Keys

The certificates issued to the Content License Certification Authority SHOULD include a *nameConstraint* extension to constrain the name-space of the subjects (License Service) it certifies. For example, the keys for signing controller objects might be constrained such that only License Services certified within a particular name-space may be authorized to sign the controls of a DRM object.

9.4.5.2 Content License Certification Authority

Content which is protected is issued a license by an authorized License Issuer service. An instance of a Content License Certification Authority issues X.509 Public Key certificates to License Issuer services. These certified keys are used by the License Issuer to sign elements of an Octopus license; Controller and Control objects. The Content License Certification Authority MAY issue a single certificate with multiple policy identifiers in the certificate policy.

The end-entity signing certificates issued MUST have a certificate policy term that indicates the certified key is intended to be used to sign Controls and Controllers (*id-cp-octopus-marlin-signing-control* and *id-cp-octopus-marlin-signing-controller*, respectively).

Section 3.3.4 describes the mechanism by which Octopus objects are signed. As noted, Controls may be indirectly signed either by reference or as a result of being contained by the object being directly signed. Implementations SHALL take this into consideration when selecting acceptable policy terms. For example, when a Control is covered by secure digest of a referring Controller object, the acceptable policy terms would be *id-cp-octopus-marlin-signing-controller* AND *id-cp-octopus-marlin-signing-control*. Because the signature over the Controller object includes a secure hash of the Control, the identity of

3285 the Controller signer (extracted from the Subject distinguished name of the signers
3286 X.509 certificate) is also the signer of the Control.
3287

Key Used to Sign…	Basic Constraint	Key Usage	Certificate Policy Identifiers
License Issuer Service			
Licenses (Controller Objects)	CA=FALSE	digitalSignature	id-cp-octopus-marlin-signing-controller
			id-cp-octopus-marlin-signing-control

3288 **Table 9-11 License Issuer Service Signing Key(s)**

3289 This subordinate CA also issues certificates which may be used for protocol interactions
3290 and to sign License Issuer role attribute assertions.
3291

Key Used to Sign…	Basic Constraint	Key Usage	Certificate Policy Identifiers
License Issuer Service			
Keys for signing the service role	CA=FALSE	digitalSignature	Id-cp-nemo-marlin-signing-service-role

3292 **Table 9-12 License Issuer Role Signing Certificate**

3293

9.4.6 Data Certification Trust Hierarchy

The Marlin Data Certification Trust Hierarchy is responsible for the lifecycle and management of end-entity certificates used to assert authenticity of data relied upon by participants as part of a governance model, business model or service model.

The following diagram depicts the certificates and certified keys which are managed under this topology.

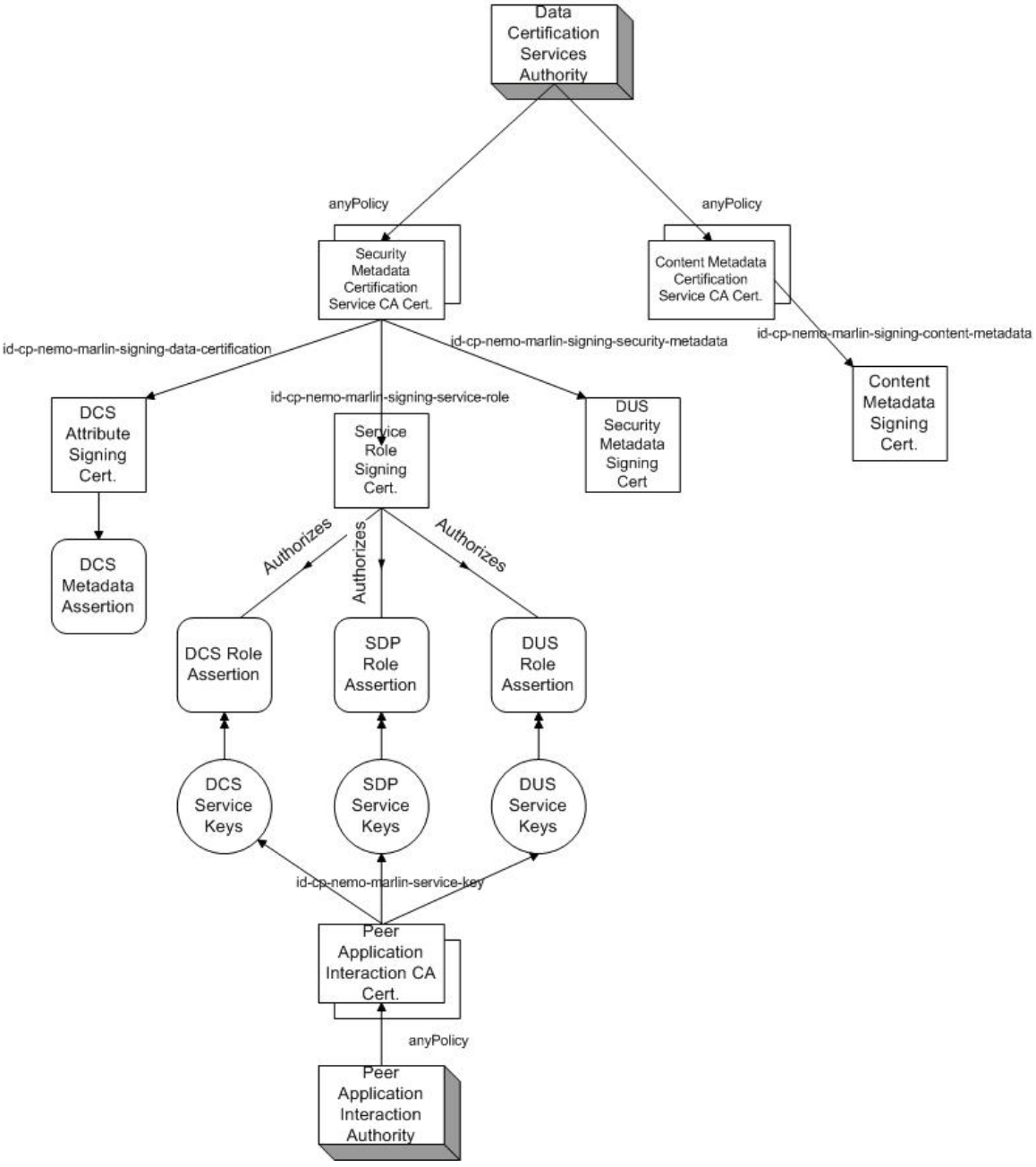


Figure 9-8 Data Certification Service Trust Hierarchy

9.4.6.1 Data Certification Services Authority

The Data Certification Services Authority may authorize instances of a two distinct Certification Authorities. These Certification Authorities are described separately so as to highlight their individual functions and characteristics. However, other factors which are out of scope for this document may influence whether this degree of separation is necessary. For example, compliance criteria or operational efficiencies may be best served by combining these CAs. This specification does not mandate separation or combination of the CAs but merely supplies an informative set of guidelines for constructing a trust topology.

The Data Certification Services Authority may authorize instances of a Security Metadata Certification Service Certification Authority and/or Content Metadata Certification Service Certification Authority.

Additionally this authority issues a certificate for the key used to sign indirect CRLs and Broadcast Key Blocks.

Key Used to Sign…	**Basic Constraint**	**Key Usage**
Indirect CRL and Broadcast Key Block	CA=FALSE	cRLSign, digitalSignature

Table 9-13 CRL and BKB Signing Certificate

A Content Metadata Certification Service Certification Authority issues end-entity certificates used to operate a Content Metadata Certification Service.

A Security Metadata Certification Service Certification Authority issues end-entity certificates used to operate Secure Metadata services such as the Security Data Provider defined by this specification. This specification also enables the operation of Data Certification Services. For example, Figure 9-8 depicts signing certificates for a Data Certification Service (DCS) and the Data Update Service (DUS.)

Certificates that Authorize a Subordinate CA to Certify …	**Basic Constraint**	**Key Usage**	**Certificate Policy Identifiers**
Security Metadata Service NEMO Interactions			
Keys for signing the service role	CA=TRUE pathLenConstraint>0	keyCertSign	Id-cp-nemo-marlin-signing-service-role
Keys for DCS Security Metadata			
Keys for signing Data Certification Assertion	CA=TRUE pathLenConstraint>0	keyCertSign	id-cp-nemo-marlin-signing-data-certification
Keys for Data Update Service Security Metadata			
Keys for signing	CA=TRUE	keyCertSign	id-cp-nemo-marlin-

security metadata items	pathLenConstraint>0		signing-security-metadata
Keys for Content Metadata			
Keys for signing content metadata items	CA=TRUE pathLenConstraint>0	keyCertSign	id-cp-nemo-marlin-signing-content-metadata

3330 **Table 9-14 Data Certification Services Authority Subordinate Certifying Keys**

3331 #### 9.4.6.2 Security Metadata Service Certification Authority

3332 The Data Certification Services Authority delegates, to a subordinate Security Metadata
3333 Certification Services Certification Authority, the authority to issue certificates used to
3334 sign attribute assertions and security metadata. Finally this CA issues certificates which
3335 are used to bind these security services to its role attribute
3336

Key Used to Sign…	Basic Constraint	Key Usage	Certificate Policy Identifiers
Security Metadata Role Signing Certificate			
Keys for signing the service role	CA=FALSE	digitalSignature	Id-cp-nemo-marlin-signing-service-role
Keys for Data Certification Service			
Keys for signing DCS Assertion	CA=FALSE	digitalSignature	id-cp-nemo-marlin-signing-data-certification
Keys for Data Update Service			
Keys for signing security metadata items	CA=FALSE	digitalSignature	id-cp-nemo-marlin-signing-security-metadata

3337 **Table 9-15 End-Entity Certificates for Security Metadata Services**

3338 #### 9.4.6.3 Content Metadata Certification Service Certification Authority

3339 The Data Certification Services authority delegates authority to a subordinate Content
3340 Metadata Certification Services Certification Authority to issue certificates used to
3341 ensure authenticity of metadata associated with content. The subordinate Content
3342 Metadata Certification Service CA issues end-entity certificates to instances of a service
3343 which provides this assurance (e.g. a content packager.)
3344
3345 The certificates issued to this entity MUST have a certificate policy term that indicates
3346 the certified key is intended to be used to sign content metadata (*id-cp-nemo-marlin-*
3347 *signing-content-metadata.*)
3348

Key Used to Sign…	Basic Constraint	Key Usage	Certificate Policy Identifiers
Keys for Content Metadata			
Keys for signing content	CA=FALSE	digitalSignature	id-cp-nemo-marlin-signing-content-metadata

Key Used to Sign…	Basic Constraint	Key Usage	Certificate Policy Identifiers
metadata items			

Table 9-16 End-Entity Certificates for signing content metadata

10 File Format for Marlin Content

In order to ensure at least a minimal set of interoperability between implementations of core system specifications, Marlin specifies a set of content file format profiles in terms of containers, media encryption mechanisms, codecs profiles, and metadata structures. However, the media file formats which Marlin supports change periodically and it is best to specify these formats independently of the core system specification. For that reason the supported media file formats are defined in a separate supporting specification, Marlin – File Formats Specification [MFF1.0].

11 Marlin Usage Rules

Marlin Usage Rules are encoded in Octopus Controls. However:

- Certain usage rules may require support of a protocol between two participating Marlin DRM Clients, a sequence of preliminary operations to prepare for the execution of a Control's *perform* method (see Move Action in Section 11.1.)
- Certain usage rules may require calling an application to perform some obligation after the Control's *perform* method has executed, including executing callbacks to the Octopus engine

Controls generally reference external objects such as Links, Node attributes, or state objects that are set by other System Calls or Controls.

In order to be interoperable, all core system implementations MUST implement the Octopus Controls specifications ([8pus] §3). This should enable all implementations to support such usage rules as:

Actions	High-Level Conditions
Play	Unconstrained Start/End Date Action Counts (per device, not domain) N times For a Period of H hours after the 1st Play Active User/Domain/Subscription link check Any combination of the above

In addition, all core system implementations SHOULD implement the following Marlin defined actions:

- Move (see Section 11.1)
- Copy (see Section 11.1)
- Export to DTCP
- Export to CPRM
- Export to VCPS
- Export to MG-R
- Export to Foreign DRM

[MEXP] defines the parameters required for a targeted export technology.

A license MAY indicate deviations from the default set of output control requirements. This SHALL be signaled using the mechanism defined by the Octopus Controls specification ([8pus] §3). All core system implementations SHOULD support the parameters defined in [MOCS1.0] to enable overriding default output controls.

Additional Usage Rules and supporting Actions and Obligations may be defined for individual Marlin Delivery System Specifications.

11.1 Move and Copy Actions

Moving and copying content to another device is supported using the Transfer action defined in the Octopus Standard Control Protocol 1.0 ([8pus] §3), in conjunction with the transfer protocol defined in Section 5.7.6 of this specification.

In addition to the standard Transfer action parameters, this specification defines the following parameter (visible under the host object /Octopus/Action/Parameters, as specified in [8pus] §3):

- Sink/Proximity/LastProbe = date of the most recent successful proximity probe from Source to Sink

11.1.1 Theory of Operation (Informative)

11.1.1.1 Overview

This mechanism may be extended to support other types of control, but is designed to support the following basic features:

- Before the Transfer operation, a content file C is "bound" to the Source device SRC (that is, the license L for C allows SRC to use the Content Key CK carried in L). The same license L does not allow the Sink device SNK to use CK).
- The content file C can be transferred to SNK as-is, without any special protection. This means that this Transfer operation does not require that the content file be re-encrypted.
- After the Transfer operation, the license L does not allow SRC to use CK anymore. The same license L allows SNK to use CK. The Control in L does not need to be re-generated or re-signed.
- This Transfer protocol can at least support the following policy: SRC can transfer C to SNK if:
    - L allows SRC to use CK (the license is "valid" on SRC) and SNK is a member of the same domain as SRC.

  Or
    - L allows SRC to use CK and SNK is in proximity of SRC

11.1.1.2 License Elements

The license L for a content file C that supports that type of Transfer semantics is constructed as follows:

- C is encrypted with CK
- The ContentKey Object (CKO) contains CK encrypted with the Scuba Sharing key SSK[SRC] of the Source device SRC's personality node.
- The Control is signed by the original Control creator (the import device most likely).

- The Control CTRL contains at least the Play and the Transfer actions.
    - The Play action is implemented in a manner similar to the following rule:
        - Play granted only if the SeaShell ([8pus] §7) state object /XXX/ContentTokens/C exists (where /XXX/ is some path prefix that represents a container in the SeaShell database into which the control can write).
    - The Transfer action is implemented in a manner similar to the following rule:
        - If the date of the last proximity probe is within acceptable time of the current date then:
            - Return a GRANTED status with an Obligation to run an Agent A1 with parameter P1 and context CTX1 on the Sink and a Callback request to call back Method CLBK when Agent A1 has been run on the Sink, with the result of the agent as input to the callback.

          Else
            - Return a GRANTED status with an Obligation to run an Agent A1 with parameter P2 on the Sink, and a Callback request to call back Method CLBK when Agent A1 has been run on the Sink, with the result of the agent as input to the callback.
        - In the callback method CLBK:
            - Read the agent's result from our input parameter
            - If the result indicates a success, then:
                - delete the SeaShell state object /XXX/ContentTokens/C and return a GRANTED status with no further callback or obligation. The protocol ends.

              else
                - The result indicates a failure, return a DENIED status with the appropriate extended status parameters.
        - The Agent A1 is a control that takes an input parameter indicating whether a certain domain membership is required or not (the domain membership node Id is included in the input parameter).
            - Require that the context host object CTX1 exist (this context host object MUST be set by the host as specified in the Transfer service protocol specification). Fail if the required context does not exist
            - If a domain membership is required, the control checks that the device in which it is running is a member of the specified

domain (typically, checks for a link reachability, and possibly some other conditions). If the device is not a member of the domain, a failure status is returned

- Create the SeaShell state object XXX/ContentTokens/C
- Return a success result

11.1.1.3 Sequence of Operations

a) The content file C is made available to the Sink.

b) The Sink initiates a Transfer protocol for content C with the Source.

c) The Sink provides its own node information (public node object, including signature and certificates) as well as the license bundle that governs C. The sink also optionally indicates whether it requires a re-encryption of the content keys or not.

d) The Source initiates the Transfer action on the license L for C with the sink's node info and proximity status as action input parameters by calling the Perform method.

e) When the return status of the Perform method indicates the initial GRANTED status with the obligation to run an Agent A1 on the sink, the Source sends A1 to the Sink and waits for the result.

f) Upon receiving the result of A1 from the Sink, the Source calls back the callback method specified in the Callback return parameter of the previous step, passing the result of the agent as an input parameter.

g) At this point, the callback method has returned an extended status block indicating if the protocol is a success or a failure. We assume it is a success.

h) The extended status block is sent to the sink along with the re-encrypted content keys if the status block indicates a success and the content keys were required in the request.

11.1.1.4 Agent

The agent included in the control for the license L has its own signature, created by the same entity as the one that created and signed the Controller for the license. This agent should be carried as an external extension of the Control object.

12 Profiles

This specification defines a variety of mechanisms by which system implementations can interact. In order to allow for flexibility and future functionality we may have defined multiple ways to achieve a similar functionality. However, interoperability may suffer if we do not narrow the choices. The following sections prescribe specific implementation guidance to aid developers and deployers.

12.1 Cryptographic Algorithm Profiles

This subsection prescribes which cryptographic, encoding and transformation algorithms are to be utilized.

12.1.1 Hashing (Digest) algorithms:

All implementations of Marlin SHALL support the following hashing algorithms:

- SHA1 [SHA1]
- SHA256 [SHA256]

12.1.2 Keyed-Hash Message Authentication Code algorithms

All implementations of Marlin SHALL support the following HMAC algorithm:

- Keyed-HMAC [hmacwithsha1]

12.1.3 Public Key algorithms

All implementations of Marlin SHALL support the following public key algorithms and key sizes:

- RSA 1.5 with 1024 and 2048 keys [RSA-1_5]
- RSA-OAEP [RSA-1_5]]

12.1.4 Signature Hash algorithms

All implementations of Marlin SHALL support the following hash functions in combination with the above public key algorithms:

- SHA1
- SHA256

The hash function of the signature algorithm MUST be consistent with the hash algorithm used in computing the individual digests. For example, if the signature algorithm is RSA with SHA1 then the digest method must also be SHA1.

12.1.5 Symmetric key algorithms

All implementations of Marlin SHALL support the following algorithms:

- AES 128 bit symmetric key [AES] in CBC mode [AES-MODES]
- AES 128 bit symmetric key [AES] in CTR mode [AES-MODES, ISMACryp] §10.0. [10]

[10] Note 128 bit AES in counter mode is only required by applications which encrypt/decrypt content.

12.1.6 Canonicalization

All implementations SHALL implement the following canonicalization methods

- [xml-exc-c14n]

12.2 XML Digital Signature Profile

12.2.1 <ds:Signature> Element

All signatures MUST be detached. There MAY be multiple signatures covering the same object.

The <ds:Signature> block MUST contain:

- A <ds:SignedInfo> element
- A <ds:SignatureValue> element
- A <ds:KeyInfo> element

12.2.2 <ds:SignedInfo>

The <ds:SignedInfo> MUST embed the following elements:

12.2.2.1 <ds:CanonicalizationMethod>

The <ds:CanonicalizationMethod> element is empty and its ds:Algorithm attribute MUST signal the use of exclusive XML canonicalization using the identifier defined in [xml-exc-c14n] §4.

12.2.2.2 <ds:SignatureMethod>

When a keyed HMAC algorithm is used for computing the signature, the <ds:SignatureMethod> element SHALL be empty and implementations SHALL ignore any spurious child elements carried in the <ds:SignatureMethod> element. The ds:Algorithm attribute of the <ds:SignatureMethod> element SHALL identify the signature algorithm and the hash function it utilizes. The hash functhion MUST be either SHA1 or SHA256. The identifiers are specified in [xmldsig] and [RFC4051], respectively. As a convenience to the reader these identifiers are repeated here:

http://www.w3.org/2000/09/xmldsig#hmac-sha1
http://www.w3.org/2001/04/xmldsig-more#hmac-sha256

When an RSA based signature algorithm is used the ds:Algorithm attribute SHALL identify the signature algorithm and the hash function it utilizes. The hash functhion MUST be either SHA1 or SHA256. The identifiers are specified in [xmldsig] and [RFC4051], respectively. As a convenience to the reader these identifiers are repeated here:

http://www.w3.org/2000/09/xmldsig#rsa-sha1
http://www.w3.org/2001/04/xmldsig-more#rsa-sha256

The hash function of the signature algorithm MUST be consistent with the hash algorithm used in computing the individual digests. For example, if the signature method is RSA with SHA1 then the digest method must also be SHA1.

12.2.2.3 <ds:Reference>

There MAY be one or more <ds:Reference> elements inside the <ds:SignedInfo> block if more than one object is signed by the same key. For example, signing an Octopus Control and an Octopus Controller with the same key.

When signing an Octopus object, the value of the ds:URI attribute MUST be the oct:Id attribute of the element carrying the encoding of the referenced Octopus object.
NOTE:

- It MUST be done this way instead of using the Uid attribute (real ID of the Octopus objects) because the production of xs:ID is not compatible with the URN syntax (xs:ID is defined as a non-colonized name.)

When signing a local XML element the value of the ds:URI attribute MUST be the value of the element's xs:ID-typed attribute (Note that the Octopus schema names this attribute oct:Id.)

When a reference is to an Octopus object, the digest in the reference MUST be the canonical byte sequence of the Octopus object as specified in [8pus] §5.

The canonicalization of Octopus objects MUST be indicated in a <ds:Tranforms> element. An example follows:

```
<ds:Tranforms>
  <ds:Transform
   Algorithm="http://www.octopus-drm.com/octopus/specs/cbs-1_0"/>
</ds:Tranforms>
```

No other <ds:Tranform> is allowed for Octopus object references.

The canonicalization for all other signed elements MUST use the exclusive canonicalization transform defined in [xml-exc-c14n]. An example follows:

```
<ds:Tranforms>
 <ds:Transform Algorithm="http://www.w3.org/2001/10/xml-exc-c14n#"/>
</ds:Tranforms>
```

12.2.2.3.1 <ds:DigestMethod>

The <ds:DigestMethod> element SHALL be empty and its ds:Algorithm attribute SHALL identify the hash function it utilizes. The hash functhion MUST be either SHA1 or SHA256. The identifiers are specified in [xmldsig] and [xmlenc], respectively. As a convenience to the reader these identifiers are repeated here:

http://www.w3.org/2000/09/xmldsig#sha1
http://www.w3.org/2001/04/xmlenc#sha256

The hash function of the signature method MUST be consistent with the hash algorithm used in computing the individual digests. For example, if the signature algorithm is RSA with SHA1 then the digest method must also be SHA1.

12.2.2.3.2 <ds:DigestValue>

The <ds:DigestValue> element MUST contain the base64 encoded value of the digest.

12.2.3 <ds:SignatureValue>

The signature value MUST be the base64 encoded value of the signature of the canonicalized ([xml-exc-c14n]) <ds:SignedInfo> element with the key described in the <ds:KeyInfo> element.

12.2.4 <ds:KeyInfo>

12.2.4.1 HMAC Signatures

When signing with HMAC, the <ds:KeyInfo> element MUST have a single child, <ds:KeyName>, that indicates the key used for computing the HMAC signature. The guidance given in Section 12.2.2.2 regarding HMAC algorithm identification SHALL be followed.

For HMAC signatures over Octopus objects this identifier MUST be the oct:uid attribute of the Octopus secret key object used to compute the signature.

Example:

```
<ds:KeyInfo>
  <ds:KeyName>urn:x-octopus:secret-key:1001</ds:KeyName>
</ds:KeyInfo>
```

12.2.4.2 Public Key Signatures

When signing with a public key algorithm, the public key used to verify the signature MUST be carried in an X.509 v3 certificate, and MUST be accompanied by other certificates necessary to complete the certificate path to a trust anchor.

These certificates MUST be carried, encoded in base64, in <ds:X509Certificate> elements. These <ds:X509Certificate> elements are embedded in an <ds:X509Data> element and this <ds:X509Data> element MUST be a child of the <ds:KeyInfo> element. The <ds:X509Certificate> elements MUST appear in sequential order, starting from the end-entity's signing key certificate. The certificate of the trust anchor SHOULD be omitted (since it can not necessarily be determined to be trusted.)

The guidance given in Section 12.2.2.2 regarding Public Key algorithm identification SHALL be followed.

```
<ds:KeyInfo>
  <ds:X509Data>
    <!-- cert of the end-entity's signing public key -->
    <ds:X509Certificate>MIICh...</ds:X509Certificate>
    <!-- intermediate cert to the trust anchor -->
    <ds:X509Certificate>MIICo...</ds:X509Certificate>
</ds:X509Data>
</ds:KeyInfo>
```

12.3 NEMO Profile for Basic Secure Messaging

This Profile defines a profile of Basic Secure Messaging defined in [NEMO] §3.

The full security version of the NEMO basic secure messaging protocol provides confidentiality, integrity, and freshness protection of NEMO service request, response, and confirmation messages. The balance of this section focuses on defining a profile of these messages to simplify implementations and to further assure interoperability.

12.3.1 Notation

The entities `&nemo;` `&nemoc;` and `nemosec;` are defined so as to provide shorthand identifiers for URIs defined in this specification. For example

```
&nemosec;/Element
```

corresponds to

```
http://nemo.intertrust.com/2005/10/security/Element
```

This entity is used as a shorthand notation in this specification; however, the use of entities in NEMO is specified in NEMO Message Bindings [NEMO] §2.

12.3.2 Request Message

12.3.2.1 SOAP Header

12.3.2.1.1 Action (Informative)

The <wsa:Action> element shall contain a same value specified in a soapAction attribute of the corresponding WSDL.

12.3.2.1.2 Message ID for the Message

The <S11:Header> element SHALL contain a <wsa:MessageID> element.

12.3.2.1.3 Correlation to Request Message

The <S11:Header> element SHALL NOT contain a <wsa:RelatesTo> element.

12.3.2.2 Security Header

In this profile, the security header SHALL also contain the requestor's long-term public message encryption key certificate, SAML assertions [SAML1.1] and a profile identifier in addition to elements which are described in [NEMO] §3.1.4.1.1.1.

The Marlin System Root SHALL NOT be included in PKIPath certificate chains. Trust Anchors SHALL NOT be included in PKIPath certificate chains.

12.3.2.2.1 Protocol Identifier

The <nemosec:ProtocolDeclaration> element SHALL include a <nemosec:Step> element with Type attribute containing the string value defined in [NEMO] §3.1.4.1.1.1.1. The <nemosec:Step> element SHALL NOT include an Index attribute.

12.3.2.2.2 Profile

The <wsse:Security> element SHALL contain a <nemosec:Profile> element to signal the profile defined by this document. Hence, the <nemosec:Profile> element SHALL include a URI attribute having the value

urn:marlin:core:1.0:nemo:protocol:profile:1

The <nemosec:Profile> element SHALL contain a nemosec:Usage attribute with the value defined in [NEMO] §3.1.3.5.6. The <nemosec:Profile> element SHALL be the immediate sibling element of the <nemosec:ProtocolDeclaration> element.

12.3.2.2.3 Requestor's Timestamp

The <wsu:Timestamp> element SHALL contain a nemosec:Usage attribute with the value defined in [NEMO] §3.1.4.1.1.1.2. The <wsu:Expire> element SHALL NOT be used in <wsu:Timestamp> element.

12.3.2.2.4 Requestor's Nonce

The <wsse:Nonce> element SHALL contain a nemosec:Usage attribute with the value defined in [NEMO] §3.1.4.1.1.1.3.

12.3.2.2.5 Responder's Identifier

The <nemosec:ToNode> element SHALL contain a nemosec:Usage attribute with the value defined in [NEMO] §3.1.4.1.1.1.5.

12.3.2.2.6 Requestor's Identifier

The <wsse:Security> element SHALL NOT contain a <nemosec:FromNode> element.

12.3.2.2.7 Self-encrypted Message Key

The <wsse:BinarySecurityToken> element SHALL contain a ValueType attribute as defined in [NEMO] §3.1.3.5.7. The <wsse:BinarySecurityToken> element SHALL contain a nemosec:Usage attribute with a value defined in [NEMO] §3.1.4.1.1.1.6. Element encryption SHALL include the <wsse:BinarySecurityToken> element and its contents.

12.3.2.2.8 Requestor's Public Signing Key Certificate Chain

The <wsse:Security> element SHALL contain a <wsse:BinarySecurityToken> with a ValueType attribute of

http://docs.oasis-open.org/wss/2004/01/oasis-200401-wss-x509-token-profile-1.0#X509PKIPathv1

In other words, following value type SHALL NOT be used.

http://docs.oasis-open.org/wss/2004/01/oasis-200401-wss-x509-token-profile-1.0#PKCS7

The <wsse:BinarySecurityToken> element SHALL contain a nemosec:Usage attribute with a value defined in [NEMO] §3.1.4.1.1.1.7.

12.3.2.2.9 Requestor's Public Encryption Key Certificate Chain

The <wsse:Security> element SHALL contain a <wsse:BinarySecurityToken> with a ValueType attribute of

http://docs.oasis-open.org/wss/2004/01/oasis-200401-wss-x509-token-profile-1.0#X509PKIPathv1

In other words, following value type SHALL NOT be used.
http://docs.oasis-open.org/wss/2004/01/oasis-200401-wss-x509-token-profile-1.0#PKCS7

The <wsse:BinarySecurityToken> element SHALL contain a nemosec:Usage attribute with a value defined in [NEMO] §3.1.4.1.1.1.8.

12.3.2.2.10 Requestor's Encrypted Message Encryption Key

Requestor's Message Encryption Key SHALL be used to encrypt <S11:Body> element and <ds:Signature> element. Element encryption SHALL include the <ds:Signature> element and its contents. Element encryption SHALL include all the contents of the <S11:Body> element. The Message Encryption Key SHOULD be unique for each message (Request, Response, and Confirmation) of the transaction. However, a message processor need not verify the uniqueness of the keys across the transaction.

12.3.2.2.11 Requestor's role attribute assertions

Requestor's role attribute assertion required by the service policy SHALL be placed as a direct child element of a <wsse:Security> element, and SHALL be referenced from a single <wsse:KeyIdentifier> element in a <wsse:SecurityTokenReference> element as specified in [WS-SECSAML] §3.3. Thus, when multiple SAML Assertions are sent in a request message, multiple <wsse:SecurityTokenReference> elements are used to refer to respective SAML Assertions.

The <wsse:SecurityTokenReference> element that references a SAML attribute assertion that asserts a NEMO node role SHALL contain a nemosec:Usage attribute with the value defined in [NEMO] §3.1.3.5.8.

12.3.2.2.12 Signature

In addition to elements described in [NEMO] §3.1.4.1.1.1.10, the signature SHALL cover the following elements

- Profile
- Message ID for this message
- Requestor's public encryption key certificate chain
- <S11:Body>

The <ds:Signature> element SHALL include a <ds:KeyInfo> element. The <ds:KeyInfo> element SHALL include a <wsse:SecurityTokenReference> element which refers to the requestor's signing key certificate. This reference MUST use <wsse:Reference> element. The <wsse:Reference> element SHALL contain URI attribute which refers to the <wsse:BinarySecurityToken> by a bare name.

12.3.2.3 Processing Rules

In addition to the processing rules defined in [NEMO] §3.1.4.1.1.2,

- The responder SHALL verify the validity of the requestor's certificate.
- The responder SHALL check that the NEMO node ID included in the requestor's role attribute assertion is identical to a requestor's node ID included in the requestor's signing certificate.

- The responder SHALL verify the requestor has the proper role attribute (see Table 4-1).

12.3.3 Response Message

12.3.3.1 SOAP Header

12.3.3.1.1 Action (Informative)

The <wsa:Action> element shall contain a same value specified in a soapAction attribute of the corresponding WSDL.

12.3.3.1.2 Message ID for the Message

The <S11:Header> element SHALL contain a <wsa:MessageID> element.

12.3.3.1.3 Correlation to Request Message

The <S11:Header> element SHALL contain a <wsa:RelatesTo> element. The <RelatesTo> element SHALL include RelationshipType attribute having the value

&nemo;/addressing/originatesFrom

and SHALL contain a value which is identical to the value contained in the Message ID element of the request message.

12.3.3.2 Security Header

In this profile, the security header SHALL also contain a profile identifier in addition to elements which are described in [NEMO] §3.1.4.1.2.1.

The Marlin System Root SHALL NOT be included in PKIPath certificate chains. Trust Anchors SHALL NOT be included in PKIPath certificate chains.

12.3.3.2.1 Protocol Identifier

The <nemosec:ProtocolDeclaration> element SHALL include a Step element with Type attribute containing the string value defined in [NEMO] §3.1.4.1.2.1.1. The Step element SHALL NOT include an Index attribute. If a fault occurs related to the security protocol defined in [NEMO] §3, the value of the Type attribute SHALL be set to "fault". Otherwise, the value of the Type attribute SHALL be set to "response".

12.3.3.2.2 Profile

The <wsse:Security> element SHALL contain a < nemosec:Profile> element to signal the profile defined in this document. Hence the <nemosec:Profile> element SHALL include a URI attribute having the value

urn:marlin:core:1.0:nemo:protocol:profile:1

The <nemosec:Profile> element SHALL contain a nemosec:Usage attribute with the value defined in [NEMO] §3.1.3.5.6. The <nemosec:Profile> element SHALL be the immediate sibling element of the <nemosec:ProtocolDeclaration> element.

12.3.3.2.3 Responder's Timestamp

The <wsu:Timestamp> element SHALL contain a nemosec:Usage attribute with the value defined in [NEMO] §3.1.4.1.2.1.2. The <wsu:Expire> element SHALL NOT be used in <wsu:Timestamp> element.

12.3.3.2.4 Requestor's Nonce

The <wsse:Nonce> element SHALL contain a nemosec:Usage attribute with the value defined in [NEMO] §3.1.4.1.2.1.6.

12.3.3.2.5 Responder's Nonce

The <wsse:Nonce> element SHALL contain a nemosec:Usage attribute with the value defined in [NEMO] §3.1.4.1.2.1.7.

12.3.3.2.6 Responder's Identifier

The <wsse:Security> element SHALL NOT contain a <nemosec:FromNode> element.

12.3.3.2.7 Requestor's Identifier

The < nemosec:ToNode> element SHALL contain a nemosec:Usage attribute with the value defined in [NEMO] §3.1.4.1.2.1.4.

12.3.3.2.8 Self-encrypted Message Key

The <wsse:BinarySecurityToken> element SHALL contain a ValueType attribute as defined in [NEMO] §3.1.3.5.7. The <wsse:BinarySecurityToken> element SHALL contain a nemosec:Usage attribute with a value defined in [NEMO] §3.1.4.1.2.1.5. Element encryption SHALL include the <wsse:BinarySecurityToken> element and its contents.

12.3.3.2.9 Responder's Public Signing Key Certificate Chain

The <wsse:Security> element SHALL NOT contain a <wsse:BinarySecurityToken> for a responder's public signing key certificate.

12.3.3.2.10 Responder's Encrypted Message Encryption Key

Responder's Message Encryption Key SHALL be used to encrypt <S11:Body> element and <ds:Signature> element. Element encryption SHALL include the <ds:Signature> element and its contents. Element encryption SHALL include all the contents of the <S11:Body> element. The Message Encryption Key SHOULD be unique for each message (Request, Response, and Confirmation) of the transaction. However, a message processor need not verify the uniqueness of the keys across the transaction.

The <xenc:EncryptedKey> element SHALL include <ds:KeyInfo> element. The <ds:KeyInfo> element SHALL include <wsse:SecurityTokenReference> element which refers to the requestor's encryption key certificate by using <wsse:KeyIdentifier> element. The <wsse:KeyIdentifier> element SHALL use the Subject Key Identifier to reference the X.509 certificate as defined in [WS-SECX509] §3.2.1. The ValueType attribute SHALL signal this with the value defined in [WS-SECX509-ER] §5.

12.3.3.2.11 Signature

In addition to elements described in [NEMO] §3.1.4.1.2.1.10, the signature SHALL cover the following elements:

- Profile

- Message ID for this message
- Correlation to request message
- <S11:Body>

When <nemoc:FaultDetails> element is included in a message and integrity protection is applied to the message the <nemoc:FaultDetails> element SHALL be signed.

The <ds:Signature> element SHALL include a <ds:KeyInfo> element. The <ds:KeyInfo> element SHALL include a <wsse:SecurityTokenReference> element which refers to the responder's signing key certificate. This reference MUST use <wsse:KeyIdentifier> element. The <wsse:KeyIdentifier> element SHALL use the Subject Key Identifier to reference the X.509 certificate as defined in [WS-SECX509] §3.2.1. The ValueType attribute SHALL signal this with the value defined in [WS-SECX509-ER] §5.

12.3.3.3 Processing Rules

In addition to the processing rules defined in [NEMO] §3.1.4.1.2.2,

- The requestor SHALL verify the validity of the responder's certificate.
- The requestor SHALL verify that the NEMO node ID in the responder's role attribute assertion identifies the same subject as is indicated in the responder's certificate.
- The requestor SHALL verify the responder has the proper role attribute (see Table 4-1).

12.3.3.4 Fault Response

In the event a fault is returned in the response, the fault message MUST follow the rules given in Sections 12.3.3.1.2, 12.3.3.1.3 and 12.3.3.2.11.

12.3.4 Confirmation Message

12.3.4.1 SOAP Header

12.3.4.1.1 Action (Informative)

The <wsa:Action> element shall contain a same value specified in a soapAction attribute of the corresponding WSDL.

12.3.4.1.2 Message ID for the Message

The <S11:Header> element SHALL contain a <wsa:MessageID> element.

12.3.4.1.3 Correlation to Request Message

The <S11:Header> element SHALL contain a <wsa:RelatesTo> element. The <wsa:RelatesTo> element SHALL include RelationshipType attribute having the value

```
&nemo;/addressing/originatesFrom
```

and SHALL contain a value which is identical to the value contained in the Message ID of the request message.

12.3.4.2 Security Header

In addition to elements which are described in [NEMO] §3.1.4.1.3.1, the security header SHALL contain a profile identifier.

The Marlin System Root SHALL NOT be included in PKIPath certificate chains. Trust Anchors SHALL NOT be included in PKIPath certificate chains.

12.3.4.2.1 Protocol Identifier

The <nemosec:ProtocolDeclaration> element SHALL include a Step element with Type attribute containing the string value defined in [NEMO] §3.1.4.1.3.1.1.
The Step element SHALL NOT include Index attribute.

12.3.4.2.2 Profile

The <wsse:Security> element SHALL contain a < nemosec:Profile> element to signal the profile defined in this document. Hence, the <nemosec:Profile> element SHALL include an URI attribute having the value

```
urn:marlin:core:1.0:nemo:protocol:profile:1
```

The <nemosec:Profile> element SHALL contain a nemosec:Usage attribute with the value defined in [NEMO] §3.1.3.5.6.
The <nemosec:Profile> element SHALL be the immediate sibling element of the <nemosec:ProtocolDeclaration> element.

12.3.4.2.3 Requestor's Timestamp

The <wsu:Timestamp> element SHALL contain a nemosec:Usage attribute with the value defined in [NEMO] §3.1.4.1.3.1.2. The <wsu:Expire> element SHALL NOT be used in <wsu:Timestamp> element.

12.3.4.2.4 Responder's Nonce

The <wsse:Nonce> element SHALL contain a nemosec:Usage attribute with the value defined in [NEMO] §3.1.4.1.3.1.3.

12.3.4.2.5 Requestor's Identifier

The <wsse:Security> element SHALL NOT contain a <nemosec:FromNode> element.

12.3.4.2.6 Responder's Identifier

The < nemosec:ToNode> element SHALL contain a nemosec:Usage attribute with the value defined in [NEMO] §3.1.4.1.3.1.5.

12.3.4.2.7 Self-encrypted Message Key

The <wsse:BinarySecurityToken> element SHALL contain a ValueType attribute as defined in [NEMO] §3.1.3.5.7. The <wsse:BinarySecurityToken> element SHALL contain a nemosec:Usage attribute with a value defined in [NEMO] §3.1.4.1.3.1.6.
Element encryption SHALL include the <wsse:BinarySecurityToken> element and its contents.

12.3.4.2.8 Requestor's Public Signing Key Certificate Chain

The <wsse:Security> element SHALL contain a <wsse:BinarySecurityToken> with a ValueType attribute of

```
http://docs.oasis-open.org/wss/2004/01/oasis-200401-wss-x509-
token-profile-1.0#X509PKIPathv1
```

In other words, the following value type SHALL NOT be used.

```
http://docs.oasis-open.org/wss/2004/01/oasis-200401-wss-x509-
token-profile-1.0#PKCS7
```

The <wsse:BinarySecurityToken> element SHALL contain a nemosec:Usage attribute with a value defined in [NEMO] §3.1.4.1.3.1.8.

12.3.4.2.9 Requestor's Encrypted Message Encryption Key

Requestor's Message Encryption Key SHALL be used to encrypt the child elements of the <S11:Body> and <ds:Signature> element. Element encryption SHALL include the <ds:Signature> element and its contents. Element encryption SHALL include all the contents of the <S11:Body> element. The Message Encryption Key SHOULD be unique for each message (Request, Response, and Confirmation) of the transaction. However, a message processor need not verify the uniqueness of the keys across the transaction.

12.3.4.2.10 Signature

In addition to elements described in [NEMO] §3.1.4.1.3.1.9, the signature SHALL cover the following elements:

- Profile
- Message ID for this message
- Correlation to request message
- <S11:Body>

The <ds:Signature> element SHALL include a <ds:KeyInfo> element. The <ds:KeyInfo> element SHALL include a <wsse:SecurityTokenReference> element which refers to the requestor's signing key certificate. This reference MUST use <wsse:Reference> element. The <wsse:Reference> element SHALL contain URI attribute which refers to the <wsse:BinarySecurityToken> by a bare name.
Note that the signing key is to be repeated in the confirmation, rather than to rely on server state.

12.3.4.3 Processing Rules

- The responder SHALL check the responder's nonce to see if it is the same as the nonce that was passed to the requestor in the corresponding response message.

The responder SHALL check that the value included in <wsa:RelatesTo> element of the confirmation message is the same as the value included in <wsa:MessageID> element of the request message.

12.3.5 License Transfer Protocol Correlation Processing Rules

The License Transfer Protocol (LTP) involves a series of messages initiated by the Sink and terminated by the Source. Between the initial setup message and the terminating teardown message there are potentially multiple rounds of Sink Request and Source Response messages. The following processing rules augment this profile to ensure proper message correlation.

The initial Sink Setup Request message and the Source RunAgent Response message MUST adhere to the correlation techniques specified in Sections 12.3.2 and 12.3.3, respectively.

Subequent messages are a series of Sink Request and Source Response messages. An AgentResult is sent from the Sink to the Source in Request messages. A RunAgent or Teardown is sent from the Source to the Sink in Response messages.

These subsequent Sink Request messages MUST be constructed like the Confirmation Message defined in Section 12.3.4 but with the following changes to facilitate the proper correlation of the multi-round exchanges.

12.3.5.1 Message ID for the Message

The <S11:Header> element SHALL contain a <wsa:MessageID> element.

12.3.5.2 Correlation to Source Response Message

The <S11:Header> element SHALL contain a <wsa:RelatesTo> element. The <wsa:RelatesTo> element SHALL include RelationshipType attribute of

```
http://www.w3.org/2005/08/addressing/reply
```

as defined in [WS-addr] and SHALL contain a value which is identical to the value contained in the MessageID of the Source Response message being replied to.

12.3.5.3 Requestor's Nonce

The <wsse:Nonce> element SHALL contain a nemosec:Usage attribute with the value defined in [NEMO] §3.1.4.1.3.1.3:

```
#confirmation-returnedNonce
```

and SHALL contain a value which is identical to the responder's nonce contained in the Response message being replied to.

12.4 SAML Assertion Profile

Marlin utilizes the Security Assertion Markup Language (SAML) for conveying a variety of information. SAML is extremely flexible insofar as what type of information can be conveyed within an assertion and the number of statements embodied by an assertion. The following subsections prescribe simple limitations on how SAML assertions are constructed so as to simplify the processing of the assertions and to enable interoperability and accuracy of the information distilled from an assertion.

12.4.1 Assertion Conditions

SAML 1.1 allows for an optional <saml:Conditions> element which when processed may influence the validity of the assertion. Presently we do not require any conditions in this profile. However, it is expected that temporal attributes (saml:NotOnOrAfter and saml:NotBefore) are used. If these attributes are present then the processing rules specified in [SAML1.1] §2.3.2.1.1 SHALL apply.

In general, Marlin uses assertions to reflect some fact that persists for a period longer than the initial issueance and consumption of the assertion. Hence, the <saml:DoNotCache> element SHOULD NOT be used for this type of assertion.

12.4.2 Assertion Subject

SAML 1.1 allows for an assertion to contain multiple statements. This leads to the possibility that a given assertion may have multiple unique subjects. There are no specific usage scenarios in Marlin which requires this flexibility. Therefore, it is RECOMMENDED that all statements in an assertion refer to the same subject.

12.4.3 Attributes

In Marlin, SAML assertions are primarily used to convey trusted attributes about a system entity (typically a Nemo node.) A DRM Client is NOT REQUIRED to check whether required attributes appear in the assertion but it MAY return an error if a required attribute is not in the assertion.

Attributes are qualified by their origin namespace so that two attributes with the same name may be unambiguously described. However, SAML does not preclude two or more fully qualified attributes from appearing in either the same statement or in multiple statements within an assertion. It is RECOMMENDED that no two fully qualified attributes appear multiple times within an assertion.

The type of attribute values that can be issued in SAML assertions is restricted to the following XML data types:

- xs:string
- xs:int
- xs:duration
- xs:dateTime
- xs:base64Binary
- xs:nonNegativeInteger

No complex type is allowed. If the type is not indicated, it SHALL be processed as an xs:string value. The XML data type xs:nonNegativeInteger SHOULD be used whenever the attribute is used to convey version information.

12.4.4 Subject Confirmation

According to [WS-SECSAML] §3.1, systems are required to implement the processing necessary to support subject confirmation methods (i.e. holder-of-key and sender-vouches).

In Marlin, subject confirmation processing is realized as described in Section 0. Therefore, the <saml:SubjectConfirmation> element SHALL NOT be specified in a SAML assertion.

12.4.5 Signature

All assertions MUST be signed so as to enable verification of their authenticity. The guidance give in Section 12.2 SHALL be followed in addition to the guidance given in [SAML1.1] §5.

12.5 Name Management Profile

A general description of the rules for specifying identifiers was given in Section 1.3. In this section we give guidance to specific naming conventions used to identify the various system objects.

12.5.1 SeaShell Object Ownership

The SeaShell database specification (see [8pus] §7.5) defines different ways of determining control program identities. In Marlin, the identities of the Control program SHALL be acquired from the Subject distinguished name found in a signer's X.509 certificate. Octopus allows for multiple signatures so there may be more than one identity attributed to a Control. Controls may be signed directly or indirectly. When a Control is indirectly signed (see Sections 3.3.4, 9.3.2.1 and 9.3.2.2) then the identities of a Control SHALL also include each of the indirect signers.

12.5.1.1 Container Delegate

A Delegate is a Plankton byte code routine, located in a Control object that performs a task on behalf of another routine, located in a different Control object.

A Delegates primary use is for making it possible for a control to access SeaShell objects that belong to a different principal than the principal that signed the control. This is necessary, because SeaShell access control is based on the principal identifiers of the signers of the Control that contains the code making the SeaShell call.

The path of the root container of the SeaShell database is

/Octopus/SeaShell/Databases/Marlin

This container is owned by the principal

urn:marlin:drmservices:seashell

To bootstrap the process of assigning ownership of descendant containers a signing certificate must be created and issued under the DRM Services (the issuer DN=urn:marlin:drmservices) trust hierarchy. This certificate would be used to sign Delegate Controls which create sub-containers and reassign ownership.

This certificate would have the following attributes:

Delegate Signing Key	Basic Constraint	Key Usage	Certificate Policy Identifiers
DRM Services Container Delegate Signing			
Delegate Control	CA=FALSE	digitalSignature	id-cp-octopus-marlin-signing-control

12.5.2 Octopus Naming

Octopus object names referenced via System.Host.Get/SetObject() fall into one of three categories. The first category, Octopus standard names, are object names that allow read only access to objects where Octopus specifications define a standard naming path such as /Octopus/Personality/Attributes. The second category is SeaShell database names. Third is Marlin standard names that are object names that allow read only access to objects defined by Marlin standards. Names in the first two categories use the Octopus defined path followed by a Marlin defined URN. The third category uses a Marlin specific extension.

Category	Prefix	Example
Octopus	<octopus-prefix>/urn:marlin:<rest of name>	/Octopus/Personality/Attributes/urn:marlin:foo
SeaShell DB	/Octopus/SeaShell/Databases/Marlin	/Octopus/SeaShell/Databases/Marlin/SomeCompany/foo
Marlin	/Marlin	/Marlin/LicenseRevocations/SomeCompany/IdList/foo

All path components that follow the Marlin component are implicitly from the Marlin standards name space unless a subsequent name component is a URN in which case all following name components are implicitly from the namespace of that URN.

URN identifiers that are referenced via a System.Host.GetObject call SHALL NOT use "/" characters as "/" is a reserved character within the namespace specific string of a URN and as the path separator for object paths. If a reserved character is to be represented then the %-encoding defined in [URN] §2.2 MUST be used to encode the URN portion of the object path.

Organizations SHOULD use their own domain name for URLs.

12.5.3 Extensions

Entity specific extensions can be defined. These extensions can only be referenced by SomeOrganization's proprietary actions because only SomeOrganization clients SHALL make this path available. Non SomeOrganization clients SHALL NOT execute a SomeOrganization proprietary action.

Object Path
/SomeOrganization

12.5.4 SeaShell Database ([8pus] §7)

The pattern for Marlin global database is as follows.

Object Path
/Octopus/SeaShell/Databases/Marlin

The pattern for Marlin service specific database is as follows.

Object Path
/Octopus/SeaShell/Databases/Marlin/<service container name>

12.5.4.1 Marlin Core

Selected information from Marlin assertions SHALL be made available for access via System.Host.GetObject. Since a client may have multiple distinct instances of assertions (e.g. role attribute assertions), the naming path includes an index (index origin 0).

Pattern	Example
/Marlin/Assertions/@<index>/<item-path>	/Marlin/Assertions/@0/Attributes/foo

Marlin core defines a naming pattern for accessing the IssueInstant element of an assertion and for accessing the attribute values in the assertion.

Pattern
/Marlin/Assertions/@<index>/IssueInstant
/Marlin/Assertions/@<index>/Attributes/<attribute-name>

12.5.4.2 Role attribute assertions

According to the NEMO Trust Management Bindings specification ([NEMO] §4), each role SHALL be represented as an attributes within a SAML assertion and multiple roles MAY be expressed within the same SAML assertion. A client may play many roles in Marlin.

We rely on the namespace equivalence defined in Table 1-3 to simplify object path composition. In general, the path composition for <attribute-name> is accomplished by taking the value of the saml:AttributeNamespace attribute, appending a ":" and then appending value of the saml:AttributeName attribute.

Attribute Namespace	Name	Object Path	Value Type
urn:marlin:nemo:2004:attribute	role	/Marlin/Assertions/@<index>/Attributes/urn:marlin:nemo:2004:attribute:role	String

Table 12-1 Role Attribute Namespace/Name

By using the equivalence we avoid the need to URL encode the attribute namespace when using it to construct an object path.

The identifiers for the various role attribute values defined by this specification are enumerated in Table 4-2.

12.5.4.3 Marlin Security Specification Attribute

The Marlin security specification version attribute represents the security versions of the various Marlin specifications the client supports. Control programs can encode a check for the security specification and version. In the event that the security of a version of the specification is deemed compromised, new licenses may encode the requirement for the new version of the specification. Also, during peer to peer (or client to server) interactions, a Nemo node can use the security specification versions in the role attribute assertion of the entity it is interacting with to determine which security versions of the specifications (and thus what features) the entity supports

If a client is updated and the supported specifications change, a new role attribute assertion for the client can be acquired that accurately reflects the new capabilities. Compliance rules require that the implementation of the client not allow the role attribute assertion to be referenced (e.g. via System.Host.GetObject) or sent in part of a Nemo protocol if the specifications and versions in the assertion do not accurately reflect the capabilities of the client.

Attribute Namespace	Name	Object Path	Value Type
urn:marlin:core	version-major	/Marlin/Assertions/@<index>/Attributes/urn:marlin:core:version-major	nonNegativeInteger
urn:marlin:core	version-minor	/Marlin/Assertions/@<index>/Attributes/urn:marlin:core:version-minor	nonNegativeInteger

12.5.4.4 Capabilities

There are many roles in Marlin. Each role is represented by a distinct role attribute. An entity implementing a given role may also implement optional capabilities. This specification defines an attribute namespace and standard values to convey the capabilities of an entity implementing a particular role.

If a component is updated and the set of capabilities changes, a new role attribute assertion for the related roles filled by the component can be acquired that accurately reflects the new capabilities.

.

The following table enumerates the capability attributes which may be attributed to for a entity assigned the *urn:marlin:core:role:drm-client* role.

Attribute Namespace	Name	Object Path
urn:marlin:core:client:capabilities	trusted-time	/Marlin/Assertions/@<index>/Attributes/urn:marlin:core:client:capabilities:trusted-time

The presence of the trusted-time attribute indicates that the client implements the System.Host.GetTrustedTime function in compliance with the Marlin compliance rules regarding trusted time.
Control programs may query the capabilities of the role attribute assertion to determine if specific actions are supported.

12.5.4.5 Role and Capability Example (Informative)

The previous sections described the various attributes which are conveyed within an assertion for a given role. The following example depicts a fragment of an SAML 1.1 assertion carrying this information.

```
<saml:AttributeStatement
 xmlns="urn:oasis:names:tc:SAML:1.0:assertion">
 <saml:Subject>
  <saml:NameIdentifier
    Format=
     "http://nemo.intertrust.com/2004/saml/name-format/uri">
    urn:marlin:organization:phony:fuse:nemo:personality:00000001
  </saml:NameIdentifier>
 </saml:Subject>
 <saml:Attribute
    AttributeNamespace="urn:marlin:nemo:2004:attribute"
    AttributeName="role">
   <saml:AttributeValue>
     urn:marlin:core:role:drm-client</saml:AttributeValue>
 </saml:Attribute>
 <saml:Attribute AttributeNamespace="urn:marlin:core"
            AttributeName="version-major">
   <saml:AttributeValue
     xsi:type="xsd:nonNegativeInteger">1</saml:AttributeValue>
 </saml:Attribute>
 <saml:Attribute AttributeNamespace="urn:marlin:core"
            AttributeName="version-minor">
   <saml:AttributeValue
     xsi:type="xsd:nonNegativeInteger">1</saml:AttributeValue>
 </saml:Attribute>
 <saml:Attribute
   AttributeNamespace="urn:marlin:core:client:capabilities"
   AttributeName="trusted-time">
   <saml:AttributeValue/>
 </saml:Attribute>
</saml:AttributeStatement>
```

12.6 Type Mapping of Host Objects

12.6.1 Mapping XML Types to Host Objects

Marlin defines mechanisms by which to make data visible to Plankton programs as Host Objects. Whenever this data is transported as XML encoded objects (e.g., SAML Assertion) the following table defines the type mapping. If not listed below all other types SHALL be treated as a String.

XML Data Type	Host Object Data Type
<xs:string/>	String
<xs:int/>	Integer
<xs:duration/>	Integer {converted to number of minutes}
<xs:dateTime/>	Integer {converted to number of minutes elapsed since Jan 1, 1970 00:00:00}
<xs:base64Binary/>	Byte Array
<xs:nonNegativeInteger/>	Integer

Table 12-2 XML to Host Object Data Types

12.6.2 Mapping Octopus Object Attributes to Host Objects

Marlin defines mechanisms by which to transport Octopus Object Attributes and make them visible to Plankton programs as Host Objects. The following table defines the type mapping, XML mixed content encoding for <oct:Attribute/> elements and the Host Object representation.

Octopus Object Attribute Type	<oct:Attribute/> XML 'type' attribute	<oct:Attribute/> Mixed Content	Host Object Data Type
[Named Attributes]	n/a	n/a	Host Object of the same name

Octopus Object Attribute Type	<oct:Attribute/> XML 'type' attribute	<oct:Attribute/> Mixed Content	Host Object Data Type
[Unnamed Attributes]	n/a	n/a	Host Object with no name
IntegerAttribute	int	string	Integer
StringAttribute	string	string	String
ByteArrayAttribute	bytes	base64Binary	Byte Array
ListAttribute	list	<oct:AttributeList/>	Container
ArrayAttribute	array	<oct:AttributeArray/>	Container

Table 12-3 Octopus Object Attributes to Host Object Data Types

12.6.3 Mapping Agent Parameters to Host Objects

Agent parameters are transported as XML encoded objects and made visible to Plankton programs as Host Objects. Agent parameters are transported in an <mc:AgentCarrier/> element. The parameters are a list of one or more <pk:ParameterBlock/> elements. Mapping a <pk:ParameterBlock/> to a Host Object is done using the following rules:

- The name attribute of the <pk:ParameterBlock/> maps to a Host Object name.
- The value of each parameter is expressed as a <pk:ValueBlock/> element.

The following table defines the <pk:ValueBlock/> type mapping, XML mixed content encoding for <pk:ValueBlock/> elements and Host Object representation.

<pk:ValueBlock/> Type	<pk:ValueBlock/> Mixed Content	Host Object Data Type
Integer	string	Integer
String	string	String
Date	string	Integer {converted to number of minutes elapsed since Jan 1, 1970 00:00:00}
ByteArray	base64Binary	Byte Array
Parameter	<pk:ParameterBlock/>	see ValueList
ValueList	<pk:ValueListBlock/>	Container

Table 12-4 Agent Parameters to Host Object Data Types

The <pk:ValueBlock/> types ExtendedParameter, Real and Resource MUST NOT be used. For each <pk:ValueBlock/> value in the list there is a corresponding Host Object in the container. If the value is a <pk:ParameterBlock/>, then it maps to a named Host Object according to the mapping rules for <pk:ParameterBlock/> elements. Other <pk:ValueBlock/> types map to an unnamed Host Object according to the mapping rules for <pk:ValueBlock/> elements as defined in Table 12-3.

The following example depicts XML encoded Agent Parameters.

```
<ValueListBlock>
  <ValueBlock type="Parameter">
    <ParameterBlock name="ContainerWithNames">
      <ValueBlock type="ValueList">
        <ValueListBlock>
          <ValueBlock type="Parameter">
            <ParameterBlock name="Color">
              <ValueBlock
                type="String">Red</ValueBlock>
            </ParameterBlock>
          </ValueBlock>
          <ValueBlock type="Parameter">
            <ParameterBlock name="Size">
              <ValueBlock
                type="Integer">37</ValueBlock>
            </ParameterBlock>
          </ValueBlock>
        </ValueListBlock>
```

```
            </ValueBlock>
          </ParameterBlock>
        </ValueBlock>
        <ValueBlock type="Parameter">
          <ParameterBlock name="ContainerWithoutNames">
            <ValueBlock type="ValueList">
              <ValueListBlock>
                <ValueBlock type="String">Blue</ValueBlock>
                <ValueBlock type="Integer">512</ValueBlock>
              </ValueListBlock>
            </ValueBlock>
          </ParameterBlock>
        </ValueBlock>
        <ValueBlock type="Parameter">
          <ParameterBlock name="SomeInteger">
            <ValueBlock type="Integer">156</ValueBlock>
          </ParameterBlock>
        </ValueBlock>
        <ValueBlock type="Parameter">
          <ParameterBlock name="SomeString">
            <ValueBlock type="String">Hello</ValueBlock>
          </ParameterBlock>
        </ValueBlock>
        <ValueBlock type="Parameter">
          <ParameterBlock name="SomeBytes">
            <ValueBlock type="ByteArray">SGVsbG8K</ValueBlock>
          </ParameterBlock>
        </ValueBlock>
      </ValueListBlock>
```

The corresponding Host Object tree follows:

```
[ContainerWithNames]
   Color="Red"
   Size=37

[ContainerWithoutNames?]
   Blue
   512

SomeInteger=156
SomeString="Hello"
SomeBytes={0x48, 0x65, 0x6c, 0x6c, 0x6f, 0x0a}
```

12.7 XML Profile

12.7.1 XML Attribute Composition Constraints

Validating XML parsers are not mandated nor recommended by this specification. Thus it is assumed that the XML parser of an implementation does not know the declared type of an XML attribute when performing Attribute-Value Normalization ([XML-1-1] §3.3.3). To ensure interoperability, constraints on the composition of XML attributes are imposed

as follows. XML attributes SHALL NOT contain leading or trailing white space (#x20, #xD, #xA, #x9) characters or sequences of more than one white space character.

12.7.2 Using QNames

Validating XML parsers are not mandated nor recommended by this specification. To ensure unambiguous treatment of attribute values and element content the following constraints on the use of QNames is adopted.

QNames SHALL only be used with XML Element and Attribute Names in conformance with [XMLns, QNAMEIDS]. Particularly, when an XML element instance explicitly asserts its type with the xsi:type attribute, the value of the xsi:type attribute SHALL be processed as a string identifier.
The following table defines string identifiers which can be used in the xsi:type attribute value to indicate the XML data types ([Schema]).

String Identifier	XML Data Type
xsd:string	xs:string
xsd:int	xs:int
xsd:duration	xs:duration
xsd:dateTime	xs:dateTime
xsd:base64Binary	xs:base64Binary
xsd:nonNegativeInteger	xs:nonNegativeInteger

12.8 Security Metadata Propagation (Informative)

The Marlin core specification defines a variety of security metadata types. Some of this metadata is provisioned at manufacturing time or conveyed to relying parties via a service interface such as a Security Data Provider. The security metadata provisioned at manufacture may be out-of-date before the device is sold to the consumer.

Keeping a device current can be a challenge in some deployment scenarios. Marlin core devices do not presume a specific network transport technology. For example, a Marlin core device may access a transport via a direct tethered connection to another device. Another example is a set of devices in a home environment using IP protocols but absent of a routable Internet connection. Even though Marlin core devices may operate in this limited setting they still require current security metadata to enable governance. The intent of the informative text is to describe alternative implementation techniques by which security metadata can be acquired and disseminated.

In the following sections we describe the traditional mechanism by which security metadata can be refreshed. Within that section we also discuss how a Security Data

Provider Service can collect (and by its service interface distribute) security metadata by Marlin specified mechanisms. Finally we describe alternative sources of security metadata and how a Security Data Provider Service might leverage these alternate sources to update the security metadata it distributes.

12.8.1 Common Security Metadata Acquisition Mechanisms

The primary mechanism by which Marlin core specification enables distribution of security metadata is through the Security Data Provider service. This service may have online access so that it can acquire new security metadata where as clients of the service may not. The obvious example is a tethered device. To support its clients the Security Data Provider service could periodically retrieve updated security metadata.

To support this deployment environment a Security Data Provider service can resolve a CRL Distribution Point to retrieve an updated CRL and supply it to the tethered device the next time it is connected. The same is true for the Broadcast Key Block since it defines a set of distribution URIs which may be resolved to retrieve a new Broadcast Key Block. Since both the CRL and the BKB are integrity protected by the issuer and contains version information the Security Data Provider can keep current.

12.8.2 Alternate Security Metadata Acquisition Mechanisms

There are some less obvious mechanisms by which a Security Data Provider service might acquire updated security metadata. This section describes some of the possible out-of-band mechanism which might be applied by an implementation.

Content is made available to a DRM Client in many ways and for all intents and purposes the DRM client is decoupled from the delivery system. Therefore a DRM Client might aggregate content from different delivery systems. That is, the content may source from both online and offline delivery systems. More current security metadata may be attached to content which sources from an online delivery system.

With that in mind it is worth noting that the Broadcast Key Block (BKB) is essentially attached to a content license when the license is created. DRM Clients must evaluate this license when they attempt to render content. A DRM Client could assist a Security Data Provider service by detaching the most current BKB it encounters and then make the BKB available to the Security Data Provider service. A DRM Client already has a well defined core service which it must expose which could be used to get the BKB to the Security Data Provider, the DRM Client Information service. The payload of the DRM Client Information service has a specific interface to retrieve security metadata from a DRM Client.

On a similar note, a DRM Client may be provisioned at manufacture time with the most up to date version of the BKB. Thus by using the same alternative propagation scheme described above a Security Data Provider can obtain a fresher BKB.

There is also the possibility for the Security Data Provider service to acquire security metadata out-of-band. For example, in a broadcast distribution setting it may be possible to embed security metadata in a transport stream. The device receiving the broadcast could extract the security metadata and make it available to a co-resident implementation of a Security Data Provider service.

Another out-of-band mechanism that could be utilized in a broadcast environment could be to acquire current time of day from time of day services in the cable operator's data network. Depending on the operational security of the cable operator’s data network this source of time might be considered trusted.

Appendix A: XML Schemas File Names

- A.1: proximity-check.xsd
- A.2: drm-client-information.xsd
- A.3: domain-manager-information.xsd
- A.4: provide-drm-object.xsd
- A.5: provide-security-data.xsd
- A.6: license-transfer.xsd
- A.7: marlin-nemo-core-exceptions.xsd

Appendix B: WSDLs File Names

- B.1: proximity-check.wsdl
- B.2: drm-client-information.wsdl
- B.3: domain-manager-information.wsdl
- B.4: provide-drm-object.wsdl
- B.5: provide-security-data.wsdl
- B.6: license-transfer.wsdl