Marlin – Simple Secure Streaming Specification

Version 1.4
FINAL

Source	Marlin Developer Community
Date	January 25, 2017

Copyright (c) Marlin Developer Community, 2003-2017. All Rights Reserved
Refer to Notices on page 2 for important legal information

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.

Copyright © 2003 - 2014 by MDC, 415-112 North Mary Avenue #383 Sunnyvale, CA 94085, USA. All rights reserved. Third-party brands and names are the property of their respective owners.

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

Contents

1 Introduction

This document describes a simple and secure solution to enable a media streaming service to authenticate a streaming client to consume content. This specification presents a solution that re-uses existing standards such as HTTP and Transport Layer Security (TLS) to deliver information to the authenticated client.

1.1 Document Organization

This document is organized as follows:

- (This) introduction, including abbreviations, definitions and references.
- An overview
- Transport Layer Security (TLS) setup and the definition of Stream Access Statement (SAS)
- Triggering MS3
- Handling of content and SAS

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].

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 in some cases the normative text of this specification imposes constraints beyond those indicated by the schema documents.

1.3.1 Identifiers

The protocol version communicated between an MS3 Client and MS3 Service reflects the specification version implemented by the client. The following table summarizes the protocol identifier and its value defined in this version of specification:

Protocol Identifier	Version
MS3 Version	1.0
MS3 Version	1.2

URI “urn:marlin:ms3:1-0” indicates the compatibility to version 1.0 and 1.1 of this specification, with protocol version 1.0 supported. URI “urn:marlin:ms3:1-2” indicates the compatibility to version 1.2, 1.3 and 1.4 of this specification, with protocol version 1.2 supported.

1.3.2 Namespaces and Notation

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

Prefix	XML Namespace	Description
ms3:	urn:marlin:ms3:1-0:services:schemas:streaming:action-token	See §6.2

The table below summarizes the external schemas used in this specification:

Prefix	XML Namespace	Description
bsa:	urn:marlin:broadband:1-2:nemo:services:action-token	[MBB] See §6.1
xsi:	http://www.w3.org/2001/XMLSchema-instance	[Schema]

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.

1.4 Data Structures and Types Notation

1.4.1 Notation

The abstract type notation used in this document uses the syntax: <name>: <type>, where <type> is of the form: <value-type> (size-in-bits) for single values, <value-type> (size-in-bits) [array-size] for arrays of values, or { … } for compound data structures.
The notation <type> [n] means an array of <n> elements of type <type>. The notation <type> [] means an array with a variable number of elements of type <type>.

1.4.2 Bit/Byte Order

All data in this specification are presented with the most significant bit (or byte) on the left hand side and the least significant bit (or byte) on the right hand side.
Also, all data in this specification are encoded using the big-endian byte order (also known as network byte order) and all bit vectors are multiples of 8 bit bytes in big-endian byte order.

1.5 Abbreviations

1.5.1 List of Abbreviations

AT	Action Token
BT	Business Token
CDN	Content Distribution Network
C-URIT	URI Template for Content URL
C-URL	Content URL
MS3	Marlin Simple Secure Streaming
MIME	Multipurpose Internet Mail Extensions
NEMO	Networked Environment for Media Orchestration
SAS	Stream Access Statement
S-URL	Stream Access Statement URL
TLS	Transport Layer Security

1.6 Terms and Definitions

Client	The Client consists of Media Player and MS3 Client.

Copyright (c) Marlin Developer Community, 2003-2017. All Rights Reserved
Refer to Notices on page 2 for important legal information

Compound URI	A combined encoding of the S-URL and C-URIT parameters, in the form of S-URL “#” C-URIT.
MS3 Client	Implementation receiving and using Stream Access Statements to gain access to, and allow rendering of, content.
MS3 Service	Service that supplies Stream Access Statements to MS3 Clients

Please refer to the Terms and Definitions introduced in [MBB].

1.7 References

1.7.1 List of referenced documents

Normative References

[MBB]	Marlin Broadband Delivery System Specification, Version1.2
[MCS]	Marlin – Core System Specification, Version1.3
[MOC]	Marlin – Output Control Specification, Version1.0
[MURIT]	URI Templates for Marlin, Version 1.0
[HTTP]	R. Fielding, J. Gettys, J. Mogul, et. Al., Hypertext Transfer Protocol -- HTTP/1.1. RFC 2616. http://www.ietf.org/rfc/rfc2616.txt
[HTTPTLS]	HTTP Over TLS, IETF RFC 2818. http://www.ietf.org/rfc/rfc2818.txt
[PKIX]	R. Housley, W. Polk, W. Ford, D. Solo. Internet X.509 Public Key Infrastructure Certificate and Certificate Revocation List (CRL) Profile. RFC 3280. http://www.ietf.org/rfc/rfc3280.txt
[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.
[RFC4281]	The Codecs Parameter for "Bucket" Media Types, IETF RFC 4281, November 2005. http://www.ietf.org/rfc/rfc4281.txt
[RFC5234]	Crocker, D., Ed. and P. Overell, "Augmented BNF for Syntax Specifications: ABNF", STD 68, RFC 5234, January 2008. http://www.ietf.org/rfc/rfc5234.txt
[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/
[TLS]	The Transport Layer Security (TLS) Protocol version 1.2, IETF RFC 5246
[TLSAES]	AES Ciphersuites for TLS, IETF RFC 3268. http://www.ietf.org/rfc/rfc3268.txt
[TLSAES-2]	E. Rescorla. TLS Elliptic Curve Cipher Suites with SHA-256/384 and AES Galois Counter Mode (GCM). RFC 5289. http://www.ietf.org/rfc/rfc5289.txt
[FIPS186]	NIST, "Digital Signature Standard (DSS)", FIPS PUB 186-4, July 2013, <http://dx.doi.org/10.6028/NIST.FIPS.186-4>.
[URI]	T. Berners-Lee, R. Fielding, L. Masinter. Uniform Resource Identifier (URI): Generic Syntax. RFC 3986. http://www.ietf.org/rfc/rfc3986.txt

[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

2 Overview (Informative)

Figure 1 provides an architectural overview of Marlin Simple Secure Streaming (MS3) technology for delivering a Stream Access Statement (SAS) to MS3 Clients via Transport Layer Security (TLS). Note the 1.0 protocol version corresponding to this figure, as this document introduces new HTTP based and HTTPS based technologies for delivering the SAS.

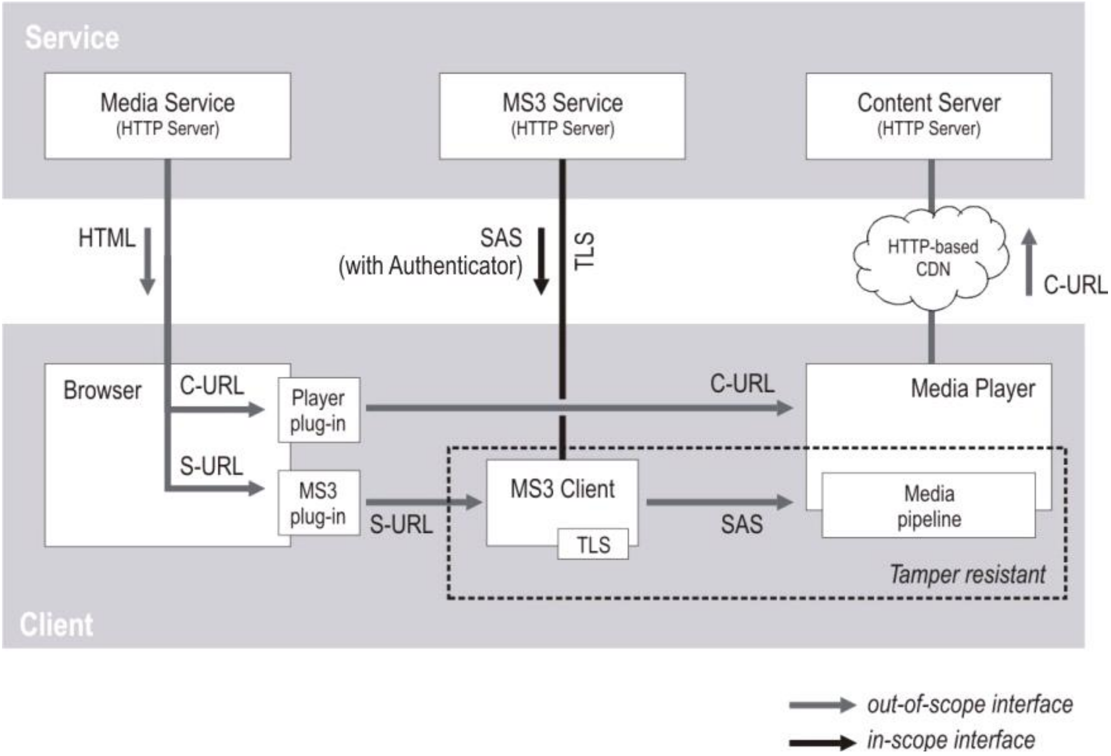


Figure 1: Architectural Overview of MS3 (protocol version 1.0)

In this figure, the Media Service supplies the Browser with content location information (C-URL) and the location of an MS3 Service (S-URL). An MS3 Service supplies Stream Access Statements to authorized clients. An SAS contains information required to acquire and consume the content referenced by the C-URL. The mechanism by which the Media Service delivers this information to a browser is out of scope for this specification, but some possible techniques are described in §4.

Also the internal architecture of the client is out-of-scope for this specification, but logically a browser plug-in forwards the S-URL and the C-URL to the MS3 Client and the Media Player respectively. The Media Player uses the C-URL to obtain the content from the Content Server, potentially via a Content Distribution Network (CDN), and passes the stream into its media processing pipeline.

Control information is required to render the (encrypted or plaintext) stream. To get this control information, the Media Player relies on the MS3 Client to securely resolve the S-URL with the MS3 Service. The S-URL embeds transaction context information (such as a Business Token [MBB]) required by the MS3 Service to respond to this request. A successful run of this protocol exchange delivers an SAS to an authorized MS3 Client. Typically an SAS contains the content key in the case of protected content and output control flags, and this information in used by the media processing pipeline to consume the content stream and enforce output controls. The SAS optionally contains an Authenticator to further ensure that access to the content is limited to the authorized client, i.e., only the client with possession of the content key and the authenticator can obtain and render the content.

2.1 Handling of Unencrypted Content

In some markets it is considered sufficiently secure to control access to a certain Content resource instead of encrypting the Content. In these cases the URL from which the Content is retrieved from the CDN typically embeds an Authenticator and is given only to a client that is entitled to have access to the Content and trusted to handle this URL and the Content as intended by the Service. When the URL is used by the Client to retrieve the Content from the CDN, the Authenticator is parsed by the CDN and used to ensure that access to the resource is limited in some way. The resource may for example only be served a limited number of times, within a limited timeframe or to a specific Client IP address.

It is NOT in scope for this specification to specify an access control mechanism or define the Authenticator. However this specification may be used to securely authenticate a Client, deliver an (opaque) Authenticator and associate an SAS with content that is not encrypted. The architecture is depicted in Figure 2.

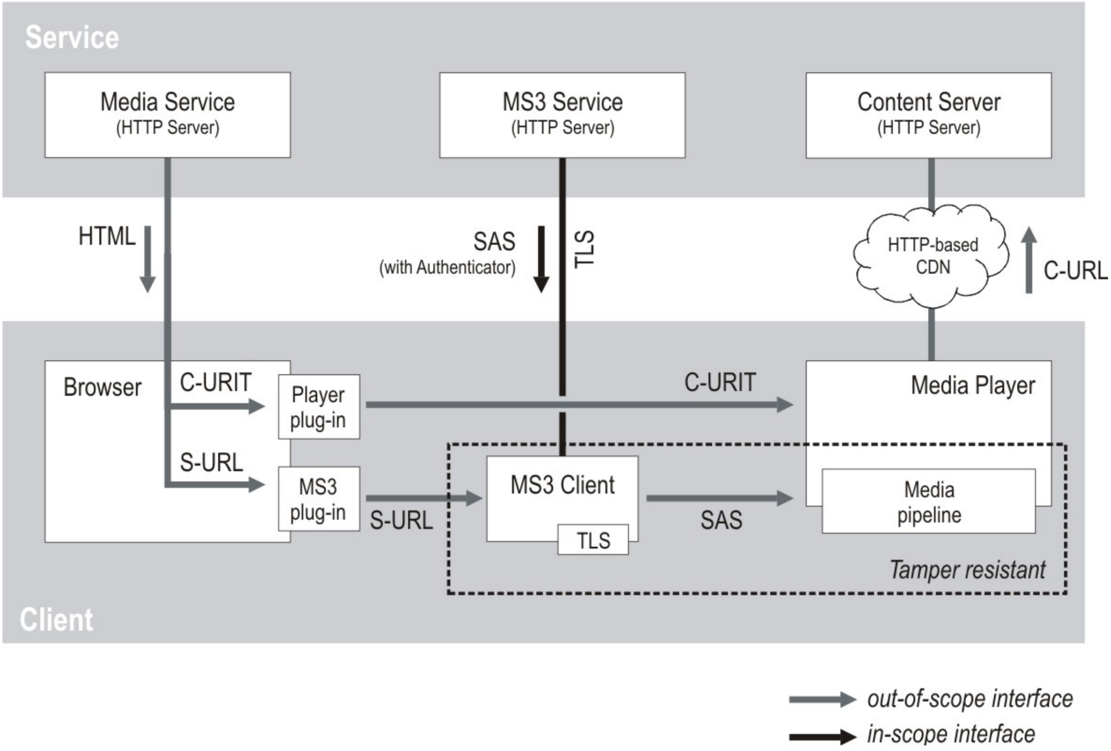
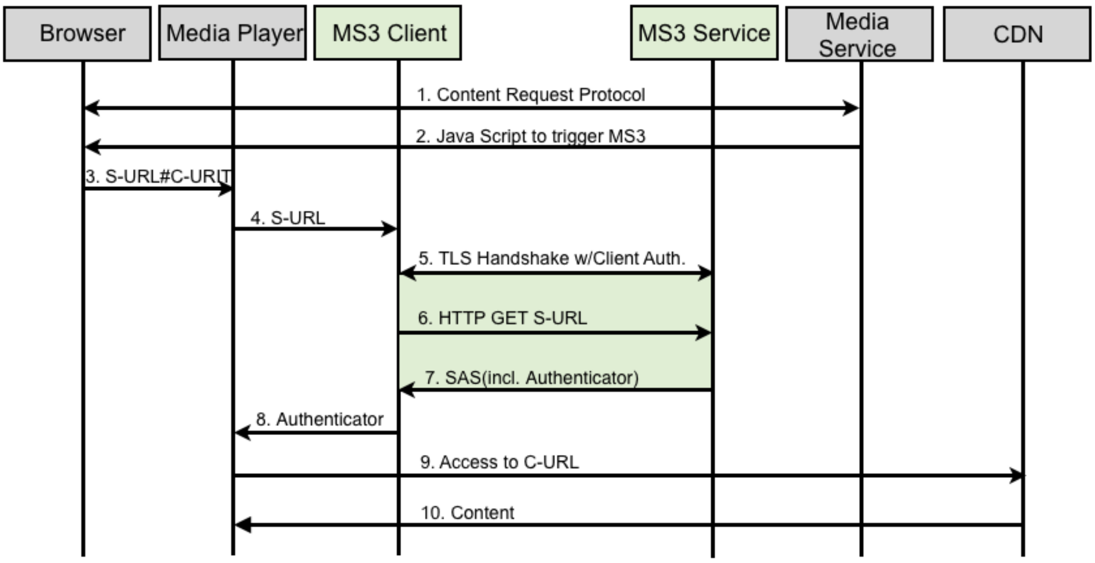


Figure 2: Architecture for unencrypted Content (protocol version 1.0)

Instead of sending S-URL and C-URL to the Browser, the Media Service can send an S-URL and content location information that consists of a URI Template (C-URIT) to the Browser. The MS3 Client then resolves the S-URL to obtain the SAS containing an Authenticator. The Authenticator is then used to fill in the URI Template in the C-URIT in order to obtain an opaque C-URL. This ensures secure delivery of the Authenticator and handling of the Content is in compliance with this specification.

2.2 Protocol flow

The application protocol binding that an MS3 Client engages in to request or purchase a given content item is outside the scope of this specification. However, once the service is triggered to request an SAS, the MS3 Client engages in the HTTP binding defined in this specification so as to acquire an SAS and access the corresponding content. The following figure depicts the general application protocol flow.



Legend	
S-URL#C-URIT	Compound URI
S-URL	Stream Access Service URL
C-URIT	URI Template for C-URL
C-URL	Content URL derived from C-URIT with supplied Authenticator
SAS	Stream Access Statement

Figure 3: Protocol Sequence Diagram

1. The Browser communicates with the Media Server to request content for playback. The mechanism by which this is accomplished is outside the scope of this specification
2. The Media Service supplies the Browser with an S-URL and C-URIT.
3. The Browser passes the S-URL and C-URIT to the Media player.
4. The Media Player initiates the MS3 Client with the S-URL. The mechanism by which this is accomplished is out of scope for this specification.
5. The client MAY establish a TLS session with the MS3 service. This will depend on the protocol version and URI scheme.
6. The MS3 Client resolves the S-URL with the MS3 Service.
7. Given the request from MS3 Client, the MS3 Service sends an SAS in the response.
8. When an MS3 Client receives the successful response, the SAS could contain an Authenticator. MS3 Client passes the Authenticator and usage information to the Media Player.
9. The client accesses to the content resource (e.g. CDN) by resolving C-URL.
10. A successful response from resolving C-URL results in the content corresponding to the SAS acquired in the step 7.

The scope of this specification is the syntax and encoding of the service and content location information, the protocol interface between the MS3 Service and the MS3 Client and the semantics of the SAS.

Copyright (c) Marlin Developer Community, 2003-2017. All Rights Reserved
Refer to Notices on page 2 for important legal information

3 MS3 Protocol

The MS3 protocol defined in this specification is designed to be simple to implement. The protocol uses HTTP (with or without TLS) to securely deliver a SAS to an authorized receiver.

For protocol version 1.0 the MS3 service location URL (S-URL) SHALL be formatted with one of the following URI schemes:

- the “https” URI scheme, as specified in §2.4 of [HTTPTLS] or,
- the “ms3” URI scheme, as defined in §3.4.2.1 of this specification. The “ms3” URI scheme SHALL only be used with a Compound URI. It is RECOMMENDED to use the “ms3” URI scheme (in lieu of the “https” URI scheme) whenever a Compound URI is used.

For protocol version 1.2 the MS3 service location URL (S-URL) SHALL be formatted with one of the following URI schemes:

- the “ms3h” URI scheme, when SAS request is made via POST over HTTP
- the “ms3hs” URI scheme, when the request is made over HTTPS without client authentication
- the “ms3hsa” URI scheme, when the request is made over HTTPS with client authentication using the client’s NEMO certificate

Note that no “http” or “https” URI scheme is provided for the schemes introduced in this specification

Each S-URL SHALL be a unique identifier that logically resolves to an SAS. The entity that constructs the S-URL SHALL ensure that there is negligible probability that the same identifier (S-URL) will resolve to a different data object (SAS).

The MS3 protocol SHALL consist of three steps:

1. For protocol version 1.0, or version 1.2 with the “ms3hsa” URI scheme, setup a mutually authenticated TLS session as specified in [TLS] using the TLS profile as defined in §3.1 or resume a previous session established as in §3.2 where client and server both implement the server state-less session resumption protocol defined in https://tools.ietf.org/html/rfc5077, or, for protocol 1.2, optionally (if the URI scheme is “ms3hs”) setup a server-authenticated TLS session using any implementation-chosen TLS profile.
2. Execute the HTTP protocol binding as defined in §3.2,
3. Receive and process the SAS, described in §3.5.2.1.

3.1 TLS Profile for MS3

The figure below describes the full handshake protocol of TLS used in version 1.0 of the protocol. All the messages MUST be present and in conformance with [TLS] and this section. In the following description the MS3 Client is acting as a TLS client.

```
Client                                        Server

ClientHello                 -------->
                                              ServerHello
                                              Certificate
                                              [ServerKeyExchange]
                                              CertificateRequest
                            <--------         ServerHelloDone
Certificate
ClientKeyExchange
```

```
CertificateVerify
[ChangeCipherSpec]
Finished                     -------->
                                          [ChangeCipherSpec]
                             <--------    Finished
Application Data             <------->    Application Data
```

Figure 4: TLS Handshake

3.1.1 ClientHello and ServerHello

The TLS client and TLS server MAY send TLS 1.0 or later as the TLS version in ClientHello and ServerHello.
TLS 1.0 SHALL be retired at the time set by the PCI Council (https://www.pcisecuritystandards.org/), at which point services SHALL support TLS1.2.

3.1.2 Cipher Suite

Conforming implementations of the specification SHALL support the TLS_RSA_WITH_AES_128_CBC_SHA cipher suite as defined in [TLSAES].
Conforming implementations SHOULD also support the TLS_ECDHE_RSA_WITH_AES_128_GCM_SHA256 cipher suite defined in the references of [TLSAES-2] with the NIST P-256 elliptic curve [FIPS186].

3.1.3 Server Certificate

The X.509v3 certificate of the TLS server SHOULD have the keyEncipherment key usage set (Note: According to [TLS], if the key usage extension is present the keyEncipherment bit MUST be set). The TLS client SHOULD validate the TLS server certificate in accordance with [PKIX].

3.1.4 Client Certificate

The X.509v3 certificate of the TLS client SHOULD have digitalSignature key usage.

The TLS client MAY use a NEMO Signing Certificate as defined in §9.4.1 of [MCS]. When the TLS client uses a NEMO certificate, the Certificate Revocation Lists SHALL conform to the profile described in §9.2 of [MCS].

When the MS3 Client uses a NEMO certificate, the client certificate MUST be validated by the service according to the process described in §9.1.4 of [MCS] otherwise the certificate MUST be validated in accordance with [PKIX], except that the service SHALL NOT resolve the CRL from the CRL Distribution Point indicated in the client certificate but instead use its own copy of the then current CRL. This latter CRL is available from the Marlin Trust Management Organization at https://www.marlin-trust.com/.

3.2 HTTP Binding for MS3

This protocol binding is triggered via web interactions between a browser and a web-based media service. This following text defines the processing rules of this binding.

Implementation guidance is given in §4 that describes common mechanisms that dynamically interrogate the capabilities of an MS3 Client and pass the requisite parameters, S-URL and C-URIT, through to the underlying implementation of this specification.

Implementations SHOULD support one of the parameter encodings defined in §3.4 so as to enable a predictable MS3 triggering mechanism (i.e., as described in §4).

In order to resolve the S-URL, the MS3 Client MAY first have to establish a TLS session with the MS3 Service depending on the URI scheme.

Upon connection (possibly with TLS session establishment), the client MUST issue an HTTP request [HTTP] to the resource specified by the S-URL. In this request, the client SHOULD include an entity header to indicate the MS3 protocol version supported by the client. In the event the client does not supply this entity header, the service SHALL assume protocol version 1.0.

If clients indicate the protocol version, clients using the “ms3” URI scheme SHALL signal protocol version 1.0, whereas client using the “ms3h”, “ms3hs”, or “ms3hsa” URI scheme SHALL signal protocol version 1.2

The syntax of this header follows (see [HTTP] for a description of this grammar):

```
MS3-Version   =   “X-MS3-Version” “:” 1*DIGIT “.”1*DIGIT
```

The first digit represents the major specification number and the second digit represents the minor specification number. Note that the major and minor numbers MUST be treated as separate integers and that each MAY be incremented higher than a single digit. For example, the following header represents major version 1 and minor version 10;

```
 X-MS3-Version: 1.10
```

For MS3 1.0 Protocol:

- Upon TLS session establishment, the client MUST issue an HTTP GET request to the resource specified by the S-URL

For MS3 1.2 Protocol:

- The Client MUST send the request to the S-URL as a POST. The Content-Type header MUST be set to application/json. The body of the request MUST contain a JSON payload consisting of a JSON object with the following fields:
- “version”: an integer specifying the client protocol version. This field MUST be equal to 1.
- “nonce”: a base64-encoded payload containing a client-generated nonce. It is recommended that this value be a random number of 64 bits or more. This nonce value MUST NOT exceed 32 bytes.
- “clientInfo”: a JSON object representing the client information. This object MUST include an “octopusNode” field, and MAY contain other fields. The “octopusNode” field MUST be a base64-encoded Octopus public personality node representation, including its signature, as specified in [MBB].

If the Client supports ‘skey’ extensions with type=1, as defined in §3.5.2, it SHOULD signal it by including the following entity header in their requests:
X-MS3-Options: kdf-1

Copyright (c) Marlin Developer Community, 2003-2017. All Rights Reserved
Refer to Notices on page 2 for important legal information

If this entity header is not included in the request, the Service MUST assume that the client does not support ‘skey’ extensions with type=1 and MUST NOT include such an extension in its response.

A successful response from the MS3 Service MUST be signaled with an HTTP 200 (OK) response. The body of the HTTP response MUST be an SAS as defined in §3.5.2.1. The HTTP Content-Type entity header MUST signal the MIME type with the following string:

Entity Body	MIME type
MS3 Stream Access Statement	application/vnd.marlin.drm.StreamAccessStatement

An unsuccessful response from the MS3 Service SHALL be signaled with a HTTP response code. In the event of an unauthorized request the service SHALL respond with 401 (Unauthorized). The service SHOULD include an HTML document with more information as to the cause of the failure.

Once the SAS has been retrieved a client will have sufficient information to acquire and consume the media. The next step is for the client to expand the C-URIT (if necessary) and resolve the C-URL to the content. A C-URIT MUST conform to the syntax and processing rules defined in [MURIT]. An MS3 Client MUST support expanding the C-URIT with template variables defined in §3.3.

The client accesses the content resource (e.g. CDN) by resolving the C-URL. The content distribution service determines whether the client is authorized to access the requested resource. The policy by which the service makes this decision is outside the scope of this specification however it is likely that the service will factor in the Authenticator information encoded in the C-URL.

3.3 Marlin Template Variables for MS3

The variable namespace for MS3 variables is “s”.
The general syntax for an MS3 variable is:

```
ms3-var = "authenticator"
```

The value of the variable is the Authenticator field of an SAS. The C-URIT parameter MAY include the above template variable.

The following is an example of C-URIT that includes a template to be expanded with an Authenticator (‘006789F5’) as provided in an SAS.

Input (C-URIT)	http://www.bok.net/music/get-token?auth={s:authenticator}&cid=8967F56D
Output (C-URL)	http://www.bok.net/music/get-token?auth= 006789F5&cid=8967F56D

Copyright (c) Marlin Developer Community, 2003-2017. All Rights Reserved
Refer to Notices on page 2 for important legal information

3.4 MS3 Parameter Encodings

The application protocol defined by this specification requires two distinct parameters to be passed into the underlying implementation, S-URL and C-URIT. So as not to dictate the client architecture a variety of parameter encodings are defined in this section. A conformant implementation MAY support any of these parameter encodings.

3.4.1 MS3 Action Token

MS3 MAY be triggered using an Action Token. The Client MAY support handling of the Action Token.
The Action Token defined by this specification is an extension of the schema defined in §6.1. The schema defines the SASAcquisitionType <bsa:Action>. Instances of this <bsa:Action> element MUST specify the xsi:type attribute with a value of ms3:SASAcquisitionType.

The <bsa:Action> element MUST contain a <ms3:SASLocation> element.
The contents of the <ms3:SASLocation> element MUST be a URL. The corresponding scheme is defined in §6.

The MIME type defined below MAY be used to signal the delivery of an Action Token bearing a SASAcquisitionType <bsa:Action> element.

Entity Body	**MIME type**
MS3 Action Token	application/vnd.marlin.drm.actiontoken2+xml

3.4.2 MS3 Compound URI

MS3 MAY be triggered using a Compound URI. The client MAY support handling of the Compound URI.
The Compound URI is a safe combined encoding of both the S-URL and C-URIT parameters. The Compound URI SHOULD use the “ms3” URI scheme as defined in §3.4.2.1 or the “ms3h(s)” URI schemes defined in §3.4.2.1.

The Compound URI SHALL be formatted as following:

Compound-URI = S-URL “#” C-URIT

The Compound URI MUST be a valid URI. Therefore, the encoding of the fragment SHALL adhere to the percent-encoding rules defined in [URI]. The following example demonstrates the encoding of a Compound URI that includes template variables.

ms3://sas.example.com/getsas/CAFEBEE#http://www.bok.net/stream/get-token?auth=%7bs:authenticator%7d&cid=8967F56D

Implementations that support this encoding SHALL be capable of parsing the Compound URI at the fragment (“#”) delimiter to derive distinct S-URL and C-URIT parameters. Subsequent processing of the resultant C-URIT SHALL decode percent-encoded characters and adhere to the expansion rules defined in §3.3.

Copyright (c) Marlin Developer Community, 2003-2017. All Rights Reserved
Refer to Notices on page 2 for important legal information

3.4.2.1 “ms3”, “ms3h, “ms3hs” and “ms3hsa” URI schemes

Syntax definitions are given using the Augmented BNF (ABNF) for syntax specifications [RFC5234].

The URI scheme's keywords in the following syntax description are case-insensitive. The syntax of the URI whose URI scheme is any of an "ms3", "ms3h", "ms3hs" or “ms3hsa” URI follows the URI base syntax defined in [URI] and is formally described below:

```
ms3-uri   = scheme ":" hier-part [ "?" query ] [ "#" fragment ]
scheme    = one of "ms3",”ms3h”, “ms3hs”, or “ms3hsa”
hier-part = as defined in [URI]
query     = as defined in [URI]
fragment  = C-URIT
C-URIT    = as defined in §3.4.2
```

The “ms3”, “ms3hs” and “ms3hsa” protocol identifications in this URI scheme result in equivalent behavior as the “https” protocol identification in the “https” URI scheme. The “ms3h” protocol identification in this URI scheme results in equivalent behavior as the “http” protocol identification in the “http” URI scheme.

3.4.3 MS3 Manifest File

MS3 MAY be triggered using an MS3 Manifest file. The Client MAY support handling of the MS3 Manifest file.
The manifest is a text document that MUST include S-URL and C-URIT fields. The grammar of these fields is defined below.

Delivery of a manifest file SHALL be signaled using the following MIME type:

Entity Body	MIME type
MS3 Manifest File	application/vnd.marlin.drm.StreamAccessDescriptor

The contents of an MS3 Manifest file SHALL adhere to the following grammar (using the grammar defined in [HTTP]):

```
one or more line separated by \r\n
line = field-name ":" field-value
field-name = LWS 1*(ALPHA | DIGIT | "_" | "-")
field-value = any ascii char except control chars
```

The line for S-URL SHALL be set in the MS3 Manifest file as following:
field-name = “S-URL”
field-value = the value of the S-URL parameter

The line for C-URIT SHALL be set in the MS3 Manifest file as following:
field-name = “C-URI-Template”
field-value = the value of the C-URIT parameter

The content type of the media stream delivered once the C-URIT is expanded and resolved SHALL be signaled in this manifest as follows:
field-name = "Content-Type"

field-value = the MIME Type and codec information following the syntax defined in [RFC4281]

3.5 Stream Access Statement (SAS)

An MS3 Service releases key material and consumption constraints to an authorized MS3 Client.

3.5.1 Handling of SAS

A conformant MS3 Client SHALL only cache an SAS for a reasonable retention period so as to enable content rendering. After playback has ended or stopped (e.g. by user interaction), a conformant MS3 Client SHALL discard the corresponding SAS. Notwithstanding the foregoing, an MS3 Client MAY continue using a retained SAS when playback is temporarily suspended (e.g., by a user pausing playback).

3.5.2 Client/Server Processing for “ms3h”, “ms3hs” and “ms3hsa” URI schemes

- The Server SHALL parse the Client request, check that the JSON payload version is 1, and that all the required fields of the JSON payload are present and syntactically correct.
- The Server SHALL determine the resources needed to generate the requested SAS payload. If an SAS cannot be determined, the Server shall return an HTTP error.
- The Server SHALL validate the signature of the Octopus personality node object as specified in [MBB]. If the signature validation is not successful, the Server SHALL return an HTTP error.
- The Server MAY inspect attributes of the Client’s Octopus personality node object in order to decide if its own policy for responding to Client requests allows an SAS response to be sent to the Client. Based on this, the Server MAY return an HTTP error response.
- The Server MUST generate a cryptographically-random 128-bit session key session_key.
- The SAS payload MUST include an ‘skey’ extension and a ‘sign’ extension as defined in sections 2.1 and 2.2. The ‘sign’ extension MUST be the last extension in the SAS.
- Each key in the SAS response MUST be encrypted (in place) with a key encryption key (KEK) using the AES-128 cipher in ECB mode. If the ‘type’ field of the ‘skey’ extension carrying the session key is 0, the KEK is the session key itself. If the ‘type’ field is 1, the KEK is derived from the session_key, as defined in section 3.5.2.1.
- When the Client receives the response carrying the SAS, it MUST check that the SAS has a valid ‘skey’ extension and ‘sign’ extension, and that the ‘type’ fields of those extensions are both supported (only the value 0 is currently defined). The client MUST then decrypt the encrypted_session_key from the ‘skey’ extension, then verify the signature carried in the ‘sign’ extension. If the signature verification fails, the entire response MUST be discarded.

Extensions defined for “ms3h”, “ms3hs” and “ms3hsa” URI schemes:

- ‘skey’ extension
  Extension type: 0x736b6579 (‘skey’)

Copyright (c) Marlin Developer Community, 2003-2017. All Rights Reserved
Refer to Notices on page 2 for important legal information

Extension payload:

Field Name	Field Size (bytes)	Field Payload
type	1	0 or 1
encrypted_session_key	variable	128-bit AES session_key encrypted with the Octopus Scuba Sharing RSA public key, using RSA OAEP.

- ‘sign’ extension

Extension type: 0x7369676e (‘sign’)

Extension payload:

Field Name	Field Size (bytes)	Field Payload
type	1	0
hmac	20	HMAC-SHA1 signature of the concatenation of the entire SAS payload up to, but not including, the ‘sign’ extension followed by the Client-supplied nonce

3.5.2.1 Derivation of the KEK

The keys carried in the SAS when using the “ms3h”, “ms3hs” or “ms3hsa” URI schemes are encrypted with a key encryption key (KEY) which is derived from the session_key carried in an ‘skey’ extension.

The KEK value is derived as follows:

KEK = TRUNCATE(SHA1(session_key))

Where,

- session_key is a 128-bit key
- SHA1 is the one-way hash function defined in [SHA1]
- TRUNCATE takes the 128 most significant bits of the 160-bit output of SHA1

3.5.3 Definition of SAS

The structure and semantics of this information is expressed in the form of a Stream Access Statement (SAS) as defined below.

```
SAS: {
   keyCount:     unsigned int (32)
   keys:         Key [keyCount]
   authenticatorSize:  unsigned int (32)
   authenticator:      bit (8) [authenticatorSize]
   controlFlags: bit (8)
   usageRule: {
      outputControl: {
         outputControlValue: bits (32)
         outputControlFlags: bits (32)
      }
   }
   extensionCount: unsigned int (32)
   extensions:     Extension [extensionCount]
}
```

Copyright (c) Marlin Developer Community, 2003-2017. All Rights Reserved
Refer to Notices on page 2 for important legal information

```
Key: {
   contentId: bit (160)
   keyData:  bit (128)
}

Extension: {
   size:         unsigned int (32)
   type:         bit(32)
   criticalFlag: bit (8)
   payload:      bit (8) [size-9]
}
```

- `keyCount`: number of keys in the `keys` array. In case of unencrypted content, the `keyCount` SHALL be set to 0.
- `keys`: array of zero or more `Key`. Each `Key` contains a `contentId` and the corresponding content key as `keyData`.
    - `contentId`: 160-bit SHA-1 hash of content identifier included in content. MS3 Client SHALL compute SHA-1 hash of content identifier in content when comparing `contentId` in SAS.
    - `keyData`: content key corresponding to the content identified with `contentId`
- `authenticatorSize`: the number in bytes of the `authenticator`. When there is no `authenticator`, `authenticatorSize` SHALL be set to 0.
- `authenticator`: opaque service specific data encoded as UTF-8. When the `authenticator` value is set, the `authenticator` is used to expand the C-URIT into a C-URL as defined in §3.3. Content that is retrieved from a URL composed using the authenticator SHALL be governed according to the SAS, regardless of whether the content is encrypted or not.
- `controlFlags`: bit vector of flags. If bit 0(LSB) is set to 1, the client SHALL NOT retain streamed content (either in encrypted or plaintext form) corresponding to this SAS except for a reasonable retention period to allow for buffering so as to preserve the fidelity of the content rendering. The remaining bits, bit1-bit7 are reserved. All reserved bits SHALL be set to 0.
- `usageRule`: information used to enforce the governance requirements of the content and its consumption.
- `outputControl`: data structure including `outputControlValue` and `outputControlFlags`. The output control requirements carried in an SAS SHALL be enforced or the corresponding content SHALL NOT be consumed.
- `outputControlValue`: bit fields indicating the value of zero or more output control fields. The meaning of the fields and their possible values are defined in §4 of [MOC]. The fields are encoded as follows:

Copyright (c) Marlin Developer Community, 2003-2017. All Rights Reserved
Refer to Notices on page 2 for important legal information

Bit range (0 is the least significant bit)	Output Control Technology	Field name
0	BasicCCI	DigitalOnlyToken
1..4	BasicCCI	Reserved
5	BasicCCI	EPN
6..7	BasicCCI	CCI
8	BasicCCI	ImageConstraintToken
9..10	BasicCCI	APS
11	DTCP	RetentionMoveMode
12..14	DTCP	RetentionState
15	DTCP	EPN
16..17	DTCP	DTCP_CCI
18	DTCP	ImageConstraintToken
19..20	DTCP	APS

- `outputControlFlags`: bit vector of flags indicating which fields are signalled in the `outputControlValue`. When a flag in this vector is set to 1, the Client SHALL set the output control parameters as specified by the corresponding bit-field in the `outputControlValue`. When a bit flag in this field is set to 0, the Client SHALL set the corresponding output control parameters as specified by the default in §3.5.4.

Flag Bit (0 is the least significant)	Output Control Technology	Field Name
0	BasicCCI	DigitalOnlyToken
1	BasicCCI	EPN
2	BasicCCI	CCI
3	BasicCCI	ImageConstraintToken
4	BasicCCI	APS
5	DTCP	RetentionMoveMode
6	DTCP	RetentionState
7	DTCP	EPN
8	DTCP	DTCP_CCI
9	DTCP	ImageConstraintToken
10	DTCP	APS

Copyright (c) Marlin Developer Community, 2003-2017. All Rights Reserved
Refer to Notices on page 2 for important legal information

- `extensionCount`: number of `Extension`s in the `extensions` array. In case of there is no Extension, the `extensionCount` is set to 0.
- `extensions`: array of zero or more `Extension`. Each `Extension` contains `type`, `size`, `criticalFlag`, and `payload`.
- `type`: by convention, the 32-bit identifier for an `Extension` is written as a 4-letter word, where each letter's 8-bit ASCII code is the corresponding 8-bit byte portion of the identifier. For example, the identifier value 0x61626364 (hexadecimal) would be written 'abcd', because the ASCII code for 'a' is 0x61, etc.
- `size`: entire byte size of the `Extension`.
- `criticalFlag`: bit vector of flags. An `Extension` that is marked critical (by the bit 0(LSB) of `criticalFlag` is set to 1) SHALL be enforced. If an `Extension` marked as critical is encountered that is not supported or understood, then the content SHALL NOT be rendered.
- `payload`: description of `Extension`.

3.5.4 Default Output Control

The default set of BasicCCI is specified in the following table.

Name	Type	Default Value	Description
EPN	Integer	1	EPN-unasserted
CCI	Integer	11	Never Copy
ImageConstraintToken	Integer	1	High Definition Analog Output in High Definition Analog Form
DigitalOnlyToken	Integer	0	Output of decrypted content is allowed for Analog/Digital Outputs
APS	Integer	01	APS on: type 1 (AGC)

Table 1: Default output control for Basic CCI

The following table defines the default set of DTCP.

Name	Type	Default Value	Description
Retention-Move-mode	Integer	1	Non-Retention-mode
Retention_State	Integer	111	90 minutes
EPN	Integer	1	EPN-unasserted
DTCP_CCI	Integer	11	Copy-Never
Image_Constraint_Token	Integer	1	High Definition Analog Output in High Definition Analog Form
APS	Integer	01	APS on: Type 1 (AGC)

Table 2: Default output control for DTCP

Copyright (c) Marlin Developer Community, 2003-2017. All Rights Reserved
Refer to Notices on page 2 for important legal information

4 Triggering MS3 Clients

This section defines common mechanisms by which a media service triggers the application protocol defined in this specification. A conforming client implementation SHOULD implement one of these trigger mechanisms. Regardless of the mechanism by which the S-URL and C-URIT are conveyed to the MS3 Client, use of the content obtained from the C-URL SHALL be subject to the constraints expressed in the SAS obtained from the S-URL.

4.1 Triggering MS3 Clients via Action Token

As with other Marlin protocols, MS3 MAY be triggered using an Action Token. Clients MAY support handling of this type of Action Token.

The Action Token defined in §3.4.1 can be used to initiate the MS3 application protocol.

The contents of the <ms3:SASLocation> element SHALL be the S-URL parameter. In addition, the content of this element MAY use the Compound URI encoding defined in §3.4.2.

The following is an example of this <bsa:Action> element:

```
<bsa:Action xsi:type="ms3:SASAcquisitionType">
    <ms3:SASLocation>https://www.xyzmovie.com/xyz.SAS?bt=
YCn70D0Av/xt5sXcSj7XWFAAAAEAAAA</ms3:SASLocation>
</bsa:Action>
```

An MS3 Client SHOULD initiate the protocol binding defined in §3.2 to resolve the URL carried in the <ms3:SASLocation> element. In the case of a Compound URI encoding the MS3 Client SHALL parse the URI to derive the S-URL and C-URIT components.

Copyright (c) Marlin Developer Community, 2003-2017. All Rights Reserved
Refer to Notices on page 2 for important legal information

4.1.1 Use of an AT in Open IPTV Forum context (using OIPF DRM Agent plugin) (Informative)

Below is an example demonstrating how MS3 can be incorporated into an OIPF context using existing OIPF mechanisms. The support of the MS3 feature is signaled by an OIPF DRMSystemID with value “urn:marlin:ms3:1-0”.

```
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Transitional//EN" "http://www.w3.org/TR/xhtml1/DTD/xhtml1-
transitional.dtd">
<html xmlns="http://www.w3.org/1999/xhtml">
<head id="head1"><title>OIPF MS3 Example</title><link type="text/css" rel="stylesheet" href="Stylesheets/style.css" />
<script type="text/javascript">

        function startPlayback()
                {
                vid = document.getElementById("videoObject");

                //Setup video object with hardcoded C-URL.
                vid.data = "videos/movie.odf";
                vid.setFullScreen(1);
                vid.play(1);
                }

        function HandleOnDRMMessageResult(msgID, resultMsg, resultCode)
                {
                if (resultCode == 0) {startPlayback();}
                else {} //SAS download failed.
                }

        function getSASandPlay()
                {
                //Assuming OIPF will choose existing Marlin DRMSystemID and use MS3 Action Token as msgType

                //Create action token for hardcoded S-URL.
                ms3AT = "<bsa:Action xsi:type=ms3:SASAcquisitionType>"
                 + “<ms3:SASLocation>https://server.com/movie.sas</ms3:SASLocation>"
                  +"</bsa:Action>";

                drm = document.getElementById("drmagent");
                drm.onDRMMessageResult = HandleOnDRMMessageResult;
                drm.sendDRMMessage('application/vnd.marlin.drm.actiontoken2+xml',ms3AT,'urn:dvb:casystemid:19188');

                }

        function init()
        {
                if (detectMS3Support())
                {
                        getSASandPlay();
                }
        }
</script>

</head>
<body onload="init();">
      <div id="videowrapper">
          <object id="videoObject" type="video/mpeg4"> </object>
          <object id="drmagent" type="application/drmagent" style="visibility:hidden;"></object>
        <object id="capabilities" type="application/oipfCapabilities" style="visibility:hidden;"></object>
      </div>
</body>
</html>
```

4.2 Triggering MS3 Clients via Compound URI

An MS3 Client that supports the Compound URI trigger SHALL support the parameter encoding defined in §3.4.2. The Compound URI SHALL be used to uniquely associate an SAS with corresponding content when contentID is not specified in an SAS and content for a plaintext form.

If C-URIT includes the placeholder for Authenticator, the MS3 Client SHALL use the associated S-URL to retrieve the SAS bearing the Authenticator. The supplied Authenticator SHALL replace the placeholder in the C-URIT.

A Client supporting the Compound URI trigger mechanism SHALL support the capability query for the SAS MIME Type.

The capability detection SHOULD include a query for the supported codecs. The codec parameter SHALL adhere to the syntax and encoding defined in [RFC4281].

The capability detection SHOULD include a query for the media container format to unambiguously indicate the media format of the content. The media format container parameter SHALL adhere to the generic syntax and encoding defined in [RFC4281]. This media format container parameter has the following syntax:

format := "container" "=" mime-type
mime-type := The MIME type of the media to be delivered when the content URL is resolved.

A Client supporting the Compound URI trigger mechanism SHALL support and process the container parameter query. If the Client does not support the media format designated in the container parameter it SHALL return a negative response when queried.

The following sample JavaScript demonstrates capability detection using the HTML5 DOM API. This script detects Client support for the Compound URI, container type and codec.

```
if (canPlayType('application/vnd.marlin.drm.StreamAccessStatement;
                container="application/vnd.marlin.drm.pdcf";
                codecs="avc1.42E01E, mp4a.40.2"') == "probably")
 // The underlying implementation. supports the CompoundURI
```

The Figure 5 is an example usage of Compound URI where step 2 provides S-URL for SAS acquisition, and step 5 provides the Compound URI which is used to associate the SAS with C-URIT to resolve URI Template in step 8.

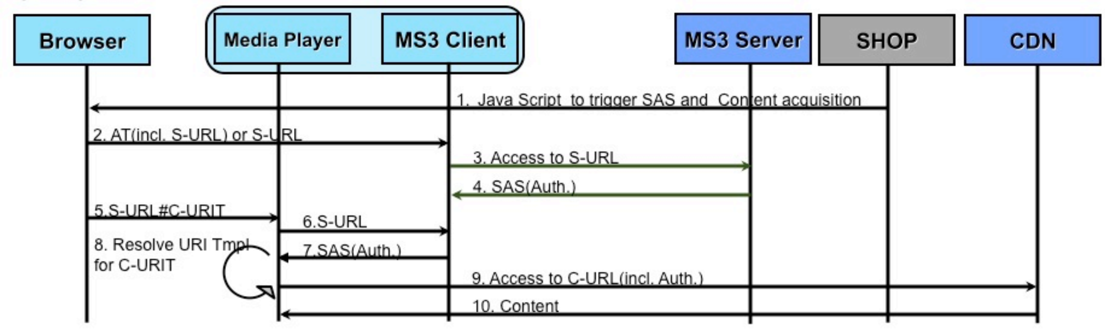

Browser
Media Player
MS3 Client
MS3 Server
SHOP
CDN
1. Java Script to trigger SAS and Content acquisition
2. AT(incl. S-URL) or S-URL
3. Access to S-URL
4. SAS(Auth.)
5.S-URL#C-URIT
6.S-URL
8. Resolve URI Tmpl for C-URIT
7.SAS(Auth.)
9. Access to C-URL(incl. Auth.)
10. Content

Figure 5: Example usage of Compound URI

The Figure 6 is an example usage of Compound URI where step 2 provides the Compound URI which is used for acquisition of SAS and also used to associate the SAS with C-URIT to resolve URI Template in step 7.

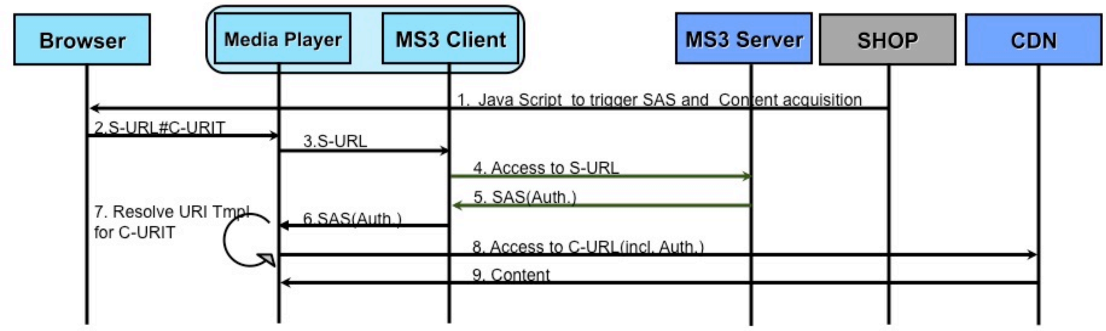


Figure 6: Example usage of Compound URI

4.3 Triggering MS3 Clients via the MS3 Manifest File

MS3 MAY be triggered with the prescribed MIME type and delivering an MS3 Manifest file as defined in §3.4.3.

An MS3 Client supporting this trigger mechanism SHALL uniquely associate the SAS acquired from S-URL with corresponding content acquired from C-URIT when contentID is not specified in SAS and content for a plaintext form. Specifically when C-URIT includes the placeholder for Authenticator, the MS3 Client SHALL use the associated S-URL to retrieve SAS to acquire Authenticator to process the placeholder in the C-URIT.

The example of a MS3 Manifest file follows:

```
S-URL: https://foo.bar/123456789/
C-URI-Template: http://hoge.bar/get-token?authenticator ={s:authenticator}
Content-Type: application/vnd.marlin.drm.pdcf; codecs="avc1.42E01E, mp4a.40.2"
```

A Client supporting the MS3 Manifest file in the context of HTML5 SHALL return "probably" or “maybe” to the capability query MS3 Manifest file MIME Type.

The following sample JavaScript demonstrates capability detection using the HTML5 DOM API. This script detects Client support for the MS3 Manifest file.

```
if (canPlayType("application/vnd.marlin.drm.StreamAccessDescriptor")
                              == "probably")
  // The underlying implementation can be feed the url to the Manifest file
```

Figure 7 is an example usage of MS3 Manifest file where step 4 provides the MS3 Manifest file which is used for acquisition of SAS and also used to associate the SAS with C-URIT to resolve URI Template in step 9.

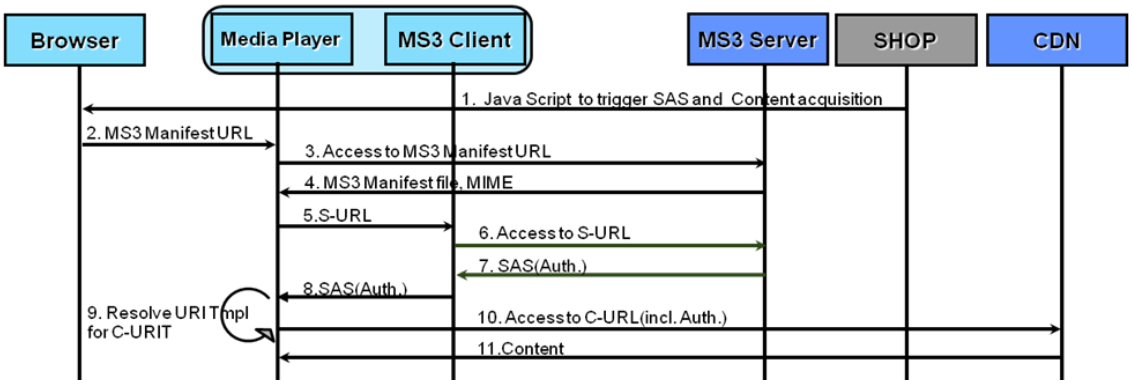

Browser
Media Player
MS3 Client
MS3 Server
SHOP
CDN
1. Java Script to trigger SAS and Content acquisition
2. MS3 Manifest URL
3. Access to MS3 Manifest URL
4. MS3 Manifest file, MIME
5.S-URL
6. Access to S-URL
7. SAS(Auth.)
8.SAS(Auth.)
9. Resolve URI Tmpl for C-URIT
10. Access to C-URL(incl. Auth.)
11.Content

Figure 7: Example usage of MS3 Manifest file

4.4 Sample Java script to trigger MS3 Client (Informative)

The following is sample java script which detects Client capability and chooses appropriate mechanism to trigger MS3 Client.

```
<script type="text/javascript">
// MS3 Example
var detectVideoSupport = function (){
    var detect = document.createElement('video') || false;
    this.html5 = detect && typeof detect.canPlayType !== "undefined";
    // test for the various protected packaged content supported by the underlying
    // video implementation
    this.dcf = this.html5 && (detect.canPlayType("application/vnd.oma.drm.dcf") === "maybe"
                        || detect.canPlayType("application/vnd.oma.drm.dcf") === "probably");
    return this;
};
var dectectOITFSupport = function (){
    this.oitf = window.oipfObjectFactory !== "undefined" || false;
    // test for content access streaming descriptor support
    this.cas    = this.oitf && window.oipfObjectFactory.isObjectSupported(
                                        "application/vnd.oipf.ContentAccessStreaming+xml");
    // test for the various protected packaged content supported by the underlying
    // video implementation
    this.dcf = this.oitf &&
            (window.oipfObjectFactory.isObjectSupported("application/oipfDrmAgent") &&
             window.oipfObjectFactory.isObjectSupported("application/vnd.oma.drm.dcf"));
    return this;
};
function initiateMS3Playback (actionToken) {
    var html5Video = detectVideoSupport();
    var oitfVideo = detectOITFSupport();
    var videoPlayer;
    var pluginElement;
    if (html5Video) {
      // Support for HTML5 <video> detected. Create the video element and source
      // child pointing it to the serviceLocation
      videoPlayer = document.createElement('video');
      // add a <source> child and off we go
    } else if (oitfVideo) {
     // Support for OITF detected. Determine if the Content Access Streaming is supported
      if (oitfVideo.cas) {
     // pass the Content Access Streaming statement URL into the MS3 plugin/player
     videoPlayer = window.oipfObjectFactory.createVideoMpegObject();
     document.getElementById('playerDiv').appendChild(videoPlayer);
     videoPlayer.data = actionToken.serviceLocation;
     // start playback
      } else {
     // Two steps using sendDRMMessage and videoPlayer.data(S-URL+C-URLTemplate)
     pluginElement = document.getElementByID("drmplugin");
     pluginElement.sendDRMMessage("application/vnd.marlin.drm.actiontoken2+xml",
                                          actionToken);
     // once that returns we pass the service url into the player
     videoPlayer = window.oipfObjectFactory.createVideoMpegObject();
     document.getElementById('playerDiv').appendChild(videoPlayer);
     videoPlayer.data = actionToken.serviceLocation;
     // start playback
      };
    };
};
</script>
```

Copyright (c) Marlin Developer Community, 2003-2017. All Rights Reserved
Refer to Notices on page 2 for important legal information

5 Annex 1: Alternative client-side MS3 architecture (Informative)

Figure 1 depicts the generic client-side architecture for MS3, in which the MS3 Client is a stand-alone component and the number of components inside the tamper resistant boundary is minimal. Although out-of-scope for MS3, in many practical cases the interface between the Media Service and the Browser also uses TLS to provide security. Consequently many browsers support TLS. In addition, on some embedded clients the firmware as a whole, including the browser, is made tamper resistant. In such a context the alternative client-side architecture depicted in Figure 8 may be considered.

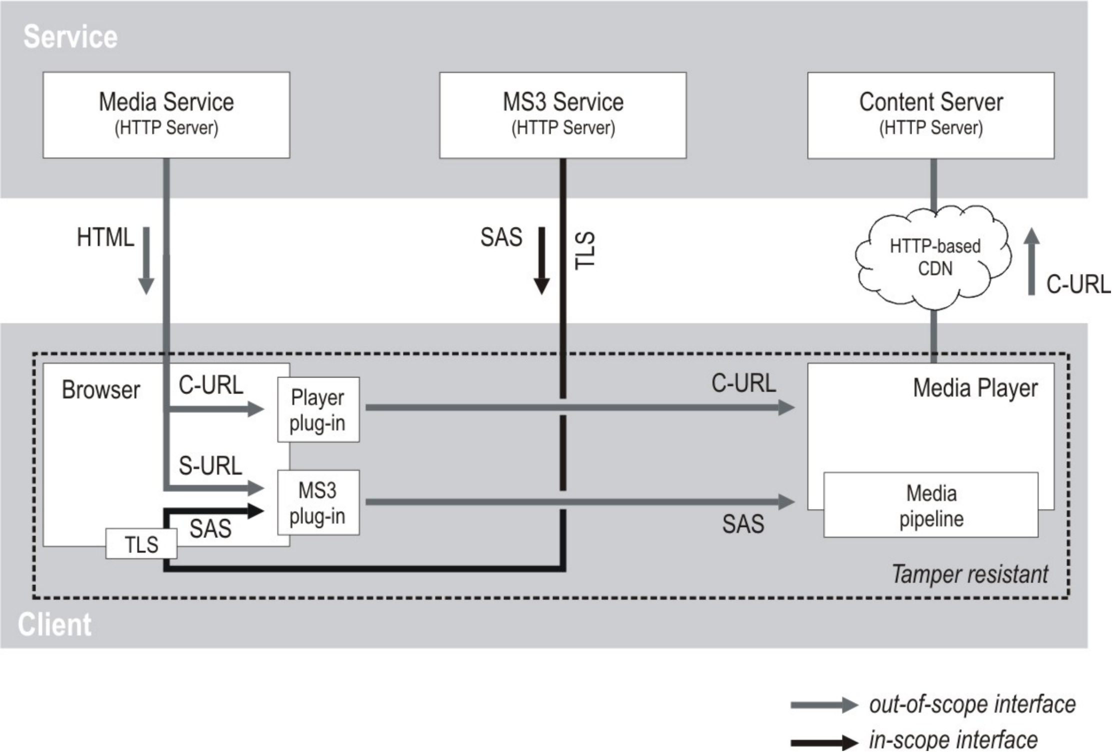


Figure 8: Alternative Client-side MS3 architecture (protocol version 1.0)

Also in this architecture the web page of the Media Service would pass the S-URL to the MS3-plugin using for example one of the mechanisms in §4. But rather than passing the S-URL on to a dedicated MS3-Client component, the implementation of the MS3-plug-in, using the Browsers plug-in API (e.g. NPN_GetURL), would request the Browser to resolve the S-URL. The Browser would initiate the TLS with the MS3Service, request and receive the SAS and pass it on to the MS3 plug-in, which would pass it on to the Media Player.

Copyright (c) Marlin Developer Community, 2003-2017. All Rights Reserved
Refer to Notices on page 2 for important legal information

6 Annex 2: XML Schemas

6.1 Marlin Broadband Action Token Schema

```
<?xml version="1.0" encoding="UTF-8"?>
<!--

 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.

 Copyright (c) 2003 - 2010 by MDC, 415-112 North Mary Avenue #383
 Sunnyvale, CA 94085, USA.   All rights reserved.   Third-party brands
 and names are the property of their respective owners.

 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/

-->
<xsd:schema xmlns="urn:marlin:broadband:1-2:nemo:services:action-token"
   xmlns:xsd="http://www.w3.org/2001/XMLSchema"
   targetNamespace="urn:marlin:broadband:1-2:nemo:services:action-token"
   elementFormDefault="qualified" attributeFormDefault="unqualified">

   <xsd:element name="BusinessToken" type="BusinessTokenType"/>
   <xsd:simpleType name="BusinessTokenType">
      <xsd:annotation>
         <xsd:documentation>Opaque data structure containing service-specific data
         </xsd:documentation>
      </xsd:annotation>
      <xsd:restriction base="xsd:base64Binary"/>
```

```
</xsd:simpleType>

<xsd:element name="ActionToken" type="ActionTokenType"/>
<xsd:element name="ConfigurationInfo" type="ConfigurationInfoType"/>
<xsd:element name="LicenseAcquisition" type="LicenseAcquisitionType"
                  substitutionGroup="Action"/>
<xsd:element name="NodeAcquisition" type="NodeAcquisitionType" substitutionGroup="Action"/>
<xsd:element name="LinkAcquisition" type="RegistrationType" substitutionGroup="Action"/>
<xsd:element name="Deregistration" type="DeregistrationType" substitutionGroup="Action"/>
<xsd:element name="CertificationStandard" type="CertificationStandardType"/>
<xsd:element name="Type" type="xsd:string"/>
<xsd:element name="Uid" type="xsd:anyURI"/>

<!-- ActionTypes -->
<xsd:complexType name="ActionType">
   <xsd:attribute name="id" type="xsd:nonNegativeInteger" use="optional"/>
</xsd:complexType>

<xsd:element name="Action" type="ActionType"/>

<!-- ActionToken -->
<xsd:complexType name="ActionTokenType">
   <xsd:sequence>
      <xsd:element ref="ConfigurationInfo"    minOccurs="0"/>
      <xsd:sequence maxOccurs="unbounded">
         <xsd:element ref="Action"/>
      </xsd:sequence>
   </xsd:sequence>
</xsd:complexType>

<!-- ConfigurationInfo -->
<xsd:complexType name="ConfigurationInfoType">
   <xsd:sequence>
      <xsd:element name="ResourceLocation" type="xsd:string" maxOccurs="unbounded"/>
   </xsd:sequence>
   <xsd:attribute name="broadbandServiceId" type="xsd:anyURI" use="required"/>
   <xsd:attribute name="configVersion" type="xsd:nonNegativeInteger" use="required"/>
</xsd:complexType>

<!-- LicenseAcquisitionType -->
<xsd:complexType name="LicenseAcquisitionType">
   <xsd:complexContent>
      <xsd:extension base="ActionType">
         <xsd:sequence>
            <xsd:choice>
               <xsd:element ref="Type"/>
               <xsd:element ref="Uid"/>
            </xsd:choice>
            <xsd:element ref="BusinessToken"/>
            <xsd:element ref="CertificationStandard" minOccurs="0" maxOccurs="unbounded"/>
         </xsd:sequence>
      </xsd:extension>
   </xsd:complexContent>
</xsd:complexType>

<!-- NodeAcquisitionType -->
<xsd:complexType name="NodeAcquisitionType">
   <xsd:complexContent>
      <xsd:extension base="ActionType">
         <xsd:sequence>
            <xsd:element ref="BusinessToken"/>
            <xsd:element ref="CertificationStandard" minOccurs="0" maxOccurs="unbounded"/>
         </xsd:sequence>
      </xsd:extension>
   </xsd:complexContent>
```

```
  </xsd:complexType>

  <!-- RegistrationType which is used for LinkAcquisition and LinkAcquisition element -->
  <xsd:complexType name="RegistrationType">
    <xsd:complexContent>
      <xsd:extension base="ActionType">
        <xsd:sequence>
          <xsd:choice>
            <xsd:element ref="Type"/>
            <xsd:element ref="Uid"/>
          </xsd:choice>
          <xsd:element ref="Uid"/>
          <xsd:element ref="BusinessToken"/>
          <xsd:element ref="CertificationStandard" minOccurs="0" maxOccurs="unbounded"/>
        </xsd:sequence>
      </xsd:extension>
    </xsd:complexContent>
  </xsd:complexType>

  <!-- DeRegistrationType -->
  <xsd:complexType name="DeregistrationType">
    <xsd:complexContent>
      <xsd:extension base="ActionType">
        <xsd:sequence>
          <xsd:choice>
            <xsd:element ref="Type"/>
            <xsd:element ref="Uid"/>
          </xsd:choice>
          <xsd:element ref="Uid"/>
          <xsd:element ref="BusinessToken"/>
        </xsd:sequence>
      </xsd:extension>
    </xsd:complexContent>
  </xsd:complexType>

  <!-- certification standard type   -->
  <xsd:complexType name="CertificationStandardType">
    <xsd:attribute name="name" type="xsd:anyURI" use="required"/>
    <xsd:attribute name="use" type="useType" use="required"/>
    <xsd:attribute name="validity" type="xsd:duration" use="optional"/>
  </xsd:complexType>
  <xsd:simpleType name="useType">
    <xsd:restriction base="xsd:string">
      <xsd:enumeration value="must"/>
      <xsd:enumeration value="should"/>
    </xsd:restriction>
  </xsd:simpleType>
</xsd:schema>
```

6.2 MS3 Action Token

```
<?xml version="1.0" encoding="UTF-8"?>
<!--

    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.

    Copyright (c) 2003 - 2010 by MDC, 415-112 North Mary Avenue #383
    Sunnyvale, CA 94085, USA.    All rights reserved.    Third-party brands
    and names are the property of their respective owners.

    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/

-->
<xsd:schema xmlns="urn:marlin:ms3:1-0:services:schemas:streaming:action-token"
    targetNamespace="urn:marlin:ms3:1-0:services:schemas:streaming:action-token"
    xmlns:xsd="http://www.w3.org/2001/XMLSchema"
    xmlns:bsa="urn:marlin:broadband:1-2:nemo:services:action-token" elementFormDefault="qualified"
    attributeFormDefault="unqualified">

    <!-- imports -->
    <xsd:import namespace="urn:marlin:broadband:1-2:nemo:services:action-token"
        schemaLocation="./Broadband-services-action.xsd"/>

    <!-- Supporting Complex Types -->
    <xsd:complexType name="SASAcquisitionType">
        <xsd:complexContent>
            <xsd:extension base="bsa:ActionType">
                <xsd:sequence>
                    <xsd:element name="SASLocation" type="xsd:anyURI"/>
                </xsd:sequence>
            </xsd:extension>
        </xsd:complexContent>
    </xsd:complexType>
</xsd:schema>
```