Media Object Server (MOSä) Protocol v2.8.1

MOS Protocol version 2.8.1
Document Revision 5.22
Revision Date: Thursday July 1, 2004

Copyright Notice

License

This document is provided to You under the conditions outlined below. These conditions are designed to guarantee the integrity of the MOSä Protocol and to ensure the compatibility of solutions implementing the MOSä Protocol. Use or implementation of the MOSä Protocol as described herein, may only occur if You agree to these conditions. Failure to follow these conditions shall void this license. "You" shall mean you as an individual, or, where appropriate, the company or other entity on whose behalf you are using this document, and includes any entity which controls, is controlled by, or is under common control with You.

You must agree to the following in order to use the MOSä Protocol:

1. You must use and implement all messages defined by the MOS protocol MOS 2.8.1 Profiles listed below per the profiles supported by your device.
2. You may not modify message names, order of defined tags within a message, tag structure, defined values, standards and constants.
3. You may not process messages in a means that is dependent on non-standard messages, tags or structures.
4. You may add additional tags to messages, but the modified messages must contain the defined minimum required tags for each message, in the order defined by this document.
5. Implementations of the MOS Protocol following the above guidelines may claim to be "MOSä Compatible" or "MOSä v2.8.1 Compatible"
6. If You add additional tags, it is strongly recommended these be included and encapsulated only within the provided <mosExternalMetadata> structure and not inserted into the pre-defined body of the message.
7. It is inappropriate to make representations of MOS Compatibility unless you have followed these guidelines.

Abstract

MOS is a six year old, evolving protocol for communications between Newsroom Computer Systems (NCS) and Media Object Servers (MOS) such as Video Servers, Audio Servers, Still Stores, and Character Generators. The MOS Protocol development is supported through cooperative collaboration among equipment vendors, software vendors and end users.

Status of this document

This document reflects changes to the MOS protocol discussed during the MOS meetings at NAB 2004 and IBC 2003 and is referred to as MOS v2.8.1. This is the final draft. Changes are summarized here.

How to use this document

The document contains bookmarks and hyperlinks. The Table of Contents and other areas contain underlined phrases. Depending on the viewer application used, clicking or double clicking on underlined links will take the viewer to the relevant portion of the document.

Please make special note of the Profiles section which provides context for the use of command messages which are later defined in detail.

Examples of each MOS message and data structure are included. These messages may be used for testing. Developers are encouraged to cut these messages from the document, change the value of ID tags as appropriate, and paste the modified messages into validation tools. Other than these example messages, validation tools are not provided by the MOS Group.

Media Object Server Protocol v2.8.1

Table of Contents

NOTE: The following messages are included only for backwards compatibility with MOS v2.6 and have been replaced by 3.6.12 roElementAction in MOS version 2.8. These messages will be dropped from future versions of the Protocol.

1. Introduction

This document reflects changes to the MOS protocol discussed during the MOS Working Group meetings at NAB 2004 and IBC 2003.

A new family of messages has been added to allow the NCS to query the MOS for object metadata meeting certain criteria. The mosReqObjList message supports a subset of the XPath query language for mosObj message retrieval. They have been added to Profile 3 – Advanced Object-Based Workflow.

A new messageID element has been added to all the MOS device messages to support intelligent retry if a connection times out before a response to a message has been received. Three ActiveX messages also had the messageID tag added:

ncsAppInfo
mosItemReplace
roStorySend

The ActiveX messages do not require a value for the tag, as the communication happens within a single process.

A new ActiveX message ncsReqAppClose has been added. A hosted ActiveX Plugin can send this message

to the NCS Host when it wants to shut itself down.

The roElementAction message description has been rewritten to make its use clearer. More examples have also been added. The message syntax and functionality have not changed.

All messages are now included in the DTD in Section 8. All the example messages have been validated by the DTD.

A reminder: MOS Protocol v2.8.1 compatible equipment will ignore, without error, any unknown tags in a message, so long as the message or structure contains properly formatted XML content and the minimum subset of required MOS tags for that message or structure.

2. MOS Profiles

The purpose of these profiles is to define basic levels of functionality enabled by the MOS Protocol v2.8.1.

There are seven Profiles defined:

Vendors wishing to claim MOS compatibility must fully support, at a minimum, Profile 0 and at least one other Profile.

When claiming MOS compatibility or using the MOS Logo, vendors must clearly state which profiles of one through six they support. For instance "MOS Compatible – Profiles 1,2,3,4,6"

In order to claim support for a specific profile the vendor must fully implement all messages in the manner specified by the profile. In addition, the vendor must fully implement and support the workflow described by the profile.

If a vendor does not state profile support, or does not support all messages or functionality described by a profile, or does not support at least two profiles, one of them being Profile 0, they cannot claim MOS compatibility.

Optional functionality is clearly identified with the word "Optional" or with the phrase "Recommended Work Practice" in bold, followed with optional information in italics.

All other functionality is required.

The purpose of MOS Profiles is to describe minimum levels of functionality and support. Vendors are encouraged to derive more complex levels of functionality from any Profile so long as support is maintained

for the basic profile and MOS syntax and transport rules are not compromised.

2.1 Profile 0 – Basic Communication

This Profile enables basic MOS XML message exchange and discovery between applications and machines using TCP/IP sockets.

Messages required for support of Profile 0:

heartbeat
reqMachInfo
listMachInfo

General Work Flow for Profile 0

- Establish communication to another MOS device
- Send a <heartbeat> message to another application and receive a <heartbeat> message in response
- Send a <reqMachInfo> message to another application and receive a <listMachInfo> message in response.

Implementation Notes

This Profile encompasses the most basic requirements and functions to support MOS Protocol message transfer. The three basic messages included in this profile, <heartbeat>, <reqMachInfo> and <listMachInfo> can be exchanged between any MOS v2.8.1 complaint devices

Profile 0 required MOS message support

heartbeat

The heartbeat message is designed to allow one application to confirm to another that it is still alive and communications between the two machines is viable.

An application will respond to a heartbeat message with another heartbeat message. However, care should be taken in implementation of this message to avoid an endless looping condition on response.

Recommended Work Practice: *It is useful for a MOS Protocol enabled application to be aware of the three levels of connectivity which are required for MOS message exchange:*

1) *Network Connectivity: You must be able to "Ping" the remote machine hosting the application you wish to communicate with.*
2) *Socket Connectivity: You must be able to establish a socket connection with the remote application*

3) *Application Response: You must be able to receive an ACK message in response to the message you have transmitted.*

If you can send a <heartbeat> message and receive a <heartbeat> message in response you have verified the continuity at all three levels.

Each heartbeat message contains a time stamp. This gives each application the opportunity to synchronize time of day, with a relatively low degree of precision, and to be aware of the other machine's local offset from GMT.

reqMachInfo

This message is a request for the target machine to respond with a listMachInfo message.

listMachInfo

This message allows the machine to identify itself with manufacturer ID, model, hardware and software revisions, MOS version supported, etc.

This message identifies which MOS Profiles it supports.

The machine may also optionally identify information necessary for remote devices to install and configure an associated ActiveX control.

Recommended Work Practice: *Applications may optionally use the information contained in this message to provide automatic or semi-automatic configuration.*

General Explanation of MOS message format and construction

Identification

In practice the MOS and NCS character names are predefined in each system and IP addresses associated with each name.

Encoding

The supported character encoding is ISO 10646 (Unicode) in UCS-2, as defined in The Unicode Standard, version 2.0. All MOS message contents are transmitted in Unicode, high-order byte first, also known as "big endian."

MOS Message Format

The MOS Protocol is fundamentally a tagged text data stream. In the version 2.x, data fields are character delimited using Extensible Markup Language (XML™) tags defined in the MOS Data Type Definition (DTD). In MOS v1.x data fields were delimited using a proprietary format.

Extensible Markup Language (XML)

The syntax of MOS v2.8.1 is an application of XML, an international standard for describing document content structure. XML is a simple, flexible text format based on SGML (ISO 8879). XML is an abbreviated version of SGML, to make it easier for you to define your own document types, and to make it easier for programmers to write programs to handle them. It omits the more complex and less-used parts of SGML in return for the benefits of being easier to write applications, easier to understand, and more suited to delivery and interoperability over the Web.

All tags are case sensitive. All MOS messages must be well formed XML, but are not required to be valid.

Each MOS message begins with the root tag ("mos"), followed by the MOS and NCS ID ("mosID" and "ncsID"), and followed by the message type. Data follows in tagged form.

Vendors are encouraged to add CR/LF and Tabs within a message to improve readability for debugging purposes.

Unknown Tags

Should a MOS or NCS encounter an unknown message or data tag the device will ignore the tag and the data and continue to process the message. Unknown data tags will not generate an application error. The application has the option of indicating a warning.

Data Format for Object <description> field

The value of Object <description> is restricted to plain Unicode UCS-2 text that includes Tab, CR, LF and the optional markup for paragraphs, tabs and emphasis. Formatted text such as HTML, RTF, etc. will not be allowed in the unstructured description area.

Languages

Data tags and constants are formatted as English.
The only Data Fields that can contain other languages are:

- createdBy
- modifiedBy
- roSlug
- storySlug
- storyBody
- itemSlug
- description

And the data structure
- mosExternalMetadata

Numbers

Numbers are formatted as their text equivalent, e.g.:
The decimal number 100 is represented as text "100".
Hex FF55 is represented as text "0xFF55" or "xFF55".

Running Orders

1) Running Order (Unique ID - may appear only once in the NCS)
2) Story (Unique ID - may appear only once in the RO)
3) Item (Unique ID - may appear only once in a story)
4) Object (Unique ID - may appear only once in an item)

It is assumed that all Unique ID's (UID's) created by one machine are respected by others.

Order of data fields within an item is significant.

Items are sent in the order they will be played.

Order of items is significant.

Multiple Running Orders may contain the same Story.

Running Orders may contain zero or more Stories.

Multiple stories can contain the same Object referenced by different Items.

Stories can contain multiple Items.

Item IDs may appear only once in a Story, but can appear in other Stories.

Objects can appear multiple times in a Story, but only one object may appear in an Item.

A Running Order Item is defined by the combination Running Order.Story.Item and contains the UID's of the Running Order, Story and Item which together can identify a unique Item within a Running Order. Additions, deletions, and moves within the running order are referenced in this way to the Item.

Definition of Object Sample Rate

Still Store and Character Generator media objects are defined as having 1 sample per second. They are special cases that require the NCS and MOS applications to understand they do not change every second.

Message Transport

MOS Lower Port (10540) is defined as the default TCP/IP port on which the NCS will accept connections from MOS devices. Multiple simultaneous connections are supported. This socket is referred to as "Media Object Metadata" port in the Message Types section.

MOS Upper Port (10541) is defined as the default TCP/IP port on which the MOS will accept

connections from the NCS. Multiple simultaneous connections are supported. This socket is referred to as "Running Order" port in the Message Types section.

MOS uses two ports bi-directionally. Applications will simultaneously listen for messages on both ports – see **Message Exchange** below.

Ports 10520 and 10521 were specified as Lower and Upper Ports in previous versions of the MOS Protocol. Beginning in version 2.5 these ports are vendor selectable but site specific. All MOS enabled machines within a site or enterprise should communicate using the same ports.

Because some vendors reported problems using port 10521 with Microsoft Windows NT the new port numbers used as examples are now 10540 and 10541.

For example, an NCS initiated create running order command and the MOS' associated ACK would take place on MOS Upper Port (10541). Object updates sent from the MOS and the associated NCS ACK would take place on MOS Lower Port (10540).

Message Exchange

To send a MOS message from MOS to NCS or vice versa:

1. An application will open a socket on the appropriate port to the receiving device if a socket has not already been established.

2. The application will then send the message.

3. The application will then hold the socket open and wait for a mosAck message to be returned on the same socket before either dropping the socket or transmitting the next message.

4. Optionally, either device may send <heartbeat> messages at regular intervals to the other machine and look for a response.

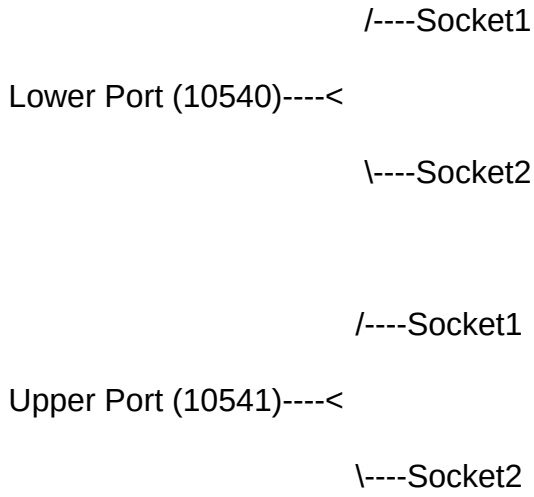
Recommended Work Practice: *It is not necessary to disconnect the socket once the ACK has been received. It may be more efficient and require less overhead to simply leave the socket open until the next message is transmitted, even if this is not immediate. If the socket is dropped the application should re-establish the socket before the next message is transmitted.*

Important Application Note: *When a socket is closed, either locally or remotely, care should be taken to ensure the socket is completely disconnected. This is a 4 step process involving communication between both machines. Socket tear down is normally taken care of at a level below application development. However, if problems are experienced establishing a socket between machines after at least one socket connection has been established and then dropped, this may be a sign the first socket was not properly closed. Check the status of all network connections on both machines. Indications of "fn_Wait2" on ports used for MOS communications are a sign of a problem.*

Both the NCS and MOS can originate messages. Transmitted messages require a response from the receiver before the transmitter will attempt to send the next message in the queue

belonging to that specific port (either upper or lower). Upper and lower port messages are not related so that while a machine is waiting for a response on the lower port it may continue to have a dialog on the upper port.

Note: "Two Ports - Four Sockets" Each pair of communicating machines uses two ports. Each machine must be able to accept and handle messages on a minimum of two sockets per port. Once established, socket connections do not need to be dropped and then re-established between messages. Generally, the acknowledgment of a message will be sent down the same socket on which the original message was received. However, machines should be able to handle situations in which each message arrives in a separate, discrete socket session (though this is not very efficient).



Note: "Multiple MOS Connections" Each machine (NCS and MOS) will be capable of establishing and maintaining connections to multiple systems of the opposite type. e.g. An NCS will be capable of connecting to multiple Media Object Servers. Media Object Servers will also be capable of connecting to multiple NCSs.

Message Acknowledgement

When a message is sent by a device to a target device, that device will not send another message to the target device on the same port until it receives an acknowledgement ("ACK") or error ("NACK") from the target device.

MOS enabled equipment and applications will retry when a timeout occurs. This applies to all messages on the same port.

Message acknowledgment on one port is independent of the flow of messages on the other.

If a message is not acknowledged, it and all subsequent waiting messages will be buffered.

Recommended Work Practice: *It is recommended that these messages be buffered in such a way that machine or application restart or reset will not destroy these buffered messages.*

2.2 Profile 1 – Basic Object Workflow

This profile allows a Media Object Server to push messages to other machines which describe objects contained on the Media Object Server.

In addition to support for Profile 0, these additional messages are required for support of Profile 1:

mosAck
mosObj
mosReqObj
mosReqAll
mosListAll

General Work Flow for Profile 1

- Media Object Servers push <mosObj> messages describing media to the NCS. This description includes a pointer to the media object as well as descriptive metadata.
- The NCS exposes <mosObj> information to users through lists, searches or other mechanisms in such a way that pointers representing the media objects are able to be moved or copied into text as Item References. Item References are derived from <mosObj> information.
- Optionally, an ActiveX control, provided by the Media Server Vendor, can be instantiated within the NCS UI. This ActiveX control has the ability to form an Item Reference and pass it to the NCS for integration as an Item Reference into a Story. (See the MOS v2.8.1 ActiveX Specification)
- Optionally, activating a pointer within the NCS (for example: in a list, embedded in a Story, etc.) instantiates an ActiveX control, provided by the Media Server Vendor, within the NCS UI. This ActiveX control provides, at a minimum, the ability to browse or display a proxy version of an object and also facilitates the integration of that object into an NCS Story as an Item Reference. (See the MOS v2.8.1 ActiveX Specification)
- The only MOS External Metadata (MEM) blocks that can be carried from the mosObj to the Item Reference are those with a <mosScope> of either "STORY" or "PLAYLIST".

Implementation Notes:

<mosObj> messages are sent from the Media Object Server to other applications to make them aware of objects stored on the Media Object Server.

Recommended Work Practice: *Other machines can populate their own database structures from the data contained within the <mosObj> messages they receive. It is possible then for these other applications to maintain a synchronized metadatabase describing objects contained within the Media Object Server.*

Other NCS applications have the opportunity to store and update a local metadatabase with this information. These applications can then perform searches on the local metadatabase and retrieve pointers to objects stored on the Media Object Server with matching records. These objects can then be referred to by unique <objID> without the immediate need to copy or move the essence of the object from the Media Object Server to the other applications.

Object Creation and Notification

When an object is created on a Media Object Server a <mosObj> message is pushed from the Media Object Server to a target application configured to receive this information. The initial <mosObj> message will have a <status> value of "NEW".

As metadata associated with an object stored on the Media Object Server changes, the Media Object Server needs to update the metadata already sent to other applications where it has been stored locally. Subsequent <mosObj> messages with updated metadata are sent from the Media Object Server with a <status> value of "UPDATED".

When the object is deleted from the Media Object Server or when the Media Object Server determines the object no longer has relevance to other devices, the Media Object Server sends a final <mosObj> message with a status of "DELETED".

Recommended Work Practice: *In many implementations both the target NCS and MOS sender need to have prior knowledge of each other stored in local configurations before messages can be meaningfully exchanged.*

It is possible, and sometimes desirable, to limit the number and type of objects which are pushed from the Media Object Server to other applications so that other applications are aware of only a subset of the entire population of objects stored on the Media Object Server.

Care should be taken to avoid unnecessary <mosObj> updates.

For instance, if an object is being ingested or recorded by a media server the duration of that object could be expected to be constantly changing as the recording continues. It is not reasonable to assume that other systems will want to receive updates every 1/10th of a second, every second, or even every few seconds when the recording is in progress. Such frequent updates, in most systems, would not be useful and would only serve to consume network, disk I/O and CPU bandwidth.

<mosObj> updates will be sent only at a frequency which is useful. There may be exceptions to this general rule and thus the protocol does not specifically define a maximum or minimum update frequency.

Object IDs Must Be Unique

<objID>'s are absolutely unique within the scope of the Media Object Server and are used to

unambiguously reference media stored on a specific server. The combination of <mosID> and <objID> will serve as a unique reference to an object on a specific server with an enterprise or multi-Media Object Server environment. The <objID> associated with an object will never change. Even if an object is moved from online, to nearline, to offline storage it will still use the same <objID> for unambiguous reference.

Applications should never, ever allow a user to enter or type an <objID>. Users should be presented indirect methods, such as lists, drop downs, drag and drop operations, etc. to choose and manipulate objects and object pointers.

Object Slugs are intended for display and use by Users

<objSlug>'s are the non-unique, human readable analog to the unique, machine assigned <objID>.

In short, <objSlug>'s are for humans. <objID>'s are for machines.

<objSlug>'s can optionally be assigned or changed as necessary by users. <objID>'s can never be assigned or modified by users directly.

Recommended Work Practice: *Display the <objSlug> to users and hide the <objID>.*

The <objSlug> field will contain the primary one line reference or name for an object exposed to users. This field is limited to 128 characters.

Abstracts and Descriptions may contain more information

The <mosAbstract> can contain a somewhat longer, but still brief, description of summary of the object which may applications may choose to alternately display.

The <description> will contain a verbose description of the object with information necessary to find the object via search functions.

MEM blocks carry Metadata Payloads

The <mosExternalMetadata> block (aka MOS MEM) is intended to be the mechanisms through which full and verbose descriptions of objects can be carried, which include the use of non-MOS schemas and tags for fielded data.

The MEM is the mechanism by which MOS supports Metadata Schema Standards such as NewsML, SMEF, SMPTE, MPEG7 and user specific schemas. MEM data blocks are not directly manipulated by the MOS Protocol and can be considered an information Payload which is carried between systems by the MOS Protocol.

Because MEM blocks can potentially carry large volumes of information, and because this information may not be relevant to all aspects of MOS applications, it makes sense to specifically state the scope of processes to which this information may be relevant. Thus, MEM blocks need only be carried as far into the process as is needed, and not unnecessarily consume network bandwidth, CPU or storage.

The <mosScope> tag describes to what extent within an NCS type workflow the MEM block will be carried.

A value of "OBJECT" implies that the MEM payload will be used for list and search purposes, but will not necessarily be carried into Stories or Play Lists/Content Lists.

A value of "STORY" implies the MEM payload will be used like the "OBJECT" case, but will be further carried into MOS Item References embedded in Stories. However, MEM Payloads with a <mosScope> of "STORY" are not carried into Play Lists/Content Lists.

A value of "PLAYLIST" implies the MEM payload will be used and included in all aspects of the production workflow, including embedding this information in the Item Reference in the Story and in Item References contained in the PlayList.

Exchanging Messages between MOS devices

To send a <mosObj> message from MOS to NCS:

1) The MOS device will open a socket on the lower port to the NCS if it is not already open

2) The MOS device will send the mosObj message

3) The MOS device will hold the socket open

4) The MOS device will wait for a mosAck message to be returned on the same socket before either dropping the socket or transmitting the next message.

5) The MOS device can optionally send <heartbeat> messages at regular intervals to the remote machine and look for a response.

Recommended Work Practice: *It is not necessary to disconnect the socket once the ACK has been received. It may be more efficient and require less overhead to simply leave the socket open until the next message is transmitted, even if this is not immediate. If the socket is dropped the application should re-establish the socket before the next message is transmitted.*

Important Application Note: When a socket is closed, either locally or remotely, care should be taken to ensure the socket is completely disconnected. This is a 4 step process involving communication between both machines. It is normally taken care of at a level below application development. However, if problems are experienced establishing a socket between machines after at least one socket connection has been established and then dropped, this may be a sign the first socket was not properly closed. Check the status of all network connections on both machines. Indications of "FIN_WAIT_2" or "CLOSE_WAIT" on ports used for MOS communications are a sign of a problem.

MOS message flow is strictly sequential

The Media Object Server will not send the next lower port message until the last message is acknowledged.

Flow of message traffic on the upper port is unrelated to acknowledgements on the lower port and vice versa.

If the value of <status> in the mosAck message is "NACK" then a more verbose error message is contained in <statusDescription>.

Data ownership and Synchronization

Metadata sent from the Media Object Server, including descriptions, pointers and MEM blocks, may not ever be changed by the NCS device. No mechanisms exist to reflect such changes back into the Media Object Server. Such an operation would be conceptually incompatible with the MOS Protocol.

If application developers wish to enable users at an NCS workstation to change this data they should provide such functionality in an ActiveX control, provided by the Media Object Server vendor, which can be instantiated within the NCS UI and provide the desired functionality. This is permitted since it is the vendor ActiveX control and not the NCS which is modifying the object information.

There may be times when an application may wish for the Media Object Server to send a full list of objects and descriptions. This may happen on initial installation and integration of systems, or at any other time when an NCS device wishes to synchronize its <mosObj> metadatabase from the Media Object Server. The <mosReqAll> and <mosListAll> messages are designed to facilitate this. There are methods enabled by these messages.

Method 1:

1. NCS sends a <mosReqAll> with a <pause> value of "0"

2. MOS replies with a <mosAck>, and then sends a series of <mosObj> messages encapsulated within a single <mosListAll> tag.

This enables the receiving NCS device to detect the start and end of the synchronization sequence. It can also potentially consume large amounts of network, CPU and disk I/O bandwidth.

Method 2:

1. NCS sends a <mosReqAll> with a <pause> value greater than "0"

2. MOS replies with a <mosAck>, and then sends a series of individual <mosObj> messages.

The value of <pause> indicates the number of seconds the MOS will pause in between <mosObj> messages intended for synchronization.

Other <mosObj> messages can be transmitted by the MOS between and concurrent with <mosObj> messages created as a result of the <mosReqAll> request. For instance, new objects, updates and deletions caused by work flow interaction.

This second method has the advantage of less impact on MOS and NCS resource

bandwidth, but there is no differentiation of <mosObj> messages intended for synchronization as opposed to those generated as a result of normal work flow.

The <mosReqObj> message is rarely used in actual operation but must be supported so that it can be used as a diagnostic tool.

2.3 Profile 2 – Basic Running Order/Content List Workflow

This Profile allows an NCS application to build dynamic Running Order/Content List sequences of Item References within a Media Object Server.

In addition to support for Profiles 0 and 1, these additional messages are required for support of Profile 2:

"roConstruction" family of messages

roAck
roCreate
roReplace
roDelete
roReq
roList
roMetadataReplace
roDelete
roStat
roElementAction
roItemStat
roReadyToAir

Included only for backwards compatibility:

roStoryAppend
roStoryInsert
roStoryReplace
roStoryMove
roStoryMoveMultiple
roStorySwap
roStoryDelete
roItemInsert
roItemReplace
roItemMoveMultiple
roItemDelete

General Work Flow for Profile 2

- Within the NCS Stories containing Item References can be placed into Running Orders (RO's or also referred to as Content Lists).

- The NCS then examines all Stories in a RO/Content List, extracts the Item References and uses these to build Playlists or Content Sequences within the parent Media Server machine.

- Playlists built in the Media Object Server by the NCS are filtered so they contain only Items which are stored on the target device.

  For instance, if a Running Order/Content List contains one story with embedded Item References from three different Media Object Servers, this single story would result in the creation of three Playlist/ContentLists – one in each of the Media Object Servers represented in the Story's Item References. Each Playlist/Content List would contain only one Item – the Item which references an Object stored on the local machine.

  In practice, this means that a Story which contains Item References for a Video Object, a CG Object and a Still Store Object will create three different playlists – one in the Video Server, one in the CG Server and one in the Still Store Server. Each playlist would contain a single Item.

  Exceptions to this rule are machines which conform to Profiles 4 and 5.

- Only MOS External Metadata (MEM) blocks included in Item References with a <mosScope> of "PLAYLIST" are included in these construction messages. MEM blocks with a <mosScope> of "STORY" are stripped and not sent.

- The NCS thus provides to the Parent Media Object Server a list pointing to Objects in the same order they are used in the Play List/Content List.

- The Media Object Server must always keep track of the list sequence as sent from the NCS without making changes to it. However, the MOS Device may choose to execute this list out of sequence without changing the list itself.

- As the content list is changed in the NCS, messages are dynamically sent from the NCS to the Media Object Server to insert, replace, delete, or otherwise resequence the contextual list of objects. This is a dynamic process and is not static.

- As objects identified by Item References are rendered or played to air, the status of these objects is sent from the MOS to the NCS via the <roItemStat> message.

- Finally, when the production of content within the NCS is complete, the NCS issues a final command to delete the RO/Content List.

Important Implementation Notes:

1) Both NCS and MOS device operation are expected to operate continuously without routine interruption.

2) Connectivity between NCS and MOS device is expected to operate continuously and without routine interruption.

3) Brief unplanned discontinuities in operation of either NCS or MOS, or connectivity between them, will be viewed as an error condition.

4) Discontinuities which result in un-ACK'd messages will be handled by buffering messages in the transmitter until they are ACK'd by the receiver.

5) Devices will not attempt to transmit further messages until the current message is acknowledged.

6) Message transmissions which do not receive a response will be retried at intervals until a response is received.

7) Message "ACK"s signify the message was received and was parsed correctly – and nothing more. These ACKs will be sent as soon as possible and without delay. They will be sent independent of and before any internal checks an application might make to, for example, check status of media objects, etc. If there is a problem with media object status, this will be communicated via the <roItemStat> message.

Very Important Note:

Changes to the list sequence are made relative to existing elements in the list. This allows a system to transmit only the changes to a list without sending the entire list, top to bottom. Thus, it is absolutely critical that all messages be applied in the order they are received. If a message in a sequence is not applied or "missed" then it is guaranteed that all subsequent messages will cause the sequence in the MOS to be even further out of sequence.

Recommended Work Practice: *It is recommended that after an object is inserted into a playlist by the NCS, either as a result of RO creation or RO modification, that the MOS system, in addition to providing the ACK, send a following <roItemStat> message to the NCS.*

Exchanging Messages between MOS devices

To send one of the "roConstruction" messages from an NCS to a MOS:

1) The NCS device will open a socket on the upper port to the MOS if it is not already open

2) The NCS will send the roConstruction message

3) The NCS will hold the socket open

4) The NCS will wait for a roAck message to be returned on the same socket before either dropping the socket or transmitting the next message.

5) The NCS can optionally send <heartbeat> messages at regular intervals to the remote machine

and look for a response.

Recommended Work Practice: *It is not necessary to disconnect the socket once the ACK has been received. It may be more efficient and require less overhead to simply leave the socket open until the next message is transmitted, even if this is not immediate. If the socket is dropped the application should re-establish the socket before the next message is transmitted. It is a good idea to establish and maintain the socket connection continuously as this gives the other application the opportunity to monitor continuity.*

Important Application Note: *When a socket is closed, either locally or remotely, care should be taken to ensure the socket is completely disconnected. This is a 4 step process involving communication between both machines. It is normally taken care of at a level below application development. However, if problems are experienced establishing a socket between machines after at least one socket connection has been established and then dropped, this may be a sign the first socket was not properly closed. Check the status of all network connections on both machines. Indications of "FIN_WAIT_2" or "CLOSE_WAIT" on ports used for MOS communications are a sign of a problem.*

MOS message flow is strictly sequential

The Media Object Server will not send the next upper port message until the last message is acknowledged.

Flow of message traffic on the lower port is unrelated to acknowledgements on the upper port and vice versa.

If the value of <status> in the roAck message is "NACK" then a more verbose error message is contained in <statusDescription>.

Recommended Work Practice: *Some devices wait only a finite time for a response. If this response is not received they will transmit an unacknowledged message again. It is recommended that all devices provide acknowledgement of messages within a maximum of 60 seconds of receipt. The faster ACK messages are received the more efficient integrated systems will function. Please keep in mind the adverse effects unnecessary cumulative delayed responses will have in high message environment.*

If a MOS device receives a message from the NCS which references an <roID> or <storyID> which is not known to the MOS within the context of the application of the message, then the MOS device will assume there has been a prior error in communication with the NCS. The MOS will then request a full list of the Running Order/Content List from the NCS via the <roReq> message to the NCS and the <roList> response to the MOS.

For instance, if a MOS device receives an <roElementAction> message which references an unknown <roID>, <storyID> or <itemID>, the MOS device will send an <roReq> message to the NCS which includes the <roID>. The NCS will then respond with an <roList> message which includes the entire current context of the RO/Content List.

Recommended Work Practice: *"ro" messages allow the NCS to dynamically transmit a sequence of objects to a MOS device. The MOS device then determines what to do with this list. In the case of video and audio equipment, this list from the NCS often represents the sequence to be played on air. Just because content is presented in an ordered list does not imply an absolute need to actually*

execute the list in order. Specific applications may allow users to "hop around" and execute the list out of order without actually changing the list.

Important Note: If for any reason the sequence of the Running Order/Content List on the MOS device is intentionally changed such that it no longer represents the sequence as transmitted from the NCS, the MOS device will immediately send a series of <roItemStat> messages to the NCS with a <status> of "DISCONNECTED" and ACK all subsequent "ro" messages with a <status> of "DISCONNECTED".

The MOS device can recover synchronization with the NCS by sending an <roReq> message to the NCS and receiving a full <roList> message in return. Information in the <roList> message will be used to replace the list previously modified through user input in the MOS device.

The MOS device can optionally send an <roStat> message to the NCS indicating the RO/Content List is under manual or NCS control.

The <roElementAction> message in MOS v2.8 functionally replaces the following messages used in older versions of the protocol:

roStoryAppend
roStoryInsert
roStoryReplace
roStoryMove
roStoryMoveMultiple
roStorySwap
roStoryDelete
roItemInsert
roItemReplace
roItemMoveMultiple
roItemDelete

Important Note:

In future versions of the MOS Protocol support for these messages will be dropped. At present, applications and devices claiming support for MOS v2.8.1 Profile 2 must support all of these messages for backwards compatibility plus the functionally equivalent <roElementAction> which will be forward compatible.

The <roReadyToAir> message, sent from the NCS to the MOS device, is used to indicate that a specified RO/Content List is editorially approved or not approved for output. This message has no direct impact on MOS message flow or sequencing. It is up to individual vendors and customers to determine what work practices and functionality may be linked to this message.

2.4 Profile 3 – Advanced Object Based Workflow

This Profile allows an NCS to request a Media Object Server create a new object with specific properties, attributes and metadata description. It also allows a Media Object Server to replace an Item Reference embedded within a specific Story/Running Order, and request that a MOS device return a list of mosObj descriptions which meet certain search criteria..

In addition to support for Profiles 0, 1 and 2, these additional MOS Protocol Messages must be supported for Profile 3:

mosObjCreate

mosItemReplace

mosReqObjList family of messages

mosReqSearchableSchema
mosListSearchableSchema
mosReqObjList
mosObjList

General Work Flow for Profile 3

The <mosObjCreate> message

- An NCS or NCS user wishes to create a new object on a Media Object Server.
- The NCS sends a request to the Media Object Server, via the <mosObjCreate> message, to create a new Object or Place Holder to the Media Object Server.
- Within the <mosObjCreate> message the NCS sends a description and metadata for the new object.
- The Media Object Server responds with a <mosAck> message, which includes:
  - If the object was created, contains a <status> value of "ACK" and a <statusDescription> which contains the new <objID> value.
  - If the object was not created, contains a <status> value of "NACK" and a <statusDescription> which contains a textual error message.
- If the Object was created as a result of the request, the Media Object Server also sends a new <mosObj> message on the lower port. This message will contain the full object description, pointer and metadata.
- **Recommended Work Practice:** *Media Objects created with this message may be either real or virtual. What really matters to work flow is that an <objID> be returned which will eventually reference the actual media object.*

The <mosItemReplace> message

- A Media Object Server wishes to replace all or part of an Item Reference embedded in Story.
- Story data is "owned" by the NCS and cannot be changed by the Media Object Server.
- Item Data that is copied from Object Data is "owned" by the Media Object Server and can be changed, even though it is embedded in a Story.
- Although the Item Reference can be changed by the Media Object Server, its position within the Story cannot.
- The Media Object Server sends a <mosItemReplace> message, referencing the <roID>, <storyID> and <itemID> which points to the Item Reference to be changed.
- The NCS will replace or merge the data in the <item> structure. (See the protocol definition for <mosItemReplace> for specifics)
- The NCS will respond with an <roAck> message, and if successful:
  - <roAck> will have a <status> value of "ACK"
  - The NCS will send a further and independent <roStoryReplace> message, which the Media Object Server must accept and ACK.
  - If Profile 4 is supported, the NCS will also send an independent <roStorySend> message which must also be accepted and ACK'd.

The <mosReqObjList> family of messages

- An NCS Server or NCS Client, communicating with the MOS on the new port 10542, wishes to receive a list of mosObj messages which meet certain search criteria.
- For a general search, the NCS or NCS Client sends a mosReqObjList message with a simple search string as the value of the <generalSearch> tag.
  - Logical operators are allowed in this string.
  - The MOS devices will search its database for this general search term. The internal data fields to which the MOS applies this search term is determined by the MOS.
- For a field specific search, the NCS or NCS client must first ask the MOS device for a list of field definitions, in the form of a schema
  - The NCS or NCS client sends a mosReqSearchableSchema message to the MOS.
  - The MOS returns a list of schema pointers, in the form of URI's, to the NCS or NCS client in the mosListSearchableSchema message.
  - If the mosListSearchableSchema message contains no URI's, then the NCS should recognize that the MOS device does not support field specific searching.
  - The NCS or NCS client then retrieves the schema and specifies field(s) to search with the value of the <searchField> tag(s) in the mosReqObjList message.
  - Multiple <searchField> tags can be included in within a single <searchGroup> structure. All

<searchField> tags will be logically "AND"ed.
    - Multiple <searchGroup> structures can be included. These will be logically "OR"ed.
- The MOS device then returns a sequence of mosObj messages which meet the search criteria.
    - These messages are encapsulated in the mosObjList message.
    - The information in the mosObj messages, including objIDs can be used as normal by the NCS or NCS Client.

It is recommended that this family of messages be used to re-synchronize the NCS and MOS devices instead of the older mosReqAll message.

2.5 Profile 4 – Advanced RO/Content List Workflow

This profile enables a device to request a full list of all Running Orders/Content Collections under MOS control in an NCS. It also allows any device to receive the full context of data associated with a Running Order/Content List and send contextual "cues" to the parent Media Object Server.

In addition to support for Profiles 0, 1 and 2, these additional MOS Protocol Messages must be supported for Profile 4:

roReqAll
roListAll
roStorySend

General Work Flow for Profile 4

<roReqAll> and <roListAll> are functionally similar to <roReq> and <roList>.

<roReqAll> is a request from the MOS device to the NCS for a list of all MOS Active Running Orders. <roListAll> is the response from the NCS. The list contains the <roID>, <roSlug> and other metadata. For a full listing of the contents of the RO the MOS device must issue a subsequent <roReq> using a <roID> from the <roListAll> message.

<roStorySend> is a method by which the NCS can send the complete body of a story to another device.

This is useful for prompters and publishing devices which must be aware not only of the sequence of stories but also the full body of text and metadata for each story, which is not otherwise sent.

Recommended Work Practice: *To send a complete list of stories associated with a Running Order along with the bodies of all Stories, the NCS must first construct a playlist in the Media Object Server using the "roConstruction" messages, taking care to send *all* <story> structures, not just Stories which contain <item> structures belonging to a specific device. (Normal practice is to use "roConstruction" messages to send only <story> structures that contain <items> belonging to the parent Media Object Server.) This is followed by an*

<roStorySend> message for each of the Stories in the NCS Running Order/Content Collection. In this way changes to the order of the Stories can be communicated without retransmitting Story objects. Likewise, a Story can be updated without making a change to the Story Sequence.

When changing the sequence of an Running Order/Content List which is linked to a MOS device via "roConstruction" and <roStorySend> messages, it is important to use the <roElementAction> message to effect moves in the story sequence, rather than using options for delete and insert. Once a story is deleted from the list the receiving device may assume the body of the story is no longer needed and delete it, thus requiring an unnecessary and repetitive <roStorySend> message after the <roElementAction> "insert" command.

2.6 Profile 5 – Item Control

This profile enables applications to send "cue" and control commands to Media Object Servers

In addition to support for Profiles 0, 1 and 2, these additional MOS Protocol Messages must be supported for Profile 5:

roItemCue
roCtrl

<roItemCue> is a method used to signal or "cue" a parent MOS device at a specific time.

This message is for notification only, but can be used by the parent Media Object Server to allow other applications to trigger rendering, publishing or output of a specific Item Reference to an Object at a specific time, which can be in the future. This is not a specific command to play or take action on an object.

<roCtrl> is a method used to signal or "cue" a parent MOS device at a specific time.

This message allows other devices to send specific and unambiguous "PLAY" "EXECUTE" "PAUSE" "STOP" AND "SIGNAL" commands to a Media Object Server. Though these messages were originally intended for control of audio and video servers, their application should not be thought of as limited to these types of output devices.

Application Note: The use and timing of these messages is subject to network propagation and application processing latency. Synchronicity and frame accuracy can be achieved in the application of the <roItemCue> message if an event to which it is linked can be anticipated by an

amount of time equal to or greater than total link latency. The <roEventTime> can then be set to an appropriate future value which in effect leads system latency.

2.7 Profile 6 – MOS Redirection

This Profile provides a mechanism for <item> structures containing media objects from one server to be meaningfully included in messages sent to a server other than the one on which they are immediately stored. This information enables servers to automate the transfer of these objects between machines, using methods independent of the MOS Protocol.

Profile 6 requires full support for Profiles 0, 1 and 2 and can be applied to Profiles 3, 4 and 5

Fully Qualified MOS ID

Profile 6 does not include any additional MOS messages. However, it does require that all MOS device compatible with Profile 6 use a specific naming convention for <mosID>'s and <ncsID>'s of this form:

<family>.<machine>.<location>.<enterprise>.mos

Where <location> and <enterprise> are optional.

This is called a "Fully Qualified MOS ID"

For example, these are valid Fully Qualified MOS ID's:

aveed.server2.camden.cbs.mos

tornado.mach2.wjla.allbritton.mos

Quantuml.VidServ2.mos

Sonny.point77.city.company.mos

Using this naming convention, it is possible for a machine to determine whether an object is stored locally or on another machine of the same family or compatible family, and for that machine to make separate arrangements for the transfer of the referenced object to the local machine.

This functionality can be extended to transfer material between machines located in the same building, different buildings or different cities.

The transfer mechanism is separate from the MOS Protocol, which only provides the Fully Qualified

MOS ID.

Vendors claiming compatibility with Profile 6 must support, at a minimum, automated transfer of objects between machines of the same family within the vendor's product line.

3. Media Object Server Protocol Message Definition

In the Structural Outline sections below use of "?" specifies optional element, "+" specifies one or more of the element, and "*"specifies zero or more of the element. Elements without any of these three special characters are required.
Tags enclosed in parenthesis and separated by "|" represent a selection.
The Syntax section shows the definition of non-terminals for this message from the DTD.
Examples shown are representative of syntax only and do not represent samples from any system.

3.1 Basic Object Communication

3.1.1 mosAck - Acknowledge MOS Object Description

Purpose

The MOS Acknowledgement for the MOS OBJ message.

Response

N/A

Port

MOS Lower Port (10540) - Media Object Metadata

Structural Outline

mos
- mosID
- ncsID
- messageID
- mosAck
- objID
- objRev
- status
- statusDescription

Syntax

```
<!ELEMENT mosAck (objID, objRev, status, statusDescription)>
```

Example

```
<mos>
  <mosID>aircache.newscenter.com</mosID>
  <ncsID>ncs.newscenter.com</ncsID>
  <messageID>99</messageID>
  <mosAck>
    <objID>M000123</objID>
    <objRev>1</objRev>
    <status>ACK</status>
    <statusDescription></statusDescription>
  </mosAck>
</mos>
```

3.1.2 mosObj - MOS Object Description

Purpose

Contains information that describes a unique MOS Object to the NCS. The NCS uses this information to search for and reference the MOS Object.

<objGroup> tag

No specific values for this element are officially defined. Definition of values is left to the configuration and agreement of MOS connected equipment. The intended use is for site configuration of a limited number of locally named storage folders in the NCS or MOS, such as "ControlA", "ControlB" , "Raw", "Finished", etc. For editorially descriptive "category" information, it is suggested that the <mosExternalMetadata> block be used.

External Metadata Block

This data block can appear in several messages as a mechanism for transporting additional metadata, independent of schema or DTD. When found within the mosObj message, this block carries data defined external to the MOS Protocol.

External Metadata <scope> tag

The value of the <scope> tag implies through what production processes this information will travel.

A scope of "object" implies this information is generally descriptive of the object and appropriate for queries. The metadata will not be forwarded into Stories or Playlists.

A scope of "story" suggests this information may determine how the Object is to be applied in a Story. For instance, Intellectual Property Management. This information will be forwarded with the contents of a Story.

A scope of "playlist" suggests this information is specific to describing how the Object is to be published, rendered, or played to air and thus, will be included in the roCreate Play List Construction and roStorySend

messages.

Scope allows systems to, optionally, roughly filter external metadata and selectively apply it to different production processes and outputs. Specifically, it is neither advisable nor efficient to send large amounts of inappropriate metadata to the Playlist in roCreate messages. In addition to these blocks of data being potentially very large, the media Object Server is, presumably, already aware of this data.

Response

mosAck

Port

MOS Lower Port (10540) - Media Object Metadata

Structural Outline

mos
- mosID
- ncsID
- messageID
- mosObj
  - objID
  - objSlug
  - mosAbstract?
  - objGroup?
  - objType
  - objTB
  - objRev
  - objDur
  - status
  - objAir
  - createdBy
  - created
  - changedBy
  - changed
  - description
    - (p | em | tab)*
  - mosExternalMetadata*
    - mosScope?
    - mosSchema
    - mosPayload

Syntax

```
<!ELEMENT mosObj (objID, objSlug, mosAbstract?, objGroup?, objType, objTB, objRev, objDur, status,
 objAir, createdBy, created, changedBy, changed, description, mosExternalMetadata*)>
<!ELEMENT description (#PCDATA | p | em | tab)*>
<!ELEMENT p (#PCDATA | em | tab)*>
<!ELEMENT mosExternalMetadata (mosScope?, mosSchema, mosPayload)>
<!ELEMENT mosScope (#PCDATA)>
<!ELEMENT mosSchema (#PCDATA)>
<!ELEMENT mosPayload ANY>
```

```
<!ELEMENT messageID (#PCDATA)>
```

Example

```
<mos>
   <mosID>aircache.newscenter.com</mosID>
   <ncsID>ncs.newscenter.com</ncsID>
   <messageID>34</messageID>
   <mosObj>
      <objID>M000123</objID>
      <objSlug>Hotel Fire</objSlug>
      <mosAbstract>
         <b>Hotel Fire</b>
         <em>vo</em>
          :30
      </mosAbstract>
      <objGroup>Show 7</objGroup>
      <objType>VIDEO</objType>
      <objTB>60</objTB>
      <objRev>1</objRev>
      <objDur>1800</objDur>
      <status>NEW</status>
      <objAir>READY</objAir>
      <createdBy>Chris</createdBy>
      <created>1998-10-31T23:39:12</created>
      <changedBy>Chris</changedBy>
      <changed>1998-10-31T23:39:12</changed>
      <description>
         <p>
            Exterior footage of
            <em>Baley Park Hotel</em>
             on fire with natural sound. Trucks are visible for the first portion of the clip.
            <em>CG locator at 0:04 and duration 0:05, Baley Park Hotel.</em>
         </p>
         <p>
            <tab/>
            Cuts to view of fire personnel exiting hotel lobby and cleaning up after the fire is out.
         </p>
         <p>
            <em>Clip has been doubled for pad on voice over.</em>
         </p>
      </description>
      <mosExternalMetadata>
         <mosScope>STORY</mosScope>
         <mosSchema>http://MOSA4.com/mos/supported_schemas/MOSAXML2.08</mosSchema>
         <mosPayload>
            <Owner>SHOLMES</Owner>
            <ModTime>20010308142001</ModTime>
            <mediaTime>0</mediaTime>
            <TextTime>278</TextTime>
            <ModBy>LJOHNSTON</ModBy>
            <Approved>0</Approved>
            <Creator>SHOLMES</Creator>
         </mosPayload>
      </mosExternalMetadata>
   </mosObj>
</mos>
```

3.1.3 mosReqObj - Request Object Description

Purpose

Message used by NCS to request object description.

Response

mosObj - if objID is found
mosAck - otherwise

Port

MOS Lower Port (10540) - Media Object Metadata

Structural Outline

mos
- mosID
- ncsID
- messageID
- mosReqObj
  - objID

Syntax

```
<!ELEMENT mosReqObj (objID)>
```

Example

```
<mos>
   <mosID>aircache.newscenter.com</mosID>
   <ncsID>ncs.newscenter.com</ncsID>
   <messageID>654</messageID>
   <mosReqObj>
      <objID>M000123</objID>
   </mosReqObj>
</mos>
```

3.2 Object Resynchronization/Rediscovery

3.2.1 mosReqAll - Request All Object Data from MOS

Purpose

Method for the NCS to request the MOS send it mosObj messages for every Object in the MOS. Pause, when greater than zero, indicates the number of seconds to pause between individual mosObj messages. Pause of zero indicates that all objects will be sent using the mosListAll message..

Response

mosAck - which then initiates one of the following:

mosListAll - if pause = 0
mosObj - if pause > 0

Port

MOS Lower Port (10540) - Media Object Metadata

Structural Outline

mos
- mosID
- ncsID
- messageID
- mosReqAll
  - pause

Syntax

```
<!ELEMENT mosReqAll (pause)>
```

Example

```
<mos>
   <mosID>aircache.newscenter.com</mosID>
   <ncsID>ncs.newscenter.com</ncsID>
   <messageID>234</messageID>
   <mosReqAll>
      <pause>0</pause>
   </mosReqAll>
</mos>
```

3.2.2 mosListAll - Listing of All Object Data from MOS

Purpose

Send MOS object descriptions in a format similar to mosObj messages from the MOS to the NCS.
 mosListAll is initiated by a properly Ack'd mosReqAll message from the NCS.

Response

mosAck

Port

MOS Lower Port (10540) - Media Object Metadata

Structural Outline

mos
- mosID
- ncsID
- messageID
- mosListAll
  - mosObj*
    - objID
    - objSlug
    - mosAbstract?
    - objGroup?
    - objType
    - objTB

objRev
objDur
status
objAir
createdBy
created
changedBy
changed
description
(p | em | tab)*
mosExternalMetadata*
mosScope?
mosSchema
mosPayload

Syntax

```
<!ELEMENT mosListAll (mosObj*)>
<!ELEMENT mosObj (objID, objSlug, mosAbstract?, objGroup?, objType, objTB, objRev, objDur, status,
 objAir, createdBy, created, changedBy, changed, description, mosExternalMetadata*)>
<!ELEMENT description (#PCDATA | p | em | tab)*>
<!ELEMENT p (#PCDATA | em | tab)*>
<!ELEMENT mosExternalMetadata (mosScope?, mosSchema, mosPayload)>
<!ELEMENT mosScope (object | story | playlist)>
<!ELEMENT mosSchema (#PCDATA)>
<!ELEMENT mosPayload ANY>
<!ELEMENT messageID (#PCDATA)>
```

Example

```
<mos>
   <mosID>aircache.newscenter.com</mosID>
   <ncsID>ncs.newscenter.com</ncsID>
   <messageID>2010</messageID>
   <mosListAll>
    <mosObj>
      <objID>M000123</objID>
      <objSlug>HOTEL FIRE</objSlug>
      <objGroup>RAW</objGroup>
      <objType>VIDEO</objType>
      <objTB>60</objTB>
      <objRev>3</objRev>
      <objDur>1530</objDur>
      <status>UPDATED</status>
      <objAir>READY</objAir>
      <createdBy>Chris</createdBy>
      <created>1998-10-31T23:39:12</created>
      <changedBy>Chris</changedBy>
      <changed>1998-11-01T14:35:55</changed>
      <description>
         <p>
            Exterior footage of

            <em>Baley Park Hotel</em>

             on fire with natural sound. Trucks are visible for the first portion of the
 clip.

            <em>CG locator at 0:04 and duration 0:05, Baley Park Hotel.</em>
         </p>
         <p>
            <tab/>

            Cuts to view of fire personnel exiting hotel lobby and cleaning up after the fire is
 out.
```

```
         </p>
         <p>
            <em>Clip has been doubled for pad on voice over.</em>
         </p>
      </description>
    </mosObj>
    <mosObj>
      <objID>M000224</objID>
      <objSlug>COLSTAT MURDER:VO</objSlug>
      <objType>VIDEO</objType>
      <objTB>60</objTB>
      <objRev>4</objRev>
      <objDur>800</objDur>
      <status>UPDATED</status>
      <objAir>READY</objAir>
      <createdBy>Phil</createdBy>
      <created>1998-11-01T15:19:01</created>
      <changedBy>Chris</changedBy>
      <changed>1998-11-01T15:21:15</changed>
      <description>VOICE OVER MATERIAL OF COLSTAT MURDER SITES SHOT ON 1-NOV.</description>
      <mosExternalMetadata>
         <mosScope>STORY</mosScope>
                        <mosSchema>http://MOSA4.com/mos/supported_schemas/MOSAXML2.08</mosSchema>
         <mosPayload>
            <Owner>SHOLMES</Owner>
            <ModTime>20010308142001</ModTime>
            <mediaTime>0</mediaTime>
            <TextTime>278</TextTime>
            <ModBy>LJOHNSTON</ModBy>
            <Approved>0</Approved>
            <Creator>SHOLMES</Creator>
         </mosPayload>
      </mosExternalMetadata>
    </mosObj>
   </mosListAll>
</mos>
```

mosReqObjList family of messages

Purpose

To retrieve only selected object descriptions from a MOS

Port

10542

Communication Type

NCS SERVER to MOS and NCS CLIENT to MOS

Messages in this family

mosReqSearchableSchema
mosListSearchableSchema
mosReqObjList
mosObjList

Workflow

1) NCS sends a mosReqSearchableSchema message to the MOS.

2) MOS responds with a mosListSearchableSchema message.

3) NCS can then perform a query by sending a mosReqObjList message, using one or both of the Search Options below

a) Search Method #1 (Simple): Perform a general search based on the textual content of the <generalSearch> field. The following six examples illustrate valid values for this field.

```
<generalSearch>man</generalSearch>
<generalSearch>man dog</generalSearch>
<generalSearch>man and dog</generalSearch>
<generalSearch>man not dog</generalSearch>
<generalSearch>man or dog</generalSearch>
<generalSearch>man and dog not poodle</generalSearch>
```

Note: only one <generalSearch> tag is allowed per message

The simple method will search all default fields in the MOS database, as defined by the MOS vendor. Only one <generalSearch> field may be present.

b) Search Method #2 (Complex): Perform a specific query based on the value of selected external metadata (MEM) fields. These queries are wrapped in the <searchGroup> tag. The <searchGroup> structure can be used with or without <generalSearch>, as in Method #1. The following is a valid example:

```
<searchGroup>
   <searchField XPath="/Presenter [. ='Bob']" sortByOrder="1"/>
   <searchField XPath="/Slug [.='Dog Abuse']"/>
   <searchField XPath="/Video/Length [.&gt;60 AND &lt;120]" sortByOrder="2" sortType="DESCENDING"/>
   <searchField XPath="Producer [.!='Susan']" sortByOrder="3"/>
</searchGroup>

<searchGroup>
   <searchField XPath="/Presenter [. ='Jennifer']" sortByOrder="1"/>
   <searchField XPath="/Slug [.='Big Mice in City']"/>
   <searchField XPath="/Video/Length [.&gt;60 AND &lt;120]" sortByOrder="2" sortType="DESCENDING"/>
   <searchField XPath="Producer [.!='Susan']" sortByOrder="3"/>
</searchGroup>
```

Multiple <searchGroup> structures are logically "OR"ed with each other.

The attributes included in each <searchField> tag were derived from the schema returned in the initial mosListSearchableSchema message.

mosSchema must be an HTTP pointer to a valid schema. This schema is passed to the NCS by the MOS via the mosListSearchableSchema message.

Note: The schema must be valid XML.

searchField must contain an XPath statement which conforms to a subset of the W3C XPath 1.0 specification which can be found here: http://www.w3.org/TR/xpath

Minimum implementations must support Basic XPath Expressions needed to process Abbreviated

Syntax for Location Paths with Location Steps that may contain Predicates with Operators "and", "or", "<", ">", ">=", "<=", "=", "!=", and the following functions:

1. **String Functions**

Function	Parameters	Return Type	Description
`String`	object?	String	Converts to string

2. **Number Functions**

Function	Parameters	Return Type	Description
`Number`	object?	Number	Converts to a number

3. **Boolean Functions**

Function	Parameters	Return Type	Description
`Boolean`	object	Boolean	Converts to a boolean value
`False`		Boolean	Returns false
`Not`	boolean	Boolean	Inverts a boolean value
`True`		Boolean	Returns true

XPath search requests are assumed to be case sensitive.

Rules on Sorting are as follows:

- All fields of the same name have to have the same sortByOrder and sortType attributes for the same fieldname. This is why, for example, /Presenter is the same in the first searchGroup as it is in the second.
- No two unlike fields can share the same sort order. Presenter can't be sortByOrder = 1 in the same request as Producer has a sortByOrder of 1.
- The MOS determines sorting rules according to the natural language of the MOS System Environment

<searchField>'s within the same <searchGroup> are logically joined (AND'ed).

Multiple <searchGroup>'s are allowed. Each <searchGroup> is logically "OR"ed with the others.

A maximum of six <searchGroup> structures is recommended.

4) The MOS returns a list of mosObj messages, encapsulated in the mosObjList message, to the NCS. The number and sequence of these messages is specified by the NCS.

5) The NCS can handle the returned mosObj messages as normal, meaning the objIDs they hold can be validly used with ActiveX controls, within item references, with playlist construction, etc.

Note: Use of the mosReqObjList group of messages between **NCS SERVER** and MOS and **NCS CLIENT** and MOS should take place on port 10542. Because of the potential for this family of messages to generate large amounts of network traffic and consume application bandwidth, this message has been assigned a separate and specific port in order to minimize potential impact on mission critical operations taking place on ports 10540 and 10541. Other future messages may also share port 10542.

General notes

Both Search Methods can be used together, in which case the <generalSearch> is logically joined (AND'ed) with the <searchGroup> results. The Simple and Complex methods may also be used separately and independent of each other.

The <generalSearch> tag must always be present, even if Null.

For both methods the NCS can specify the number of search results to return, which is the difference between the integer values of <listReturnStart> and <listReturnEnd>.

The <listReturnStart> tag must always be present and must always have a value of 1 or greater.

The <ListReturnEnd> tag must always be present, but a value is optional. If a value is present, it must be greater than or equal to <listReturnStart>.

Omission of a value for <listReturnEnd> implies that *all* possible search results should be returned. Care should be taken when implementing this option to avoid returning more pointers than is necessary which may overwhelm network or application bandwidth.

Paging is supported by supplying chained values for <listReturnStart> and <listReturnEnd>, e.g. 1-20, 21-40, 41-80. These values represent requests in the mosReqObjList message. In the mosObjList message these values indicate the actual number of objects returned.

<listReturnTotal>applies only to the mosObjList message and this tag is required. If the value is null, this indicates generally more than one object has been found. A non zero value indicates the total number of objects which meet the search criteria and can be returned.

A zero value of <listReturnTotal> indicates no objects were located which meet the search criteria. In this case the values of <listReturnStart> and <listReturnEnd> would also be zero.

<listReturnStatus> should contain a human readable status message, of 128 max characters, which indicates the reason for a zero value in <listReturnTotal>. <listReturnStatus> can optionally be used to return additional human readable status information when <listReturnTotal> is a non-zero value.

Cached searching is enabled by the mandatory use of the <queryID> field, which is defined as a 128 character ID unique within the scope of the NCS system. Though the full values of <generalSearch> and/or <searchGroup>'s must still be supplied in each and every <mosReqObjList> message, the <queryID> value provides a short cut for the MOS device, which may choose to buffer the results of first query and then return additional paged requests for the same query from a buffer.

3.2.3 mosReqSearchableSchema

Purpose

A mechanism for the NCS to request the MOS send a pointer to a schema in which searchable fields are defined by the MOS device.

Port

10542

Communication Type

NCS SERVER to MOS and NCS CLIENT to MOS

Response

mosListSearchableSchema

Structural Outline

mos
  mosID
  ncsID
  messageID
  mosReqSearchableSchema

Syntax

```
<!ELEMENT mosReqSearchableSchema EMPTY>
```

Example

```
<mos>
   <mosID>aircache.newscenter.com</mosID>
   <ncsID>ncs.newscenter.com</ncsID>
   <messageID>9012</messageID>
   <mosReqSearchableSchema/>
</mos>
```

3.2.4 mosListSearchableSchema

Purpose

A mechanism for the MOS to send a pointer to a schema in which searchable fields are defined for the NCS device.

Port

10542

Communication Type

MOS to NCS SERVER and MOS to NCS CLIENT

Response

None – this is a response to mosReqSearchableSchema

Structural Outline

mos
  mosID
  ncsID
  messageID
  mosListSearchableSchema
    mosSchema

Syntax

```
<!ELEMENT mosListSearchableSchema (mosSchema)>
```

Example

```
<mos>
   <mosID>aircache.newscenter.com</mosID>
   <ncsID>ncs.newscenter.com</ncsID>
   <messageID>2782</messageID>
   <mosListSearchableSchema>
      <mosSchema>http://MOSA4.com/mos/supported_schemas/MOSAXML2.08</mosSchema>
   </mosListSearchableSchema>
</mos>
```

3.2.5 mosReqObjList

Purpose

To retrieve only selected object descriptions from a MOS.

Port

10542

Communication Type

NCS SERVER to MOS and NCS CLIENT to MOS

Response

mosObjList

Structural Outline

mos
  mosID
  ncsID

messageID
mosReqObjList
queryID
listReturnStart
listReturnEnd
generalSearch
mosSchema
searchGroup*
searchField+
(XPath, sortByOrder, sortType)

Syntax

```
<!ELEMENT mosReqObjList (queryID, listReturnStart, listReturnEnd, generalSearch, mosSchema,
 searchGroup*)>
<!ELEMENT queryID (#PCDATA)>
<!ELEMENT generalSearch (#PCDATA)>
<!ELEMENT listReturnStart (#PCDATA)>
<!ELEMENT listReturnEnd (#PCDATA)>
<!ELEMENT searchGroup (searchField+)>
<!ELEMENT searchField EMPTY>
<!ATTLIST searchField
      XPath CDATA #REQUIRED
      sortByOrder CDATA #IMPLIED
      sortType CDATA # IMPLIED
>
```

Example

```
<mos>
   <mosID>aircache.newscenter.com</mosID>
   <ncsID>ncs.newscenter.com</ncsID>
   <messageID>6666</messageID>
   <mosReqObjList>
      <queryID>123439392039393ade0393zdkdls</queryID>
      <listReturnStart>1</listReturnStart>
      <listReturnEnd/>
      <generalSearch>man bites dog</generalSearch>
      <mosSchema>http://MOSA4.com/mos/supported_schemas/MOSAXML2.08</mosSchema>
      <searchGroup>
         <searchField XPath="/Presenter [. =’Bob’]" sortByOrder="1"/>
         <searchField XPath="/Slug [.=’Dog Abuse’]"/>
         <searchField XPath="/Video/Length [.&gt;60 AND &lt;120]" sortByOrder="2"
 sortType="DESCENDING"/>
         <searchField XPath="Producer [.!=’Susan’]" sortByOrder="3"/>
      </searchGroup>
      <searchGroup>
         <searchField XPath="/Presenter [. =’Jennifer’]" sortByOrder="1"/>
         <searchField XPath="/Slug [.=’Big Mice in City’]"/>
         <searchField XPath="/Video/Length [.&gt;60 AND &lt;120]" sortByOrder="2"
 sortType="DESCENDING"/>
         <searchField XPath="Producer [.!=’Susan’]" sortByOrder="3"/>
      </searchGroup>
   </mosReqObjList>
</mos>
```

3.2.6 mosObjList

Purpose

To retrieve only selected object descriptions from a MOS

Port

10542

Communication Type

MOS to NCS SERVER and MOS to NCS CLIENT

Response

None – this is a response to mosReqObjList

Structural Outline

mos
  mosID
  ncsID
        messageID
        mosObjList
          queryID
          listReturnStart
          listReturnEnd
          listReturnTotal
          listReturnStatus?
          list?
            mosObj+

Syntax

```
<!ELEMENT mosObjList (queryID, listReturnStart, listReturnEnd, listReturnTotal, listReturnStatus?,
 list?)>
<!ELEMENT listReturnTotal (#PCDATA)>
<!ELEMENT listReturnStatus (#PCDATA)>
<!ELEMENT list (mosObj+)>
```

Example

```
<mos>
   <mosID>aircache.newscenter.com</mosID>
   <ncsID>ncs.newscenter.com</ncsID>
   <messageID>321</messageID>
   <mosObjList>
      <queryID>A392938329kdakd2039300d0s9l3l9d0bzAQ</queryID>
      <listReturnStart>1</listReturnStart>
      <listReturnEnd>20</listReturnEnd>
      <listReturnTotal>128</listReturnTotal>
      <list>
         <mosObj>
            <objID>M000121</objID>
            <objSlug>Hotel Fire</objSlug>
            <mosAbstract>
               <b>Hotel Fire</b>
               <em>vo</em>

          :30

      </mosAbstract>
            <objGroup>Show 7</objGroup>
            <objType>VIDEO</objType>
            <objTB>60</objTB>
            <objRev>1</objRev>
```

```
            <objDur>1800</objDur>
            <status>NEW</status>
            <objAir>READY</objAir>
            <createdBy>Chris</createdBy>
            <created>1998-10-31T23:39:12</created>
            <changedBy>Chris</changedBy>
            <changed>1998-10-31T23:39:12</changed>
            <description>
               <p>

            Exterior footage of

            <em>Baley Park Hotel</em>

             on fire with natural sound. Trucks are visible for the first portion of the clip.

            <em>CG locator at 0:04 and duration 0:05, Baley Park Hotel.</em>
               </p>
               <p>
                  <tab/>

            Cuts to view of fire personnel exiting hotel lobby and cleaning up after the fire is
  out.

         </p>
               <p>
                  <em>Clip has been doubled for pad on voice over.</em>
               </p>
            </description>
            <mosExternalMetadata>
               <mosScope>STORY</mosScope>
               <mosSchema>http://MOSA4.com/mos/supported_schemas/MOSAXML2.08</mosSchema>
               <mosPayload>
                  <Owner>SHOLMES</Owner>
                  <ModTime>20010308142001</ModTime>
                  <mediaTime>0</mediaTime>
                  <TextTime>278</TextTime>
                  <ModBy>LJOHNSTON</ModBy>
                  <Approved>0</Approved>
                  <Creator>SHOLMES</Creator>
               </mosPayload>
            </mosExternalMetadata>
         </mosObj>
         <mosObj>
            <objID>M000122</objID>
            <objSlug>Another Hotel Fire</objSlug>
            <mosAbstract>
               <b>Hotel Fire</b>
               <em>vo</em>

          :30

      </mosAbstract>
            <objGroup>Show 7</objGroup>
            <objType>VIDEO</objType>
            <objTB>60</objTB>
            <objRev>1</objRev>
            <objDur>1800</objDur>
            <status>NEW</status>
            <objAir>READY</objAir>
            <createdBy>Chris</createdBy>
            <created>1998-10-31T23:39:12</created>
            <changedBy>Chris</changedBy>
            <changed>1998-10-31T23:39:12</changed>
            <description>
               <p>

            Exterior footage of

            <em>Baley Park Hotel</em>

             on fire with natural sound. Trucks are visible for the first portion of the clip.

            <em>CG locator at 0:04 and duration 0:05, Baley Park Hotel.</em>
               </p>
               <p>
                  <tab/>

            Cuts to view of fire personnel exiting hotel lobby and cleaning up after the fire is
  out.
```

```
         </p>
               <p>
                  <em>Clip has been doubled for pad on voice over.</em>
               </p>
            </description>
            <mosExternalMetadata>
               <mosScope>STORY</mosScope>
               <mosSchema>http://MOSA4.com/mos/supported_schemas/MOSAXML2.08</mosSchema>
               <mosPayload>
                  <Owner>SHOLMES</Owner>
                  <ModTime>20010308142001</ModTime>
                  <mediaTime>0</mediaTime>
                  <TextTime>278</TextTime>
                  <ModBy>LJOHNSTON</ModBy>
                  <Approved>0</Approved>
                  <Creator>SHOLMES</Creator>
               </mosPayload>
            </mosExternalMetadata>
         </mosObj>
         <mosObj>
            <objID>M000123</objID>
            <objSlug>Yet Another Hotel Fire</objSlug>
            <mosAbstract>
               <b>Hotel Fire</b>
               <em>vo</em>

          :30

      </mosAbstract>
            <objGroup>Show 7</objGroup>
            <objType>VIDEO</objType>
            <objTB>60</objTB>
            <objRev>1</objRev>
            <objDur>1800</objDur>
            <status>NEW</status>
            <objAir>READY</objAir>
            <createdBy>Chris</createdBy>
            <created>1998-10-31T23:39:12</created>
            <changedBy>Chris</changedBy>
            <changed>1998-10-31T23:39:12</changed>
            <description>
               <p>

            Exterior footage of

            <em>Baley Park Hotel</em>

             on fire with natural sound. Trucks are visible for the first portion of the clip.

            <em>CG locator at 0:04 and duration 0:05, Baley Park Hotel.</em>
               </p>
               <p>
                  <tab/>

            Cuts to view of fire personnel exiting hotel lobby and cleaning up after the fire is
  out.

         </p>
               <p>
                  <em>Clip has been doubled for pad on voice over.</em>
               </p>
            </description>
            <mosExternalMetadata>
               <mosScope>STORY</mosScope>
               <mosSchema>http://MOSA4.com/mos/supported_schemas/MOSAXML2.08</mosSchema>
               <mosPayload>
                  <Owner>SHOLMES</Owner>
                  <ModTime>20010308142001</ModTime>
                  <mediaTime>0</mediaTime>
                  <TextTime>278</TextTime>
                  <ModBy>LJOHNSTON</ModBy>
                  <Approved>0</Approved>
                  <Creator>SHOLMES</Creator>
               </mosPayload>
            </mosExternalMetadata>
         </mosObj>
      </list>
   </mosObjList>
</mos>
```

3.3 Object and Item management

3.3.1 mosObjCreate – MOS Object Create

Purpose

Allows an NCS to request the Media Object Server to create a Media Object with specific metadata associated with it.

Response

mosAck (if object can be created then status description = objID)
NACK (if object CANNOT be created, status description = reason for error)
mosObj

Port

MOS Lower Port (10540) – MOS Object

Structural Outline

mos
mosID
ncsID
messageID
mosObjCreate
objSlug
objGroup?
objType
objTB
objDur?
time?
createdBy?
description?
mosExternalMetadata*

Syntax

```
<!ELEMENT mosObjCreate (objSlug, objGroup?, objType, objTB, objDur?, time?, createdBy?,
 description?, mosExternalMetadata*)>
<!ELEMENT description (#PCDATA | p | em | tab)*>
<!ELEMENT p (#PCDATA | em | tab)*>
```

Example

```
<mos>
   <mosID>videoserver.station.com</mosID>
   <ncsID>ncs.station.com</ncsID>
   <messageID>724</messageID>
   <mosObjCreate>
```

```
      <objSlug>Hotel Fire</objSlug>
      <objGroup>Show 7</objGroup>
      <objType>VIDEO</objType>
      <objTB>60</objTB>
      <objDur>1800</objDur>
      <createdBy>Chris</createdBy>
      <description>
         <p>
            Exterior footage of
            <em>Baley Park Hotel</em>
             on fire with natural sound. Trucks are
        visible for the first portion of the clip.
            <em>CG locator at 0:04 and duration 0:05, Baley Park
        Hotel.</em>
         </p>
         <p>
            <tab/>
            Cuts to view of fire personnel exiting hotel lobby and cleaning up after the fire is
 out.
         </p>
         <p>
            <em>Clip has been doubled for pad on voice over.</em>
         </p>
      </description>
      <mosExternalMetadata>
         <mosScope>STORY</mosScope>
         <mosSchema>http://NCSA4.com/mos/supported_schemas/NCSAXML2.08</mosSchema>
         <mosPayload>
            <Owner>SHOLMES</Owner>
            <ModTime>20010308142001</ModTime>
            <mediaTime>0</mediaTime>
            <TextTime>278</TextTime>
            <ModBy>LJOHNSTON</ModBy>
            <Approved>0</Approved>
            <Creator>SHOLMES</Creator>
         </mosPayload>
      </mosExternalMetadata>
   </mosObjCreate>
</mos>
```

3.3.2 mosItemReplace – Replace one Item Reference with another

Purpose

This message allows a media Object Server to replace an Item Reference in a Story with new metadata values and/or additional tags. The Story must be in a MOS Active Play List. Thus, this message is in the "ro" family of messages rather than the "mos," or lower port, family. However, this message is initiated by the media Object Server, rather than the NCS.

Behavior

This message must reference an existing unique Item Reference in a MOS Active Play List through the values of ncsID, roID, storyID, and itemID.

If metadata tags in the mosItemReplace message already exist in the target Item Reference, values within the Item Reference will be replaced by the values in the mosItemReplace message.

If the metadata tags do not already exist in the target Item Reference they will be added.

If a mosExternalMetadata block is included in the mosItemReplace message, it will replace an existing mosExternalMetadata block **only** if the values of <mosSchema> in the two blocks match, and only for the first occurrence of a block with a matching <mosSchema> tag. Otherwise the mosExternalMetadata block will be added to the target Item Reference.

If the ItemID in the mosItemReplace message does not match an existing ItemID in the specified Story then no action will be taken and the mosItemReplace message will be replied to with an roAck message specifying the Item values in the mosItemReplace message and carrying a status value of "NACK."

Response

roAck
roStoryReplace
roStorySend

Subsequent messages

roStoryReplace – A successful mosItemReplace operation will result in a change to an Item reference embedded in a Story. This new information must now be placed in associated MOS Playlists. The roStoryReplace message will replace the old item reference in the playlist with the newly updated item reference from this Story.

roStorySend – A successful mosItemReplace operation will result in a change in the body of a Story. This change must be sent back out if an roStorySend target has been defined for the RO.

Port

MOS Upper Port (10541) - Running Order

Structural Outline

mos
  mosID
   ncsID
   messageID
  mosItemReplace
      roID
      storyID
          item*
          itemID
          itemSlug?
          objID
         mosID
         mosAbstract?
          itemChannel?
          itemEdStart?
          itemEdDur?
        itemUserTimingDur?
         itemTrigger?
          macroIn?
          macroOut?
         mosExternalMetadata*

Syntax

```
<!ELEMENT mosItemReplace (roID, storyID, item)>
<!ELEMENT item (itemID, itemSlug?, objID, mosID, mosAbstract?, itemChannel?, itemEdStart?,
 itemEdDur?, itemUserTimingDur?, itemTrigger?, macroIn?, macroOut?, mosExternalMetadata*>
<!ELEMENT mosExternalMetadata (mosScope?, mosSchema, mosPayload)>
```

Example

```
<mos>
   <mosID>aircache.newscenter.com</mosID>
   <ncsID>ncs.newscenter.com</ncsID>
   <messageID>9088</messageID>
   <mosItemReplace>
      <roID>5PM</roID>
      <storyID>HOTEL FIRE</storyID>
      <item>
         <itemID>30848</itemID>
         <objID>M000627</objID>
         <mosID>testmos.enps.com</mosID>
         <itemEdStart>0</itemEdStart>
         <itemEdDur>815</itemEdDur>
         <itemUserTimingDur>310</itemUserTimingDur>
         <macroIn>c01/l04/dve07</macroIn>
         <macroOut>r00</macroOut>
         <mosExternalMetadata>
            <mosScope>PLAYLIST</mosScope>
            <mosSchema>HTTP://VVENDOR/MOS/supportedSchemas/vvend280</mosSchema>
            <mosPayload>
               <trigger>837</trigger>
               <key>110</key>
               <fade>17</fade>
               <efxTime>15</efxTime>
            </mosPayload>
         </mosExternalMetadata>
      </item>
   </mosItemReplace>
</mos>
```

ro (Running Order) family of messages

3.4 ro Playlist Construction

3.4.1 roAck - Acknowledge Running Order

Purpose

MOS response to receipt of any Running Order command. The response may contain the status for one or more Items in a Running Order. This is useful when the roAck is in response to a roCreate or roReplace command.

Response

None

Port

MOS Upper Port (10541) - Running Order

Structural Outline

mos
mosID
ncsID
messageID
roAck
roID
roStatus
(storyID, itemID, objID, status)*

Syntax

```
<!ELEMENT roAck (roID, roStatus, (storyID, itemID, objID, status)*)>
```

Example

```
<mos>
   <mosID>aircache.newscenter.com</mosID>
   <ncsID>ncs.newscenter.com</ncsID>
   <messageID>23569</messageID>
   <roAck>
      <roID>96857485</roID>
      <roStatus>Unknown object M000133</roStatus>
      <storyID>5983A501:0049B924:8390EF2B</storyID>
      <itemID>0</itemID>
      <objID>M000224</objID>
      <status>LOADED</status>
      <storyID>3854737F:0003A34D:983A0B28</storyID>
      <itemID>0</itemID>
      <objID>M000133</objID>
      <status>UNKNOWN</status>
   </roAck>
</mos>
```

3.4.2 roCreate - Create Running Order

Purpose

Message from the NCS to the MOS that defines a new Running Order.

Response

roAck

Port

MOS Upper Port (10541) - Running Order

Structural Outline

mos
mosID
ncsID
messageID

roCreate
roID
roSlug
roChannel?
roEdStart?
roEdDur?
roTrigger?
macroIn?
macroOut?
mosExternalMetadata*
story*
storyID
storySlug?
storyNum?
mosExternalMetadata*
item*
itemID
itemSlug?
objID
mosID
mosAbstract?
itemChannel?
itemEdStart?
itemEdDur?
itemUserTimingDur?
itemTrigger?
macroIn?
macroOut?
mosExternalMetadata*

Syntax

```
<!ELEMENT roCreate (roID, roSlug, roChannel?, roEdStart?, roEdDur?, roTrigger?, macroIn?,
 macroOut?, mosExternalMetadata*, story*)>
<!ELEMENT story (storyID, storySlug?, storyNum?, mosExternalMetadata*,  item*)>
<!ELEMENT item (itemID, itemSlug?, objID, mosID, mosAbstract?, itemChannel?, itemEdStart?,
 itemEdDur?, itemUserTimingDur?, itemTrigger?, macroIn?, macroOut?, mosExternalMetadata*)>
```

Example

```
<mos>
   <mosID>aircache.newscenter.com</mosID>
   <ncsID>ncs.newscenter.com</ncsID>
   <messageID>30334</messageID>
   <roCreate>
      <roID>96857485</roID>
      <roSlug>5PM RUNDOWN</roSlug>
      <roEdStart>1999-04-17T17:02:00</roEdStart>
      <roEdDur>00:58:25</roEdDur>
      <story>
         <storyID>5983A501:0049B924:8390EF2B</storyID>
         <storySlug>COLSTAT MURDER</storySlug>
         <storyNum>A5</storyNum>
         <item>
            <itemID>0</itemID>
            <itemSlug>COLSTAT MURDER:VO</itemSlug>
            <objID>M000224</objID>
            <mosID>testmos.enps.com</mosID>
            <itemEdDur>645</itemEdDur>
            <itemUserTimingDur>310</itemUserTimingDur>
```

```
            <itemTrigger>CHAINED</itemTrigger>
            <mosExternalMetadata>
               <mosScope>PLAYLIST</mosScope>
               <mosSchema>http://MOSA4.com/mos/supported_schemas/MOSAXML2.08</mosSchema>
               <mosPayload>
                  <Owner>SHOLMES</Owner>
                  <transitionMode>2</transitionMode>
                  <transitionPoint>463</transitionPoint>
                  <source>a</source>
                  <destination>b</destination>
               </mosPayload>
            </mosExternalMetadata>
         </item>
      </story>
      <story>
         <storyID>3854737F:0003A34D:983A0B28</storyID>
         <storySlug>AIRLINE INSPECTIONS</storySlug>
         <storyNum>A6</storyNum>
         <item>
            <itemID>0</itemID>
            <objID>M000133</objID>
            <mosID>testmos.enps.com</mosID>
            <itemEdStart>55</itemEdStart>
            <itemEdDur>310</itemEdDur>
             <itemUserTimingDur>200</itemUserTimingDur>
            <mosExternalMetadata>
               <mosScope>PLAYLIST</mosScope>
               <mosSchema>http://MOSA4.com/mos/supported_schemas/MOSAXML2.08</mosSchema>
               <mosPayload>
                  <Owner>SHOLMES</Owner>
                  <transitionMode>2</transitionMode>
                  <transitionPoint>463</transitionPoint>
                  <source>a</source>
                  <destination>b</destination>
               </mosPayload>
            </mosExternalMetadata>
         </item>
      </story>
   </roCreate>
</mos>
```

3.4.3 roReplace - Replace Running Order

Purpose

Replaces an existing Running Order definition in the MOS with another one sent from the NCS.

Response

roAck

Port

MOS Upper Port (10541) - Running Order

Structural Outline

mos
- mosID
- ncsID
- messageID
- roReplace
  - roID

roSlug
roChannel?
roEdStart?
roEdDur?
roTrigger?
macroIn?
macroOut?
mosExternalMetadata*
story*
storyID
storySlug?
storyNum?
mosExternalMetadata*
item*
itemID
itemSlug?
objID
mosID
mosAbstract?
itemChannel?
itemEdStart?
itemEdDur?
itemUserTimingDur?
itemTrigger?
macroIn?
macroOut?
mosExternalMetadata*

Syntax

```
<!ELEMENT roReplace (roID, roSlug, roChannel?, roEdStart?, roEdDur?, roTrigger?, macroIn?,
 macroOut?, mosExternalMetadata*, story*)>
<!ELEMENT story (storyID, storySlug?, storyNum?, mosExternalMetadata*,  item*)>
<!ELEMENT item (itemID, itemSlug?, objID, mosID, mosAbstract?, itemChannel?, itemEdStart?,
 itemEdDur?, itemUserTimingDur?, itemTrigger?, macroIn?, macroOut?, mosExternalMetadata*)>
```

Example

```
<mos>
   <mosID>aircache.newscenter.com</mosID>
   <ncsID>ncs.newscenter.com</ncsID>
   <messageID>2345</messageID>
   <roReplace>
      <roID>96857485</roID>
      <roSlug>5PM RUNDOWN</roSlug>
      <story>
         <storyID>5983A501:0049B924:8390EF2B</storyID>
         <storySlug>COLSTAT MURDER</storySlug>
         <storyNum>A1</storyNum>
         <item>
            <itemID>0</itemID>
            <itemSlug>COLSTAT MURDER:VO</itemSlug>
            <objID>M000224</objID>
            <mosID>testmos.enps.com</mosID>
            <itemEdDur>645</itemEdDur>
            <itemUserTimingDur>310</itemUserTimingDur>
            <itemTrigger>CHAINED</itemTrigger>
            <mosExternalMetadata>
               <mosScope>PLAYLIST</mosScope>
               <mosSchema>http://MOSA4.com/mos/supported_schemas/MOSAXML2.08</mosSchema>
               <mosPayload>
```

```
                  <Owner>SHOLMES</Owner>
                  <transitionMode>2</transitionMode>
                  <transitionPoint>463</transitionPoint>
                  <source>a</source>
                  <destination>b</destination>
               </mosPayload>
            </mosExternalMetadata>
         </item>
      </story>
      <story>
         <storyID>3852737F:0013A64D:923A0B28</storyID>
         <storySlug>AIRLINE SAFETY</storySlug>
         <storyNum>A2</storyNum>
         <item>
            <itemID>0</itemID>
            <objID>M000295</objID>
            <mosID>testmos.enps.com</mosID>
            <itemEdStart>500</itemEdStart>
            <itemEdDur>600</itemEdDur>
            <itemUserTimingDur>310</itemUserTimingDur>
            <mosExternalMetadata>
               <mosScope>PLAYLIST</mosScope>
               <mosSchema>http://MOSA4.com/mos/supported_schemas/MOSAXML2.08</mosSchema>
               <mosPayload>
                  <Owner>SHOLMES</Owner>
                  <transitionMode>2</transitionMode>
                  <transitionPoint>463</transitionPoint>
                  <source>a</source>
                  <destination>b</destination>
               </mosPayload>
            </mosExternalMetadata>
         </item>
      </story>
   </roReplace>
</mos>
```

3.4.4 roMetadataReplace – Replace RO metadata without deleting the RO structure

Purpose

This message allows metadata associated with a running order to be replaced without deleting the running order and sending the entire running order again.

Behavior

This message must reference an existing running order

If metadata tags in the roMetadataReplace message already exist in the target RO, values within the RO will be replaced by the values in the roMetadataReplace message.

If the metadata tags do not already exist in the target RO they will be added.

If a mosExternalMetadata block is included in the roMetadataReplace message, it will replace an existing mosExternalMetadata block **only** if the values of mosSchema in the two blocks match. Otherwise the mosExternalMetadata block will be added to the target RO.

If the ROID in the roMetadataReplace message does not match an existing ROID then no action will be taken and the roMetadataReplace message will be replied to with an roAck message which carrying a status value of "NACK."

Response

roAck

Port

MOS Upper Port (10541) - Running Order

Structural Outline

mos
- mosID
- ncsID
- messageID
- roMetadataReplace
  - roID
  - roSlug
  - roChannel?
  - roEdStart?
  - roEdDur?
  - roTrigger?
  - roMacroIn?
  - roMacroOut?
  - mosExternalMetadata?

Syntax

```
<!ELEMENT roMetadataReplace (roID, roSlug, roChannel?, roEdStart?, roEdDur?, roTrigger?,
 roMacroIn?, roMacroOut?, mosExternalMetadata?)>
```

Example

```
<mos>
   <mosID>aircache.newscenter.com</mosID>
   <ncsID>ncs.newscenter.com</ncsID>
   <messageID>654033</messageID>
   <roMetadataReplace>
      <roID>96857485</roID>
      <roSlug>5PM RUNDOWN</roSlug>
      <roEdStart>1999-04-17T17:02:00</roEdStart>
      <roEdDur>00:58:25</roEdDur>
      <mosExternalMetadata>
         <mosScope>PLAYLIST</mosScope>
<mosSchema>http://MOSA4.com/mos/supported_schemas/MOSAXML2.08</mosSchema>
         <mosPayload>
            <Owner>SHOLMES</Owner>
            <transitionMode>2</transitionMode>
            <transitionPoint>463</transitionPoint>
            <source>a</source>
            <destination>b</destination>
         </mosPayload>
      </mosExternalMetadata>
   </roMetadataReplace>
</mos>
```

3.4.5 roDelete - Delete Running Order

Purpose

Deletes a Running order in the MOS.

Response

roAck

Port

MOS Upper Port (10541) - Running Order

Structural Outline

mos
- mosID
- ncsID
- messageID
- roDelete
  - roID

Syntax

```
<!ELEMENT roDelete (roID)>
```

Example

```
<mos>
   <mosID>aircache.newscenter.com</mosID>
   <ncsID>ncs.newscenter.com</ncsID>
   <messageID>544</messageID>
   <roDelete>
      <roID>49478285</roID>
   </roDelete>
</mos>
```

3.5 ro Synchronization, Discovery and Status

3.5.1 roReq - Request Running Order

Purpose

Request for a complete build of a Running Order Playlist.
NOTE: This message can be used by either NCS or MOS.
A MOS can use this to "resync" its Playlist with the NCS Running Order or to obtain a full description of the Playlist at any time.
An NCS can use this as a diagnostic tool to check the order of the Playlist constructed in the MOS versus the sequence of Items in the Running Order.

Response

roList or roAck (roAck with a NACK value is sent if the Running Order ID is not valid or roList cannot be returned for some reason)

Port

MOS Upper Port (10541) - Running Order

Structural Outline

mos
- mosID
- ncsID
- messageID
- roReq
  - roID

Syntax

```
<!ELEMENT roReq (roID)>
```

Example

```
<mos>
   <mosID>aircache.newscenter.com</mosID>
   <ncsID>ncs.newscenter.com</ncsID>
   <messageID>30761</messageID>
   <roReq>
      <roID>96857485</roID>
   </roReq>
</mos>
```

3.5.2 roList - List Running Order

Purpose

A complete build or rebuild of a Running Order Playlist in response to an roReq message.
NOTE: This message can be sent by either the NCS or MOS
A MOS can use this to "resync" its Playlist with the NCS Running Order or to obtain a full description of the Playlist at any time.
An NCS can use this as a diagnostic tool to check the order of the Playlist constructed in the MOS versus the sequence of Items in the Running Order.
roList is functionally similar to roCreate.

Response

None

Port

MOS Upper Port (10541) - Running Order

Structural Outline

mos
- mosID
- ncsID
- messageID
- roList
  - roID
  - roSlug
  - roChannel?
  - roEdStart?
  - roEdDur?
  - roTrigger?
  - macroIn?
  - macroOut?
  - mosExternalMetadata*
  - story*
    - storyID
    - storySlug?
    - storyNum?
    - mosExternalMetadata*
    - item*
      - itemID
      - itemSlug?
      - objID
      - mosID
      - mosAbstract?
      - itemChannel?
      - itemEdStart?
      - itemEdDur?
      - itemUserTimingDur?
      - itemTrigger?
      - macroIn?
      - macroOut?
      - mosExternalMetadata*

Syntax

```
<!ELEMENT roList (roID, roSlug, roChannel?, roEdStart?, roEdDur?, roTrigger?, macroIn?, macroOut?,
 mosExternalMetadata*, story*)>
<!ELEMENT story (storyID, storySlug?, storyNum?, mosExternalMetadata*,  item*)>
<!ELEMENT item (itemID, itemSlug?, objID, mosID, mosAbstract?, itemChannel?, itemEdStart?,
 itemEdDur?, itemUserTimingDur?, itemTrigger?, macroIn?, macroOut?, mosExternalMetadata*)>
```

Example

```
<mos>
   <mosID>aircache.newscenter.com</mosID>
   <ncsID>ncs.newscenter.com</ncsID>
   <messageID>111</messageID>
   <roList>
      <roID>96857485</roID>
```

```
      <roSlug>5PM RUNDOWN</roSlug>
      <story>
         <storyID>5983A501:0049B924:8390EF2B</storyID>
         <storySlug>Colstat Murder</storySlug>
         <storyNum>B10</storyNum>
         <item>
            <itemID>0</itemID>
            <itemSlug>COLSTAT MURDER:VO</itemSlug>
            <objID>M000224</objID>
            <mosID>testmos.enps.com</mosID>
            <itemEdDur>645</itemEdDur>
             <itemUserTimingDur>310</itemUserTimingDur>
            <itemTrigger>CHAINED</itemTrigger>
            <mosExternalMetadata>
               <mosScope>PLAYLIST</mosScope>
               <mosSchema>http://MOSA4.com/mos/supported_schemas/MOSAXML2.08</mosSchema>
               <mosPayload>
                  <Owner>SHOLMES</Owner>
                  <transitionMode>2</transitionMode>
                  <transitionPoint>463</transitionPoint>
                  <source>a</source>
                  <destination>b</destination>
               </mosPayload>
            </mosExternalMetadata>
         </item>
      </story>
      <story>
         <storyID>3854737F:0003A34D:983A0B28</storyID>
         <storySlug>Test MOS</storySlug>
         <storyNum>B11</storyNum>
         <item>
            <itemID>0</itemID>
            <objID>M000133</objID>
            <mosID>testmos.enps.com</mosID>
            <itemEdStart>55</itemEdStart>
            <itemEdDur>310</itemEdDur>
             <itemUserTimingDur>310</itemUserTimingDur>
            <mosExternalMetadata>
               <mosScope>PLAYLIST</mosScope>
               <mosSchema>http://MOSA4.com/mos/supported_schemas/MOSAXML2.08</mosSchema>
               <mosPayload>
                  <Owner>SHOLMES</Owner>
                  <transitionMode>2</transitionMode>
                  <transitionPoint>463</transitionPoint>
                  <source>a</source>
                  <destination>b</destination>
               </mosPayload>
            </mosExternalMetadata>
         </item>
      </story>
   </roList>
</mos>
```

3.5.3 roReqAll - Request All Running Order Descriptions

Purpose

Request for a description of all Running Orders known by a NCS from a MOS.

Response

roListAll

Port

MOS Upper Port (10541) - Running Order

Structural Outline

mos
- mosID
- ncsID
- messageID
- roReqAll

Syntax

```
<!ELEMENT roReqAll EMPTY>
```

Example

```
<mos>
   <mosID>aircache.newscenter.com</mosID>
   <ncsID>ncs.newscenter.com</ncsID>
   <messageID>60543</messageID>
   <roReqAll/>
</mos>
```

3.5.4 roListAll - List All Running Order Descriptions

Purpose

Provides a description of all Running Orders known by a NCS to a MOS.

Response

None

Port

MOS Upper Port (10541) - Running Order

Structural Outline

mos
- mosID
- ncsID
- messageID
- roListAll
  - ro*
    - roID
    - roSlug?
    - roChannel?
    - roEdStart?
    - roEdDur?
    - roTrigger?
    - mosExternalMetadata*

Syntax

```
<!ELEMENT roListAll (ro*)>
<!ELEMENT ro (roID, roSlug?, roChannel?, roEdStart?, roEdDur?, roTrigger?, mosExternalMetadata*)>
```

Example

```
<mos>
 <mosID>aircache.newscenter.com</mosID>
 <ncsID>ncs.newscenter.com</ncsID>
   <messageID>300045</messageID>
   <roListAll>
     <ro>
       <roID>5PM</roID>
       <roSlug>5PM Rundown</roSlug>
       <roChannel></roChannel>
       <roEdStart>2003-07-11T17:00:00</roEdStart>
       <roEdDur>00:30:00</roEdDur>
       <roTrigger>MANUAL</roTrigger>
       <mosExternalMetadata>
         <mosScope>PLAYLIST</mosScope>
         <mosSchema>http://ncsA4.com/mos/supported_schemas/NCSAXML2.08</mosSchema>
         <mosPayload>
            <Owner>SHOLMES</Owner>
            <mediaTime>0</mediaTime>
            <TextTime>278</TextTime>
            <ModBy>LJOHNSTON</ModBy>
            <Approved>0</Approved>
            <Creator>SHOLMES</Creator>
         </mosPayload>
      </mosExternalMetadata>
     </ro>
     <ro>
       <roID>6PM</roID>
       <roSlug>6PM Rundown</roSlug>
       <roChannel></roChannel>
       <roEdStart>2003-07-09T18:00:00</roEdStart>
       <roEdDur>00:30:00</roEdDur>
       <roTrigger>MANUAL</roTrigger>
       <mosExternalMetadata>
         <mosScope>PLAYLIST</mosScope>
         <mosSchema>http://ncsA4.com/mos/supported_schemas/NCSAXML2.08</mosSchema>
         <mosPayload>
            <Owner>SHOLMES</Owner>
            <mediaTime>0</mediaTime>
            <TextTime>350</TextTime>
            <ModBy>BSMITH</ModBy>
            <Approved>1</Approved>
            <Creator>SHOLMES</Creator>
         </mosPayload>
      </mosExternalMetadata>
      </ro>
   </roListAll>
</mos>
```

3.5.5 roStat - Status of a MOS Running Order

Purpose

Method for the MOS to update the NCS on the status of Play List. This allows the NCS to reflect the status of the Play List in the NRC Running Order Display.

Response

roAck

Port

MOS Upper Port (10541) - Running Order

Structural Outline

mos
- mosID
- ncsID
- messageID
- roStat
  - roID
  - status
  - time

Syntax

```
<!ELEMENT roStat (roID, status, time)>
```

Example

```
<mos>
   <mosID>aircache.newscenter.com</mosID>
   <ncsID>ncs.newscenter.com</ncsID>
   <messageID>27</messageID>
   <roStat>
      <roID>5PM</roID>
      <status>MANUAL CTRL</status>
      <time>1999-04-11T14:22:07</time>
   </roStat>
</mos>
```

3.5.6 roReadyToAir - Identify a Running Order as Ready to Air

Purpose

The message allows the NCS to signal the MOS that a Running Order has been editorially approved ready for air.

Response

roAck

Port

MOS Upper Port (10541) - Running Order

Structural Outline

mos
- mosID
- ncsID
- messageID
- roReadyToAir
  - roID
  - roAir

Syntax

```
<!ELEMENT roReadyToAir (roID, roAir)>
```

Example

```
<mos>
   <mosID>aircache.newscenter.com</mosID>
   <ncsID>ncs.newscenter.com</ncsID>
   <messageID>30467</messageID>
   <roReadyToAir>
      <roID>5PM</roID>
      <roAir>READY</roAir>
   </roReadyToAir>
</mos>
```

3.6 ro Story and Item Sequence Modification

3.6.1 roStoryAppend - Append Stories to Running Order

Purpose

Appends stories and all of its defined items at the end of a running order.
Note: The NCS should use the equivalent roElementAction message to achieve the same result.

Response

roAck

Port

MOS Upper Port (10541) - Running Order

Structural Outline

mos
- mosID
- ncsID
- messageID
- roStoryAppend
  - roID
  - story+

storyID
storySlug?
storyNum?
mosExternalMetadata*
item*
itemID
itemSlug?
objID
mosID
mosAbstract?
itemChannel?
itemEdStart?
itemEdDur?
itemUserTimingDur?
itemTrigger?
macroIn?
macroOut?
mosExternalMetadata*

Syntax

```
<!ELEMENT roStoryAppend (roID, story+)>
<!ELEMENT story (storyID, storySlug?, storyNum?, mosExternalMetadata*, item*)>
<!ELEMENT item (itemID, itemSlug?, objID, mosID, mosAbstract?, itemChannel?, itemEdStart?,
 itemEdDur?, itemUserTimingDur?, itemTrigger?, macroIn?, macroOut?, mosExternalMetadata*)>
```

Example

```
<mos>
   <mosID>aircache.newscenter.com</mosID>
   <ncsID>ncs.newscenter.com</ncsID>
   <messageID>12055</messageID>
   <roStoryAppend>
      <roID>5PM</roID>
      <story>
         <storyID>V: BRIDGE COLLAPSE</storyID>
         <storySlug>Bridge Collapse</storySlug>
         <storyNum>B7</storyNum>
         <item>
            <itemID>30848</itemID>
            <objID>M000627</objID>
            <mosID>testmos.enps.com</mosID>
            <itemEdStart>0</itemEdStart>
            <itemEdDur>815</itemEdDur>
            <itemUserTimingDur>310</itemUserTimingDur>
            <macroIn>c01/l04/dve07</macroIn>
            <macroOut>r00</macroOut>
            <mosExternalMetadata>
               <mosScope>PLAYLIST</mosScope>
               <mosSchema>http://MOSA4.com/mos/supported_schemas/MOSAXML2.08</mosSchema>
               <mosPayload>
                  <Owner>SHOLMES</Owner>
                  <transitionMode>2</transitionMode>
                  <transitionPoint>463</transitionPoint>
                  <source>a</source>
                  <destination>b</destination>
               </mosPayload>
            </mosExternalMetadata>
            <mosExternalMetadata>
               <mosScope>PLAYLIST</mosScope>
               <mosSchema>http://MOSA4.com/mos/supported_schemas/MOSBXML2.08</mosSchema>
               <mosPayload>
                  <rate>52</rate>
                  <background>2</background>
```

```
                  <overlay>463</overlay>
               </mosPayload>
            </mosExternalMetadata>
         </item>
         <item>
            <itemID>30849</itemID>
            <objID>M000628</objID>
            <mosID>testmos</mosID>
            <itemEdStart>0</itemEdStart>
            <itemEdDur>815</itemEdDur>
            <itemUserTimingDur>310</itemUserTimingDur>
            <macroIn>c01/l04/dve07</macroIn>
            <macroOut>r00</macroOut>
            <mosExternalMetadata>
               <mosScope>PLAYLIST</mosScope>
               <mosSchema>http://MOSA4.com/mos/supported_schemas/MOSAXML2.08</mosSchema>
               <mosPayload>
                  <Owner>SHOLMES</Owner>
                  <transitionMode>2</transitionMode>
                  <transitionPoint>463</transitionPoint>
                  <source>a</source>
                  <destination>b</destination>
               </mosPayload>
            </mosExternalMetadata>
         </item>
      </story>
   </roStoryAppend>
</mos>
```

3.6.2 roStoryInsert - Insert Stories in Running Order

Purpose

Inserts stories and all of its defined items before the referenced Story in the running order.

Note: The NCS should use the equivalent roElementAction message to achieve the same result.

Response

roAck

Port

MOS Upper Port (10541) - Running Order

Structural Outline

mos
- mosID
- ncsID
- messageID
- roStoryInsert
  - roID
  - storyID
  - story+
    - storyID
    - storySlug?
    - storyNum?

mosExternalMetadata*
item*
itemID
itemSlug?
objID
mosID
mosAbstract?
itemChannel?
itemEdStart?
itemEdDur?
itemUserTimingDur?
itemTrigger?
macroIn?
macroOut?
mosExternalMetadata*

Syntax

```
<!ELEMENT roStoryInsert (roID, storyID, story+)>
<!ELEMENT story (storyID, storySlug?, storyNum?, mosExternalMetadata*, item*)>
<!ELEMENT item (itemID, itemSlug?, objID, mosID, mosAbstract?, itemChannel?, itemEdStart?,
 itemEdDur?, itemUserTimingDur?, itemTrigger?, macroIn?, macroOut?, mosExternalMetadata*)>
```

Example

```
<mos>
   <mosID>aircache.newscenter.com</mosID>
   <ncsID>ncs.newscenter.com</ncsID>
   <messageID>2000340</messageID>
   <roStoryInsert>
      <roID>5PM</roID>
      <storyID>HOTEL FIRE</storyID>
      <story>
         <storyID>V: BRIDGE COLLAPSE</storyID>
         <storySlug>Bridge Collapse</storySlug>
         <storyNum>B7</storyNum>
         <item>
            <itemID>30848</itemID>
            <objID>M000627</objID>
            <mosID>testmos.enps.com</mosID>
            <itemEdStart>0</itemEdStart>
            <itemEdDur>815</itemEdDur>
            <itemUserTimingDur>310</itemUserTimingDur>
            <macroIn>c01/l04/dve07</macroIn>
            <macroOut>r00</macroOut>
            <mosExternalMetadata>
               <mosScope>PLAYLIST</mosScope>
               <mosSchema>http://MOSA4.com/mos/supported_schemas/MOSAXML2.08</mosSchema>
               <mosPayload>
                  <Owner>SHOLMES</Owner>
                  <transitionMode>2</transitionMode>
                  <transitionPoint>463</transitionPoint>
                  <source>a</source>
                  <destination>b</destination>
               </mosPayload>
            </mosExternalMetadata>
            <mosExternalMetadata>
               <mosScope>PLAYLIST</mosScope>
               <mosSchema>http://MOSA4.com/mos/supported_schemas/MOSBXML2.08</mosSchema>
               <mosPayload>
                  <rate>52</rate>
                  <background>2</background>
                  <overlay>463</overlay>
               </mosPayload>
            </mosExternalMetadata>
         </item>
```

```
         <item>
            <itemID>30849</itemID>
            <objID>M000628</objID>
            <mosID>testmos</mosID>
            <itemEdStart>0</itemEdStart>
            <itemEdDur>815</itemEdDur>
            <itemUserTimingDur>310</itemUserTimingDur>
            <macroIn>c01/l04/dve07</macroIn>
            <macroOut>r00</macroOut>
            <mosExternalMetadata>
               <mosScope>PLAYLIST</mosScope>
               <mosSchema>http://MOSA4.com/mos/supported_schemas/MOSAXML2.08</mosSchema>
               <mosPayload>
                  <Owner>SHOLMES</Owner>
                  <transitionMode>2</transitionMode>
                  <transitionPoint>463</transitionPoint>
                  <source>a</source>
                  <destination>b</destination>
               </mosPayload>
            </mosExternalMetadata>
         </item>
      </story>
   </roStoryInsert>
</mos>
```

3.6.3 roStoryReplace - Replace Story with Another in a Running Order

Purpose

Replaces the referenced story with another story or stories and all of its associated Items in the Running Order.
Note: The NCS should use the equivalent roElementAction message to achieve the same result.

Response

roAck

Port

MOS Upper Port (10541) - Running Order

Structural Outline

mos
- mosID
- ncsID
- messageID
- roStoryReplace
  - roID
  - storyID
  - story+
    - storyID
    - storySlug?
    - storyNum?
    - mosExternalMetadata*
    - item*
      - itemID

itemSlug?
objID
mosID
mosAbstract?
itemChannel?
itemEdStart?
itemEdDur?
itemUserTimingDur?
itemTrigger?
macroIn?
macroOut?
mosExternalMetadata*

Syntax

```
<!ELEMENT roStoryReplace (roID, storyID, story+)>
<!ELEMENT story (storyID, storySlug?, storyNum?, mosExternalMetadata*,  item*)>
<!ELEMENT item (itemID, itemSlug?, objID, mosID, mosAbstract?, itemChannel?, itemEdStart?,
 itemEdDur?, itemUserTimingDur?, itemTrigger?, macroIn?, macroOut?, mosExternalMetadata*)>
```

Example

```
<mos>
   <mosID>aircache.newscenter.com</mosID>
   <ncsID>ncs.newscenter.com</ncsID>
   <messageID>567</messageID>
   <roStoryReplace>
      <roID>5PM</roID>
      <storyID>P: PHILLIPS INTERVIEW</storyID>
      <story>
         <storyID>V: HOTEL FIRE</storyID>
         <storySlug>Hotel Fire</storySlug>
         <storyNum>C1</storyNum>
         <item>
            <itemID>30848</itemID>
            <itemSlug>Hotel Fire vo</itemSlug>
            <objID>M000702</objID>
            <mosID>testmos</mosID>
            <itemEdStart>0</itemEdStart>
            <itemEdDur>900</itemEdDur>
            <itemUserTimingDur>800</itemUserTimingDur>
            <macroIn>c01/l04/dve07</macroIn>
            <macroOut>r00</macroOut>
            <mosExternalMetadata>
               <mosScope>PLAYLIST</mosScope>
               <mosSchema>http://MOSA4.com/mos/supported_schemas/MOSAXML2.08</mosSchema>
               <mosPayload>
                  <Owner>SHOLMES</Owner>
                  <transitionMode>2</transitionMode>
                  <transitionPoint>463</transitionPoint>
                  <source>a</source>
                  <destination>b</destination>
               </mosPayload>
            </mosExternalMetadata>
         </item>
      </story>
      <story>
         <storyID>V: DORMITORY FIRE</storyID>
         <storySlug>Dormitory Fire</storySlug>
         <storyNum>C2</storyNum>
         <item>
            <itemID>1</itemID>
            <itemSlug>Dormitory Fire vo</itemSlug>
            <objID>M000705</objID>
            <mosID>testmos</mosID>
            <itemEdStart>0</itemEdStart>
            <itemEdDur>800</itemEdDur>
            <itemUserTimingDur>310</itemUserTimingDur>
```

```
            <macroIn>c01/l04/dve07</macroIn>
            <macroOut>r00</macroOut>
            <mosExternalMetadata>
               <mosScope>PLAYLIST</mosScope>
               <mosSchema>http://MOSA4.com/mos/supported_schemas/MOSAXML2.08</mosSchema>
               <mosPayload>
                  <Owner>SHOLMES</Owner>
                  <transitionMode>2</transitionMode>
                  <transitionPoint>463</transitionPoint>
                  <source>a</source>
                  <destination>b</destination>
               </mosPayload>
            </mosExternalMetadata>
         </item>
      </story>
   </roStoryReplace>
</mos>
```

3.6.4 roStoryMove – Move a story to a new position in the Playlist

Purpose

This message allows a story to be moved to a new location in a playlist. The first storyID is the ID of the story to be moved. The second storyID is the ID of the story above which the first story is to be moved.

Note: The NCS should use the equivalent roElementAction message to achieve the same result.

noteIf the second <storyID> tag is blank move to the bottom.

Response

roAck

Port

MOS Upper Port (10541) - Running Order

Structural Outline

mos
- mosID
- ncsID
- messageID
- roStoryMove
  - roID
  - storyID
  - storyID

Syntax

```
<!ELEMENT roStoryMove (roID, storyID, storyID)>
```

Example

```
<mos>
   <mosID>aircache.newscenter.com</mosID>
   <ncsID>ncs.newscenter.com</ncsID>
   <messageID>40411</messageID>
   <roStoryMove>
      <roID>5PM</roID>
      <storyID>V: BRIDGE COLLAPSE</storyID>
      <storyID>P: PHILLIPS INTERVIEW</storyID>
   </roStoryMove>
</mos>
```

3.6.5 roStorySwap - Swap Positions of Stories in Running Order

Purpose

Swaps the Positions of stories and all of their associated Items in the Running Order.

Note: The NCS should use the equivalent roElementAction message to achieve the same result.

Response

roAck

Port

MOS Upper Port (10541) - Running Order

Structural Outline

mos
- mosID
- ncsID
- messageID
- roStorySwap
  - roID
  - storyID
  - storyID

Syntax

```
<!ELEMENT roStorySwap (roID, storyID, storyID)>
```

Example

```
<mos>
   <mosID>aircache.newscenter.com</mosID>
   <ncsID>ncs.newscenter.com</ncsID>
   <messageID>509213</messageID>
   <roStorySwap>
      <roID>5PM</roID>
      <storyID>V: BRIDGE COLLAPSE</storyID>
      <storyID>P: PHILLIPS INTERVIEW</storyID>
   </roStorySwap>
</mos>
```

3.6.6 roStoryDelete - Delete Stories from Running Order

Purpose

Deletes the referenced Stories and all associated Items from the Running Order.

Note: The NCS should use the equivalent roElementAction message to achieve the same result.

Response

roAck

Port

MOS Upper Port (10541) - Running Order

Structural Outline

mos
- mosID
- ncsID
- messageID
- roStoryDelete
  - roID
  - storyID+

Syntax

```
<!ELEMENT roStoryDelete (roID, storyID+)>
```

Example

```
<mos>
   <mosID>aircache.newscenter.com</mosID>
   <ncsID>ncs.newscenter.com</ncsID>
   <messageID>213405</messageID>
   <roStoryDelete>
      <roID>5PM</roID>
      <storyID>V: BRIDGE COLLAPSE</storyID>
      <storyID>P: PHILLIPS INTERVIEW</storyID>
   </roStoryDelete>
</mos>
```

3.6.7 roStoryMoveMultiple – Move one or more stories to a new position in the playlist

Purpose

This command allows one or more stories to be moved to a new location in the playlist. The last storyID is the ID of the story before which to insert the new stories. All remaining storyIDs identify stories to insert at that location. The resulting playlist has all the moved stories appearing in the order specified in the command before the reference story. If the last storyID is blank, the stories are moved to the end of the

playlist.

Note: The NCS should use the equivalent roElementAction message to achieve the same result.

Validation

There may be no duplicates in the list of storyIDs. This prevents the move from being ambiguous; if two IDs are the same, it is unclear where in the playlist the story with that ID must be placed.

Response

roAck

Port

MOS Upper Port (10541) – Running Order

Structural Outline

mos
- mosID
- ncsID
- messageID
- roStoryMoveMultiple
  - roID
  - storyID+

Syntax

```
<!ELEMENT roStoryMoveMultiple (roID, storyID+ )>
```

Example

If the 5PM running order has stories in the order 1 2 3 4 5 6, the following command will change the story order to 2 3 5 6 1 4.

```
<mos>
      <mosID>aircache.newscenter.com</mosID>
      <ncsID>ncs.newscenter.com</ncsID>
      <messageID>789437</messageID>
      <roStoryMoveMultiple>
            <roID>5PM</roID>
            <storyID>2</storyID>
            <storyID>3</storyID>
            <storyID>5</storyID>
            <storyID>6</storyID>
            <storyID>1</storyID>
      </roStoryMoveMultiple>
</mos>
```

3.6.8 roItemInsert – Insert Items in Story

Purpose

This message allows one or more items to be inserted before a referenced item in a story in the playlist. The first itemID is the ID of the item before which to insert the new items. If the first itemID is blank, the items are inserted at the end of the story.

Note: The NCS should use the equivalent roElementAction message to achieve the same result.

Validation

There may be no duplicates in the list of itemIDs. The protocol requires that itemIDs must be unique within a story.

Response

roAck

Port

MOS Upper Port (10541) – Running Order

Structural Outline

mos
- mosID
- ncsID
- messageID
- roItemInsert
  - roID
  - storyID
  - itemID
  - item+
    - itemID
    - itemSlug?
    - objID
    - mosID
    - mosAbstract?
    - itemChannel?
    - itemEdStart?
    - itemEdDur?
    - itemUserTimingDur?
    - itemTrigger?
    - macroIn?
    - macroOut?
    - mosExternalMetadata*

Syntax

```
<!ELEMENT roItemInsert (roID, storyID, itemID, item+)>
<!ELEMENT item (itemID, itemSlug?, objID, mosID, mosAbstract?, itemChannel?, itemEdStart?,
 itemEdDur?, itemUserTimingDur?, itemTrigger?, macroIn?, macroOut?, mosExternalMetadata*)>
```

Example

```
<mos>
   <mosID>aircache.newscenter.com</mosID>
   <ncsID>ncs.newscenter.com</ncsID>
   <messageID>5099</messageID>
   <roItemInsert>
      <roID>5PM</roID>
      <storyID>2597609</storyID>
      <itemID>5</itemID>
      <item>
         <itemID>30848</itemID>
         <itemSlug>Hotel Fire vo</itemSlug>
         <objID>M00702</objID>
         <mosID>testmos</mosID>
         <itemEdStart>0</itemEdStart>
         <itemEdDur>900</itemEdDur>
         <itemUserTimingDur>310</itemUserTimingDur>
      </item>
      <item>
         <itemID>1</itemID>
         <itemSlug>Dormitory Fire vo</itemSlug>
         <objID>M00705</objID>
         <mosID>testmos</mosID>
         <itemEdStart>0</itemEdStart>
         <itemEdDur>800</itemEdDur>
         <itemUserTimingDur>310</itemUserTimingDur>
      </item>
   </roItemInsert>
</mos>
```

3.6.9 roItemReplace – Replace an Item with one or more Items in a Story

Purpose

Replaces the referenced item in a story with one or more items.

Note: The NCS should use the equivalent roElementAction message to achieve the same result.

Validation

There may be no duplicates in the list of items. The protocol requires this; itemIDs must be unique within a story. The referenced item, specified by the first itemID, may appear at most once in the list of items.

Response

roAck

Port

MOS Upper Port (10541) – Running Order

Structural Outline

mos
- mosID
- ncsID
- messageID
- roItemReplace
  - roID
  - storyID
  - itemID
  - item+
    - itemID
    - itemSlug?
    - objID
    - mosID
    - mosAbstract?
    - itemChannel?
    - itemEdStart?
    - itemEdDur?
    - itemUserTimingDur?
    - itemTrigger?
    - macroIn?
    - macroOut?
    - mosExternalMetadata*

Syntax

```
<!ELEMENT roItemReplace (roID, storyID, itemID, item+)>
<!ELEMENT item (itemID, itemSlug?, objID, mosID, mosAbstract?, itemChannel?, itemEdStart?,
 itemEdDur?, itemUserTimingDur?, itemTrigger?, macroIn?, macroOut?, mosExternalMetadata*)>
```

Example

```
<mos>
   <mosID>aircache.newscenter.com</mosID>
   <ncsID>ncs.newscenter.com</ncsID>
   <messageID>6528</messageID>
   <roItemReplace>
      <roID>5PM</roID>
      <storyID>2597609</storyID>
      <itemID>5</itemID>
      <item>
         <itemID>30848</itemID>
         <itemSlug>Hotel Fire vo</itemSlug>
         <objID>M00702</objID>
         <mosID>testmos</mosID>
         <itemEdStart>0</itemEdStart>
         <itemEdDur>900</itemEdDur>
         <itemUserTimingDur>810</itemUserTimingDur>
```

```
      </item>
      <item>
         <itemID>1</itemID>
         <itemSlug>Dormitory Fire vo</itemSlug>
         <objID>M00705</objID>
         <mosID>testmos</mosID>
         <itemEdStart>0</itemEdStart>
         <itemEdDur>800</itemEdDur>
         <itemUserTimingDur>610</itemUserTimingDur>
      </item>
   </roItemReplace>
</mos>
```

3.6.10 roItemMoveMultiple – Move one or more Items to a specified position within a Story

Purpose

This message allows one or more items in a story to be moved to a new location in the story . The last itemID is the ID of the item before which to insert the new items. All remaining itemIDs identify items to insert at that location. The resulting story has all the moved items appearing before the reference item in the order specified in the command. If the last itemID is blank, the items are moved to the end of the story.

Note: The NCS should use the equivalent roElementAction message to achieve the same result.

Validation

There may be no duplicates in the list of itemIDs. This prevents the move from being ambiguous; if two IDs are the same, it is unclear where in the story the item with that ID must be placed.

Response

roAck

Port

MOS Upper Port (10541) – Running Order

Structural Outline

mos
- mosID
- ncsID
- messageID
- roItemMoveMultiple
  - roID
  - storyID
  - itemID+

Syntax

```
<!ELEMENT roItemMoveMultiple (roID, storyID, itemID+ )>
```

Example

In the 5PM running order, if the ‘Barn Fire’ story has items in the order 1 2 3 4 5 6, the following command will change the item order to 2 3 5 6 1 4.

```
<mos>
      <mosID>aircache.newscenter.com</mosID>
      <ncsID>ncs.newscenter.com</ncsID>
      <messageID>3011</messageID>
      <roItemMoveMultiple>
            <roID>5PM</roID>
            <storyID>Barn Fire</storyID>
            <itemID>2</itemID>
            <itemID>3</itemID>
            <itemID>5</itemID>
            <itemID>6</itemID>
            <itemID>1</itemID>
      </roItemMoveMultiple>
</mos>
```

3.6.11 roItemDelete – Delete Items in Story

Purpose

This message deletes one or more items in a story.

Note: The NCS should use the equivalent roElementAction message to achieve the same result.

Response

roAck

Port

MOS Upper Port (10541) – Running Order

Structural Outline

mos
- mosID
- ncsID
- messageID
- roItemDelete
  - roID
  - storyID
  - itemID+

Syntax

```
<!ELEMENT roItemDelete (roID, storyID, itemID+ )>
```

Example

```
<mos>
      <mosID>aircache.newscenter.com</mosID>
      <ncsID>ncs.newscenter.com</ncsID>
      <messageID>4000512</messageID>
      <roItemDelete>
            <roID>5PM</roID>
            <storyID>2</storyID>
            <itemID>4</itemID>
            <itemID>7</itemID>
            <itemID>10</itemID>
            <itemID>6</itemID>
      </roItemDelete>
</mos>
```

3.6.12 roElementAction – Performs specific Action on a Running Order

Purpose

This command executes INSERT, REPLACE, MOVE and DELETE operations on one or more elements in a playlist. The elements can be either Stories or Items. The command specifies one or more source elements and a single target element. The source elements are those Stories or Items to be acted upon. The target element specifies where in the running order the actions take place.

As with the story-level and item-level commands, the INSERT and REPLACE operations send new content to the MOS. Thus the element_source tag must contain either all Stories or all Items, depending on which is being inserted or replaced.

The MOVE and DELETE operations act on content already existing in the MOS. Thus the element_source tag must contain either all Story IDs or all Item IDs, depending on which is being moved or deleted.

The following table describes what goes in the element_source and the element_target for each of the eight possible operations.

Operation	In element_target	In element_source
Inserting stories	A storyID specifying the story before which the source stories are inserted	One or more stories to insert
Inserting items	A storyID and itemID specifying the item before which the source items are inserted	One or more items to insert
Replacing a story	A storyID specifying the story to be replaced	One or more stories to put in its place

Replacing an item	A storyID and itemID specifying the item to be replaced	One or more items to put in its place
Moving stories	A storyID specifying the story before which the source stories are moved	One or more storyIDs specifying the stories to be moved
Moving items	A storyID and itemID specifying the item before which the source items are moved	One or more itemIDs specifying the items in the story to be moved
Deleting stories	Not needed, since deletes don't happen relative to another story	One or more storyIDs specifying the stories to be deleted
Deleting items	A storyID specifying the story containing the items to be deleted	One or more itemIDs specifying the items in the story to be deleted

Note: This message effectively replaces messages 3.6.1 – 3.6.11 and will be supported in future version of MOS.

Response

roAck

Port

MOS Upper Port (10541) - Running Order

Structural Outline

mos
  mosID
  ncsID
  messageID
  roElementAction (operation = (INSERT, REPLACE, MOVE, DELETE))
    roID
    element_target
      storyID
      itemID?
    element_source
      story+ or item+ or storyID+ or itemID+

Syntax

```
<!ELEMENT roElementAction (roID, element_target?, element_source)>
<!ELEMENT element_target  (storyID, itemID?)>
<!ELEMENT element_source  (story+ | item+ | storyID+ | itemID+)>
 <!ATTLIST roElementAction operation CDATA #REQUIRED>
```

Because this command is complex, we provide several examples.

Insert example 1 - inserting a story in a rundown:

Insert a new story with ID=17 before the story with ID = 2 in the 5PM running order.

```
<mos>
      <mosID>aircache.newscenter.com</mosID>
      <ncsID>ncs.newscenter.com</ncsID>
       <messageID>4433250443</messageID>
      <roElementAction operation="INSERT">
            <roID>5PM</roID>
            <element_target>
                  <storyID>2</storyID>
            </element_target>
            <element_source>
              <story>
                  <storyID>17</storyID>
                  <storySlug>Barcelona Football</storySlug>
                  <storyNum>A2</storyNum>
                    <item>
                         <itemID>27</itemID>
                         <objID>M73627</objID>
                         <mosID>testmos</mosID>
                         <itemEdStart>0</itemEdStart>
                         <itemEdDur>715</itemEdDur>
                         <itemUserTimingDur>415</itemUserTimingDur>
                    </item>
                    <item>
                         <itemID>28</itemID>
                         <objID>M73628</objID>
                         <mosID>testmos</mosID>
                         <itemEdStart>0</itemEdStart>
                         <itemEdDur>315</itemEdDur>
                    </item>
              </story>
            </element_source>
        </roElementAction>
</mos>
```

Insert example 2 - inserting a new item into a story:

Insert a new item with ID=27 before the item with ID = 23 within the story with ID=2.

```
<mos>
      <mosID>aircache.newscenter.com</mosID>
      <ncsID>ncs.newscenter.com</ncsID>
      <messageID>44333</messageID>
      <roElementAction operation="INSERT">
            <roID>5PM</roID>
            <element_target>
                  <storyID>2</storyID>
                  <itemID>23</itemID>
            </element_target>
            <element_source>
              <item>
                  <itemID>27</itemID>
                  <itemSlug>NHL PKG</itemSlug>
                  <objID>M19873</objID>
                  <mosID>testmos</mosID>
                  <itemEdStart>0</itemEdStart>
                  <itemEdDur>700</itemEdDur>
                  <itemUserTimingDur>690</itemUserTimingDur>
              </item>
            </element_source>
        </roElementAction>
</mos>
```

Replace example 1 - replacing a story in a rundown:

Replace the story with ID=2 (in the 5PM running order) with a new story with ID = 17.

```
<mos>
      <mosID>aircache.newscenter.com</mosID>
      <ncsID>ncs.newscenter.com</ncsID>
```

```
      <messageID>44334</messageID>
      <roElementAction operation="REPLACE">
            <roID>5PM</roID>
            <element_target>
                  <storyID>2</storyID>
            </element_target>
            <element_source>
              <story>
                  <storyID>17</storyID>
                  <storySlug>Porto Football</storySlug>
                  <storyNum>A2</storyNum>
                    <item>
                         <itemID>27</itemID>
                         <objID>M73627</objID>
                         <mosID>testmos</mosID>
                         <itemEdStart>0</itemEdStart>
                         <itemEdDur>715</itemEdDur>
                         <itemUserTimingDur>415</itemUserTimingDur>
                    </item>
                    <item>
                         <itemID>28</itemID>
                         <objID>M73628</objID>
                         <mosID>testmos</mosID>
                         <itemEdStart>0</itemEdStart>
                         <itemEdDur>315</itemEdDur>
                    </item>
              </story>
            </element_source>
        </roElementAction>
</mos>
```

Replace example 2 - replacing an item in a story:

Replace the item with ID = 23 with new item with ID=27 within the story with ID=2.

```
<mos>
      <mosID>aircache.newscenter.com</mosID>
      <ncsID>ncs.newscenter.com</ncsID>
      <messageID>44335</messageID>
      <roElementAction operation="REPLACE">
            <roID>5PM</roID>
            <element_target>
                  <storyID>2</storyID>
                  <itemID>23</itemID>
            </element_target>
            <element_source>
              <item>
                  <itemID>27</itemID>
                  <itemSlug>NHL PKG</itemSlug>
                  <objID>M19873</objID>
                  <mosID>testmos</mosID>
                  <itemEdStart>0</itemEdStart>
                  <itemEdDur>700</itemEdDur>
                  <itemUserTimingDur>690</itemUserTimingDur>
              </item>
            </element_source>
        </roElementAction>
</mos>
```

Move example 1 - moving a story:

This moves the story with ID=7 before the story with ID=2 in the 5PM running order.

```
<mos>
      <mosID>aircache.newscenter.com</mosID>
      <ncsID>ncs.newscenter.com</ncsID>
      <messageID>44336</messageID>
      <roElementAction operation="MOVE">
            <roID>5PM</roID>
            <element_target>
```

```
                  <storyID>2</storyID>
            </element_target>
            <element_source>
                  <storyID>7</storyID>
            </element_source>
        </roElementAction>
</mos>
```

Move example 2 - moving a block of stories:

This moves stories with ID=7 and ID=12 before story with ID=2 in the 5PM running order.

```
<mos>
      <mosID>aircache.newscenter.com</mosID>
      <ncsID>ncs.newscenter.com</ncsID>
      <messageID>44337</messageID>
      <roElementAction operation="MOVE">
            <roID>5PM</roID>
            <element_target>
                  <storyID>2</storyID>
            </element_target>
            <element_source>
                  <storyID>7</storyID>
              <storyID>12</storyID>
            </element_source>
        </roElementAction>
</mos>
```

Move example 3 - moving items within a story:

This moves an item with ID=23 and ID= 24 before the item with ID=12 within the story with ID=2 in the 5PM running order.

```
<mos>
      <mosID>aircache.newscenter.com</mosID>
      <ncsID>ncs.newscenter.com</ncsID>
      <messageID>44338</messageID>
      <roElementAction operation="MOVE">
            <roID>5PM</roID>
            <element_target>
                  <storyID>2</storyID>
                  <itemID>12</itemID>
            </element_target>
            <element_source>
                  <itemID>23</itemID>
                  <itemID>24</itemID>
            </element_source>
        </roElementAction>
</mos>
```

Delete example 1 - deleting a story from the rundown:

This removes the story with ID=3 from the 5PM running order.

```
<mos>
      <mosID>aircache.newscenter.com</mosID>
      <ncsID>ncs.newscenter.com</ncsID>
      <messageID>44339</messageID>
      <roElementAction operation="DELETE">
            <roID>5PM</roID>
            <element_source>
                  <storyID>3</storyID>
            </element_source>
        </roElementAction>
```

```
</mos>
```

Delete example 2 - deleting items from a story:

This removes a items with ID=23 and ID=24 from the story with ID=2.

```
<mos>
      <mosID>aircache.newscenter.com</mosID>
      <ncsID>ncs.newscenter.com</ncsID>
      <messageID>44340</messageID>
      <roElementAction operation="DELETE">
            <roID>5PM</roID>
            <element_target>
                  <storyID>2</storyID>
            </element_target>
            <element_source>
                  <itemID>23</itemID>
                  <itemID>24</itemID>
            </element_source>
        </roElementAction>
</mos>
```

3.7 ro Control and Status feedback

3.7.1 roItemStat - Status of a Single Item in a MOS Running Order

Purpose

Method for the MOS to update the NCS on the status of an individual Item in a Running Order. This allows the NCS to reflect the status of individual Items in the MOS Running Order in the NCS Running Order display.

Response

roAck

Port

MOS Upper Port (10541) - Running Order

Structural Outline

mos
- mosID
- ncsID
- messageID
- roItemStat
  - roID
    - storyID
    - itemID
    - objID
    - status

time

Syntax

<!ELEMENT roItemStat (roID, storyID, itemID, objID, status, time)>

Example

```
<mos>
  <mosID>aircache.newscenter.com</mosID>
  <ncsID>ncs.newscenter.com</ncsID>
  <messageID>506702</messageID>
  <roItemStat>
    <roID>5PM</roID>
    <storyID>HOTEL FIRE</storyID>
    <itemID>0</itemID>
    <objID>A0295</objID>
    <status>PLAY</status>
    <time>1999-04-11T14:13:53</time>
  </roItemStat>
</mos>
```

3.7.2 roItemCue – Notification of Item Event

Purpose

Allows a device, such as prompter, to send a time cue for an Item.

Description

This command allows a non MOS or NCS device to send a time cue to the parent Media Object Server (or Automation MOS) for a specific Item event. This is not a command to execute or play. Instead, this is intended to provide feedback to the parent device as to the current execution point of the program.
The values <mosID>, <roID>, <storyID>, and <itemID> are derived from the Item reference embedded in a story. The story information is assumed to be transmitted via the roStorySend message.
The Media Object Server or automation device that receives this command may use this information to update status or generate device triggers. Optionally, the message may be redirected to the NCS as a means of providing additional status information.

Response

roAck

Port

MOS Upper Port (10541) - Running Order

Structural Outline

mos
- mosID
- ncsID
- messageID
- roItemCue
  - mosID
  - roID
  - storyID
  - itemID
  - roEventType
  - roEventTime
  - mosExternalMetadata*

Syntax

```
<!ELEMENT roItemCue (mosID, roID, storyID, itemID, roEventType, roEventTime, mosExternalMetadata*)>
<!ELEMENT roEventType (#PCDATA)>
<!ELEMENT roEventTime (#PCDATA)>
```

Example

An example of a notification message forced to the NCS:

```
<mos>
   <mosID>prompt.station.com</mosID>
   <ncsID>ncs.station.com</ncsID>
   <messageID>400932</messageID>
   <roItemCue>
      <mosID>videoserver.station.group.com</mosID>
      <roID>96857485</roID>
      <storyID>5983A501:0049B924:8390EF2B</storyID>
      <itemID>234343234</itemID>
      <roEventType>Prompter</roEventType>
      <roEventTime>2000-03-20T10:45:00.00</roEventTime>
      <mosExternalMetadata>
         <mosScope>PLAYLIST</mosScope>
         <mosSchema>http://ncsA4.com/mos/supported_schemas/NCSBBBBXML2.08</mosSchema>
         <mosPayload>
            <triggeredby>operator</triggeredby>
            <operatorType>prompter</operatorType>
            <netPropDelay>10</netPropDelay>
         </mosPayload>
      </mosExternalMetadata>
   </roItemCue>
</mos>
```

An example of a notification message sent to the parent MOS. Note the counterintuitive assignment of the parent MOS name to the <ncsID> field:

```
<mos>
   <mosID>prompt.station.com</mosID>
   <ncsID>videoserver.station.group.com</ncsID>
   <messageID>400932</messageID>
   <roItemCue>
      <mosID>videoserver.station.group.com</mosID>
      <roID>96857485</roID>
      <storyID>5983A501:0049B924:8390EF2B</storyID>
      <itemID>234343234</itemID>
      <roEventType>Prompter</roEventType>
      <roEventTime>2000-03-20T10:45:00.00</roEventTime>
      <mosExternalMetadata>
         <mosScope>PLAYLIST</mosScope>
         <mosSchema>http://ncsA4.com/mos/supported_schemas/NCSBBBBXML2.08</mosSchema>
         <mosPayload>
```

```
            <triggeredby>operator</triggeredby>
            <operatorType>prompter</operatorType>
            <netPropDelay>10</netPropDelay>
         </mosPayload>
      </mosExternalMetadata>
   </roItemCue>
</mos>
```

3.7.3 roCtrl – Running Order Control

Purpose

Allow basic control of a media object server via simple commands such as READY, EXECUTE, PAUSE, STOP and SIGNAL

Description

The roCtrl message allows control of a running order at three levels, the Running Order itself, Story, and Item. The commands READY, EXECUTE, PAUSE and STOP, as well as general indicator, SIGNAL, can be addressed at each level. In other words, a single command can begin EXECUTION of an entire Running Order, of a Story containing multiple Items, or of a single Item.

Response

roAck

Port

MOS Upper Port (10541) - Running Order

Structural Outline

mos
- mosID
- ncsID
- messageID
- roCtrl
  - roID
  - storyID
  - itemID
  - command
  - mosExternalMetadata*

Syntax

```
<!ELEMENT roCtrl (roID, storyID, itemID, command, mosExternalMetadata*)>
<!ELEMENT command (READY|EXECUTE|PAUSE|STOP|SIGNAL)>
```

Example

```
<mos>
   <mosID>aircache.newscenter.com</mosID>
   <ncsID>ncs.newscenter.com</ncsID>
   <messageID>3007</messageID>
   <roCtrl>
      <roID>3dedde9jd</roID>
      <storyID>A3fds3d</storyID>
      <itemID>30848</itemID>
      <command>EXECUTE</command>
   </roCtrl>
</mos>
```

3.8 Metadata Export

3.8.1 roStorySend – Send Story information, including Body of the Story

Purpose

This message enables sending the body of story from the NCS to a Media Object Server. Item references (storyItem) are embedded within the story's text. These item references are not intended to be displayed on the prompter, but instead can optionally be used to send a message (roItemCue) to the media object server indicated in the embedded reference. Composed from information in the embedded item reference, the roItemCue message could be generated by the prompter as this hidden text (storyItem) scrolls past the imaginary execution/read line of the prompter display.

The storyItem information can also optionally allow the prompter vendor to display the length of the embedded object and perhaps even a countdown.

Prompters, radio systems, external archive systems, accounting systems, and potentially other systems and devices can make use of this information.

Response

roAck

Port

MOS Upper Port (10541) - Running Order

Structural Outline

mos
- mosID
- ncsID
- messageID
- roStorySend
  - roID
  - storyID
  - storySlug?
    - storyNum?
    - storyBody

storyPresenter*
storyPresenterRR*
p*
em*
tab*
pi*
pkg*
b*
i*
u*
storyItem*
itemID
itemSlug?
objID
mosID
mosAbstract?
itemChannel?
itemEdStart?
itemEdDur?
itemUserTimingDur?
itemTrigger?
macroIn?
macroOut?
mosExternalMetadata*
mosExternalMetadata*

Syntax

```
<!ELEMENT roStorySend (roID, storyID, storySlug?, storyNum?, storyBody, mosExternalMetadata*)>
<!ELEMENT storyBody ((storyPresenter*, storyPresenterRR*, p*, storyItem*)*)>
<!ELEMENT storyPresenter (#PCDATA)>
<!ELEMENT storyPresenterRR (#PCDATA)>
<!ELEMENT storyItem (itemID, itemSlug?, objID, mosID, mosAbstract?, itemChannel?, itemEdStart?,
 itemEdDur?, itemUserTimingDur?, itemTrigger?, macroIn?, macroOut?, mosExternalMetadata*)>
<!ELEMENT p (#PCDATA | em | tab | pi | pkg | b | i | u)*>
<!ELEMENT pi (#PCDATA |  b | i | u)*>
<!ELEMENT pkg (#PCDATA |  b | i | u)*>
<!ELEMENT b (#PCDATA |  i | u)*>
<!ELEMENT i (#PCDATA |  b | u)*>
<!ELEMENT u (#PCDATA |  b | i)*>
```

Example - RoStorySend

In order to apply roStorySend to implementation of a prompter, you must use this message in conjunction with an roCreate message which sends all stories to the prompter from the Running Order (Forced Play List Construction), not just stories which contain the prompter's MOS ID. This establishes a list of all story ID's and pointers in the order they will air for the associated running order. The prompter uses this information to sequence the story information sent in the roStorySend message.

RO messages, such as roCreate, roStoryInsert, roStoryDelete, etc. should be sent before the roStorySend messages. RoStorySend messages can be sent alone after the story is initially referenced in an RO message (e.g. after roCreate or roStoryInsert) if only the body of the story has changed and not its position within the Running Order.

```
<mos>
   <mosID>prompt.station.com</mosID>
   <ncsID>ncs.station.com</ncsID>
```

```
   <messageID>507891</messageID>
   <roStorySend>
      <roID>96857485</roID>
      <storyID>5983A501:0049B924:8390EF1F</storyID>
      <storySlug>Show Open</storySlug>
      <storyNum>C8</storyNum>
      <storyBody>
         <storyPresenter>Suzie</storyPresenter>
         <storyPresenterRR>10</storyPresenterRR>
         <p>
            <pi> Smile </pi>
         </p>
         <p> Good Evening, I'm Suzie Humpries </p>
         <storyPresenter>Chet </storyPresenter>
         <storyPresenterRR>12</storyPresenterRR>
         <p> - and I'm Chet Daniels, this is the 5PM news on Monday November 5th.</p>
         <p>First up today - a hotel fire downtown</p>
         <storyItem>
            <itemID>1</itemID>
            <itemSlug>Hotel Fire vo</itemSlug>
            <objID>M000705</objID>
            <mosID>testmos</mosID>
            <itemEdStart>0</itemEdStart>
            <itemEdDur>800</itemEdDur>
            <itemUserTimingDur>310</itemUserTimingDur>
            <macroIn>c01/l04/dve07</macroIn>
            <macroOut>r00</macroOut>
            <mosExternalMetadata>
               <mosScope>PLAYLIST</mosScope>
               <mosSchema>http://MOSA4.com/mos/supported_schemas/MOSAXML2.08</mosSchema>
               <mosPayload>
                  <Owner>SHOLMES</Owner>
                  <transitionMode>2</transitionMode>
                  <transitionPoint>463</transitionPoint>
                  <source>a</source>
                  <destination>b</destination>
               </mosPayload>
            </mosExternalMetadata>
         </storyItem>
         <p>...as you can see, the flames were quite high. </p>
      </storyBody>
      <mosExternalMetadata>
         <mosScope>PLAYLIST</mosScope>
         <mosSchema>http://NCSA4.com/mos/supported_schemas/NCSAXML2.08</mosSchema>
         <mosPayload>
            <Owner>SHOLMES</Owner>
            <changedBy>MPalmer</changedBy>
            <length>463</length>
            <show>10 pm</show>
         </mosPayload>
      </mosExternalMetadata>
   </roStorySend>
</mos>
```

4 Other messages and data structures

4.1.1 heartbeat - Connection Confidence Indicator

Purpose

Message sent for the purpose of verifying network and application continuity.

Response

heartbeat

Port

MOS Lower Port (10540) - Media Object Metadata
MOS Upper Port (10541) - Running Order

Structural Outline

mos
mosID
ncsID
messageID
heartbeat
time

Syntax

```
<!ELEMENT heartbeat (time)>
```

Example

```
<mos>
  <mosID>aircache.newscenter.com</mosID>
  <ncsID>ncs.newscenter.com</ncsID>
  <messageID>9988</messageID>
  <heartbeat>
    <time>1999-04-11T17:20:42</time>
  </heartbeat>
</mos>
```

4.1.2 reqMachInfo - Request Machine Information

Purpose

Method for an NCS or MOS to determine more information about its counterpart.

Response

listMachInfo

Port

MOS Lower Port (10540) - Media Object Metadata
MOS Upper Port (10541) - Running Order

Structural Outline

mos
mosID
ncsID
messageID
reqMachInfo

Syntax

```
<!ELEMENT reqMachInfo EMPTY>
```

Example

```
<mos>
  <mosID>aircache.newscenter.com</mosID>
  <ncsID>ncs.newscenter.com</ncsID>
  <messageID>3940</messageID>
  <reqMachInfo/>
</mos>
```

4.1.3 listMachInfo - Machine Description List

Purpose

Method for an NCS or MOS to send information about itself.

Response

None

Port

MOS Lower Port (10540) - Media Object Metadata
MOS Upper Port (10541) - Running Order

Structural Outline

mos
- mosID
- ncsID
- messageID
- listMachInfo
  - manufacturer
  - model
  - hwRev
  - swRev
  - DOM
  - SN
  - ID
  - time
  - opTime?
  - mosRev
  - mosProfile0
  - mosProfile1
  - mosProfile2
  - mosProfile3
  - mosProfile4

- mosProfile5
- mosProfile6
- defaultActiveX*
  - mode
  - controlFileLocation
  - controlSlug
  - controlName
  - controlDefaultParams
- mosExternalMetadata*

NOTE: `No two <defaultActiveX> elements can have the same <mode> value.`

Syntax

```
<!ELEMENT listMachInfo (manufacturer, model, hwRev, swRev, DOM, SN, ID, time, opTime?, mosRev,
 mosProfile0, mosProfile1, mosProfile2, mosProfile3, mosProfile4, mosProfile5, mosProfile6,
 defaultActiveX*, mosExternalMetadata*)>
<!ELEMENT defaultActiveX (mode, controlFileLocation, controlSlug, controlName,
 controlDefaultParams)>
```

Example

```
<mos>
  <mosID>aircache.newscenter.com</mosID>
  <ncsID>ncs.newscenter.com</ncsID>
  <messageID>6331</messageID>
  <listMachInfo>
    <manufacturer>RadioVision, Ltd.</manufacturer>
    <model>TCS6000</model>
    <hwRev></hwRev>
    <swRev>2.1.0.37</swRev>
    <DOM></DOM>
    <SN>927748927</SN>
    <ID>airchache.newscenter.com</ID>
    <time>1999-04-11T17:20:42</time>
    <opTime>1999-03-01T23:55:10</opTime>
    <mosRev>2.8.1</mosRev>
    <mosProfile0>YES</mosProfile0>
    <mosProfile1>YES</mosProfile1>
    <mosProfile2>YES</mosProfile2>
    <mosProfile3>YES</mosProfile3>
    <mosProfile4>YES</mosProfile4>
    <mosProfile5>NO</mosProfile5>
    <mosProfile6>YES</mosProfile6>
    <defaultActiveX>
       <mode>CONTAINED</mode>
       <controlFileLocation>\\MOSDEVICE\Controls\</controlFileLocation>
       <controlSlug>Contained Control</controlSlug>
       <controlName>contained.containedCTRL.1</controlName>
       <controlDefaultParams>URL=http://containedcontrolpage.com</controlDefaultParams>
    </defaultActiveX>
    <defaultActiveX>
       <mode>MODAL</mode>
       <controlFileLocation>\\MOSDEVICE\Controls\</controlFileLocation>
       <controlSlug>MODAL Control</controlSlug>
       <controlName>modal.modalCTRL.1</controlName>
       <controlDefaultParams></controlDefaultParams>
    </defaultActiveX>
  </listMachInfo>
</mos>
```

4.1.4 mosExternalMetadata – External Metadata

Purpose

This data block can appear in several messages as a mechanism for transporting additional metadata, independent of schema or DTD.

Behavior

The value of the <scope> tag implies through what production processes this information will travel.

A scope of "OBJECT" implies this information is generally descriptive of the object and appropriate for queries.

A scope of "STORY" suggests this information may determine how the Object is used in a Story. For instance, Intellectual Property Management. This information will be stored and used with the Story.

A scope of "PLAYLIST" suggests this information is specific to describing how the Object is to be published, rendered, or played to air and thus, will be included in the Play List in addition to the Story.

This mechanism allows us to roughly filter external metadata and selectively apply it to different production processes and outputs. Specifically, it is neither advisable nor appropriate to send large amounts of inappropriate metadata to the Playlist in roCreate messages. In addition to these blocks of data being potentially very large, the media Object Server is, presumably, already aware of this data.

The value of the <mosSchema> tag will be descriptive of the schema used within the <mosPayload>. The value of <mosSchema> is implied to be a pointer or URL to the actual schema document.

The contents of <mosPayload> must be well formed XML, regardless of the schema used.

Structural Outline

mosExternalMetadata
- mosScope?
- mosSchema
- mosPayload

Syntax

```
<!ELEMENT mosExternalMetadata (mosScope?, mosSchema, mosPayload>

(note: The value of mosSchema is recommended to be a URI – the rightmost element of which is
 considered significant and uniquely identifying for the purposes of validation)
```

Example

```
<mosExternalMetadata>
    <mosScope>STORY</mosScope>
     <mosSchema>http://ncsA4.com/mos/supported_schemas/NCSAXML2.08</mosSchema>
    <mosPayload>
      <Owner>SHOLMES</Owner>
      <ModTime>20010308142001</ModTime>
      <mediaTime>0</mediaTime>
```

```
      <TextTime>278</TextTime>
      <ModBy>LJOHNSTON</ModBy>
      <Approved>0</Approved>
      <Creator>SHOLMES</Creator>
    </mosPayload>
</mosExternalMetadata>
```

4.1.5 mosItemReference (or "item") – Metadata block transferred by ActiveX Controls included in roCreate messages

Purpose

This data block appears in the MOS Protocol as a subset of the roCreate commands, but may also stand alone as recommended mechanism for transferring Item information from an NCS plug-in to the NCS. It is implied that this format will also be included in the body of the Story and thus will be output in the roStorySend messages.

Behavior

The metadata in the Item Reference is a description of how to execute playback or instantiation of an object, pointed to by the <objID>. The <mosID> is required for forced playlist construction, maintenance of Item References within stories and for association with MOS ActiveX components. The <mosAbstract> field provides a displayable abstract of the Object/Item, more verbose than the <itemSlug>. <mosAbstract> may contain formatting.

It is recommended that vendor or site specific tags be included in the <mosExternalMetadata> structure, not in the body of the message.

Structural Outline

item
- itemID
- itemSlug?
- objID
- mosID
- mosAbstract?
- itemChannel?
- itemEdStart?
- itemEdDur?
- itemUserTimingDur?
- itemTrigger?
- macroIn?
- macroOut?
- mosExternalMetadata*

Syntax

```
<!ELEMENT item (itemID, itemSlug?, objID, mosID, mosAbstract?, itemChannel?, itemEdStart?,
 itemEdDur?, itemUserTimingDur?, itemTrigger?, macroIn?, macroOut?, mosExternalMetadata*)>
```

Example

```
<item>
 <itemID>1</itemID>
 <itemSlug>Man Bites Dog vo</itemSlug>
 <objID>34323</objID>
 <mosID>ncs.com</mosID>
 <mosAbstract>Man Bites Dog vo trt :48</mosAbstract>
 <itemChannel>1</itemChannel>
 <itemEdStart>0</itemEdStart>
 <itemEdDur>1440</itemEdDur>
 <itemUserTimingDur>1310</itemUserTimingDur>
 <itemTrigger>manual</itemTrigger>
 <macroIn>2e9de</macroIn>
 <macroOut>2399a</macroOut>
 <mosExternalMetadata>
    <mosScope>PLAYLIST</mosScope>
     <mosSchema>http://mosA4.com/mos/supported_schemas/MOSAXML2.08</mosSchema>
     <mosPayload>
      <source>production</source>
      <machine>A5</machine>
    </mosPayload>
 <mosExternalMetadata>
<item>
```

4.1.6 messageID- Unique Identifier for Requests

Purpose

MOS messages which expect an answer from a recipient have a unique messageID as the value of a messageID field.
Messages which are only repeated messages due to a retry attempt have the same messageID as the original message sent at the first try.

Behavior

The messageID is useful in retry scenarios, when the NCS or the MOS Server is re-sending identical messages.

Possible usage in retry scenario:

The NCS sends a request (e.g. roStoryInsert) to the MOS Server.
The NCS receives no response within the timeout and therefore resets the connection.
The NCS sends the same request a second time.

The NCS cannot really know if the first sent message was processed by the MOS Server.
Assume the MOS Server processed the first sent message.
The MOS Server will receive the repeated request, but without a messageID it could not know that it is a repeated request, as the requests had no identity.
Therefore the MOS Server would be forced to process the repeated message, which will lead to an unwanted result in many cases.

When messageIDs are provided by the NCS the MOS Server can keep the messageIDs of the last received message(s).
When it receives a message now it can see from the messageID, whether it processed this message already or not, as only messages having identical messageIDs are repeated messages.

The scenario described is just one special situation of course, it is not relevant for messages which say "give me that piece of information" like the mosReqObj message.
However the messageID field was added to most messages, because there might be other usages also.

When debugging MOS integrations it will be helpful to see the messageID in the log files of both sides of a communication.

Messages including a messageID field:

Messages used for requests, which expect an answer from a recipient, have a unique messageID in the messageID field.
Examples: roCreate, mosObj

Messages used as response to a request have the same messageID as the messages they are the response to.
Examples: mosAck, roAck

Messages which are only repeated messages due to a retry attempt have the same messageID as the original message sent at the first try.

Mandatory field:

The messageID is a mandatory field in the messages it occurs in.
However an empty messageID tag is allowed for messages when used in the ActiveX interface, as for the ActiveX interface there is no need for a unique identifier.

Incrementing:

The sender in a MOS communication increments the messageID by one for each new request it sends, the last used messageID must be persistent.
The messageID wraps to 1 when the limit of the data type is reached.

Syntax

```
<!ELEMENT messageID (#PCDATA)>
```

The contents of the element must be a 32-bit signed integer, decimal or hexadecimal, with a value larger than or equal to 1.

Example

```
<messageID>437</messageID>
```

5. MOS v2.8.1 ActiveX Control Specification

This specification describes the way an NCS Host and an ActiveX Plug-In hosted inside it interact. The two main goals are:

- To allow flexibility in defining the size and function of the ActiveX Plug-In
- To allow modification of Item References inside NCS stories
-

It is assumed that any communication initiated by either application is the result of the user who is running them performing an action.

5.1 Methods, Events and Data Types

Methods

mosMsgFromHost

VB code to call the method might look like this:

```
sActiveXResponse = mosMsgFromHost(mosMsg)
```

Events

mosMsgFromPlugIn

VB code for the NCS Host event handler might look like this:

```
Private Sub MosActiveX_mosMsgFromPlugIn(mosMsg as String, mosResponse
      as String)
     ' Process mosMsg...

     ' Assign Response accordingly..
     mosResponse = GetROStorySend()
End Sub
```

Data Type

mosMsg (Unicode UCS-2)

5.2 Behavior

General

Messages are sent from the NCS Host to the ActiveX Plug-In via the mosMsgFromHost Method or the OLE drag and drop data type mosMsg. Responses to select messages are returned via the return value of the mosMsgFromHost method.

Messages are sent from the ActiveX Plug-In to the NCS Host via the mosMsgFromPlugIn event or the OLE drag and drop data type mosMsg. Responses to mosMsgFromPlugIn events are returned via the mosResponse parameter. Messages sent via OLE drag and drop receive no response.

We have tried to enable data transfer between the NCS Host and the ActiveX Plug-In via drag and drop operations as well as non-drag and drop Methods and Events. Thus it should be possible for application developers to enable keyboard equivalent operations for most drag and drop operations.

Start Up:

1) Before the ActiveX Plug-In is instantiated, the NCS Host determines whether it will be instantiated in modal or non-modal mode

2) The NCS Host determines what options for screen metrics are available for the control.

3) The NCS Host enumerates these options, giving option "0" to the metrics within which the control will be initially instantiated. It is recommended that the NCS Host provide enumerated options for both absolute maximum and minimum screen metrics.

4) If the NCS Host chooses, it can optionally place either a mosObj message or Item info in the ncsAppInfo message. The storyID must be included if Item information is sent. The roID can optionally be included with Item information.

5) The NCS Host chooses the mode the control will be instantiated in and includes this in the ncsAppInfo message.

6) The NCS Host optionally provides additional context of BROWSE, EDIT or CREATE. The semantics of these choices are:

    a. BROWSE tells the control to provide the ability to look at the inventory list for the MOS, and optionally to preview specific MOS Objects;
    b. EDIT tells the control to provide the ability to edit a MOS item that is passed in the ncsAppInfo message;

c. CREATE tells the control to provide the ability to create a new MOS Object on the Media Object server.

7) The NCS Host identifies the local user's name and places this in the ncsAppInfo message.

8) The NCS Host also identifies its own manufacturer name, product name, and software version in the ncsAppInfo message.

9) The NCS Host instantiates the control.

10) The NCS Host sends the ncsAppInfo message via the mosMsgFromHost method.

11) The ActiveX Plug-In recognizes the mode it was instantiated in and checks for optional context (BROWSE, EDIT or CREATE) and the availability of window closing functionality.

12) The ActiveX Plug-In optionally recognizes and uses the user's name, manufacturer name, product name and software version.

13) The ActiveX Plug-In then checks for either a mosObj structure or Item structure embedded in the ncsAppInfo message and, if it is present, uses this information to fetch the appropriate object from its associated database/storage.

Additional functionality:

1) Story Information can be sent from the NCS Host to the ActiveX Plug-In in three different manners.

a. Users can choose to drag and drop a Story from the NCS Host to the ActiveX Plug-In.

The NCS Host will form an roStorySend message.

If a value for roID does not exist a value of "0" will be used.

The roStorySend message will be sent via OLE drag and drop and the mosMsg data type or the mosMsgFromHost method.

b. An alternate method exists which allows the ActiveX Plug-In to request the NCS Host send Story information.

Rather than use drag and drop to send Story data from the NCS Host to the ActiveX Plug-In, the ncsStoryRequest message can alternately be used by the ActiveX Plug-In to request Story information from the NCS Host via the mosMsgFromPlugIn event and mosResponse returned parameter.

An roStorySend message is sent from the NCS Host in the mosResponse parameter as a normal response.

If the NCS Host cannot logically respond to the request, then an ncsAck message with

a status value of "ERROR" is returned to the ActiveX Plug-In instead of the roStorySend message.

c. Unsolicited roStorySend messages can also be sent from the NCS Host to the ActiveX Plug-In.

The NCS Host can initiate the stand alone roStorySend message, via the mosMsgFromHost method.

An ncsAck message is returned with a value of either "ACK" or "ERROR" through the return value of the mosMsgFromHost method.

2) Item reference information can be exchanged between the NCS Host and the ActiveX Plug-In in three different manners.

d. Users can choose to drag and drop an Item reference from the ActiveX Plug-In to the NCS Host.

The ActiveX Plug-In will form an ncsItem message, using an itemID value of "0" (the NCS Host will actually ignore the itemID value), and send this message via OLE drag and drop and the mosMsg data type.

e. A second method exists to send Item reference information between the ActiveX Plug-In and the NCS Host.

The ncsItemRequest message can alternately be used to send Item information from either from the NCS Host to the ActiveX Plug-In via the mosMsgFromHost Method and return value, or from the ActiveX Plug-In to the NCS Host via the mosMsgFromPlugIn event and mosResponse parameter.

This allows either the ActiveX Plugin or NCS Host to request the other send an Item reference.

An ncsItem message is sent as a normal response.

If one side or the other cannot logically respond to the request, then an ncsAck message with a status value of "ERROR" is returned instead of the ncsItem message.

f. Unsolicited ncsItem messages can be sent between the ActiveX Plug-In and the NCS Host.

Either the ActiveX Plug-In or the NCS Host can initiate the stand alone ncsItem message, via the mosMsgFromHost method or the mosMsgFromPlugIn event.

An ncsAck message is returned with a status value of either "ACK" or "ERROR" through either the value of the mosMsgFromHost method or the mosResponse parameter of the mosMsgFromPlugIn event.

3) The ActiveX Plug-In may request the window in which it runs be resized to one of an enumerated list of modes provided by the NCS Host.

   The ActiveX Plug-In sends an ncsReqAppInfo message to the NCS Host via the mosMsgFromPlugIn event.

   The NCS Host responds with an ncsAppInfo message sent via the mosResponse return parameter of the mosMsgFromPlugIn event.

   Alternately, the ncsAppInfo message received on start-up can be used if a significant delay does not exist between start up and the resize request.

   The ncsAppInfo message includes an enumerated list of display metrics which it will support and from which the plug-in may choose. Enumerated option "0" will always be the current mode.

   It is recommended that the NCS Host always return options for maximum and minimum possible display metrics.

   The ActiveX Plug-In then chooses the new mode in which it wishes to run and sends the mode number to the NCS Host in a ncsReqAppMode message via a second mosMsgFromPlugIn event.

   The NCS Host will then respond by resizing the window and sending a final ncsAppInfo message, via the mosResponse return parameter of the mosMsgFromPlugIn event, indicating the new current mode as enumerated option "0".

4) The ActiveX Plug-In may request the window in which it runs be closed.

   The ActiveX Plug-In sends an ncsReqAppClose message to the NCS Host via the mosMsgFromPlugIn event.

   If the request is not supported by the NCS Host for the current mode, the NCS Host will take no action, except to return an ncsAck message with a status of ERROR. This should be a rare event – the ActiveX should check for support of the message in the ncsAppInfo message.

   If the request is supported by the NCS Host for the current mode, the NCS Host will close the window. The NCS may, at its option, transfer the ActiveX control to another window or release its reference to the object.

   If the NCS Host changes the mode or context of the control rather than releasing it, it will return an ncsAppIno message indicating the new mode in the mosResponse parameter.

   If the NCS Host releases its reference to the ActiveX control, it will return an ncsAck message with a status of ACK. It may release its reference to the ActiveX during the mosMsgFromPlugIn event or after returning. The ActiveX must not allow itself to be destroyed until after the call has returned. Depending on the compiler functionality, this may require the temporary increment of the object's reference count until the call has completed. Failure to do so may result in an access violation when the call returns.

5.3 ActiveX Communication messages

MOS Messages

5.3.1 ncsAck - Acknowledge message

Purpose

This allows either the NCS Host or ActiveX Plug-In to acknowledge receipt of certain messages. The status value is either "ACK" or "ERROR." If it is "ERROR," the statusDescription value optionally describes the problem.

Response

N/A

Structural Outline

```
mos
    ncsAck
        status
        statusDescription?
```

Syntax

```
<!ELEMENT ncsAck (status, statusDescription?)>
```

Example

```
<mos>
        <ncsAck>
                <status>ERROR</status>
        </ncsAck>
</mos>
```

5.3.2 ncsReqAppInfo – Request Application information and context

Purpose

This allows the ActiveX Plug-In to request contextual information from the NCS Host. If the NCS Host can fulfill the request, it returns the ncsAppInfo message; otherwise, it returns ncsAck with a status of "ERROR."

Response

ncsAppInfo

or

ncsAck

Structural Outline

```
        mos
                ncsReqAppInfo
```

Syntax

```
<!ELEMENT ncsReqAppInfo ()>
```

Example

```
<mos>
        <ncsReqAppInfo/>
</mos>
```

5.3.3 ncsAppInfo – Application information and context

Purpose

This allows the NCS Host to send contextual information to the ActiveX Plug-In. The information includes product information about the NCS Host, the context in which the ActiveX Plug-In can be instantiated, the available screen sizes and modes for the ActiveX Plug-In window and whether the window can be closed by the ActiveX.

Note: This message has two forms. The first uses the mosObj structure and the second uses the roID/StoryID/item structure. These tags are listed as optional; the message can be used without context of mosObj/roID/StoryID/Item information if the purpose is to instantiate an "empty" control.

The following are valid values of <mode>:

CONTAINED – The ActiveX Plug-In window is contained in an NCS Host window.
MODALDIALOG – The ActiveX Plug-In window is contained in an NCS Host modal dialog.
NONMODAL – The ActiveX Plug-In window is contained in an NCS Host modeless dialog.
TOOLBAR – The ActiveX Plug-In window is contained in an NCS Host toolbar.

Response

None if sent as a response to ncsReqAppInfo

or

ncsAck if sent as an unsolicited message

Structural Outline

```
mos
    mosID
    ncsID
    messageID
    ncsAppInfo
        mosObj?
        roID?
        storyID?
        item?
        ncsInformation
            userID
            runContext
            software
                manufacturer
                product
                 version
            mosActiveXversion
            UImetric num="x"
                startx
                starty
                endx
```

endy
mode
canClose?

Note: The optional mosObj and item elements shown in the outline refer to the corresponding elements defined in the MOS 2.8.1 Protocol. See [link] and [link], respectively.

Syntax

```
<!ELEMENT ncsAppInfo (mosObj?, roID?, storyID?, item?, ncsInformation)>
<!ELEMENT ncsInformation (userID, runContext, software, Uimetric*)>
<!ELEMENT software (manufacturer, product, version)>
<!ELEMENT UIMetric (startx, starty, endx, endy, mode, canClose?)>
```

Example – mosObj form (note: no roID, storyID, or item structure is included)

```
<mos>
        <mosID>aircache.newscenter.com</mosID>
        <ncsID>ncs.newscenter.com</ncsID>
        <messageID/>
          <ncsAppInfo>
            <mosObj>
                <objID>M000123</objID>
                <objSlug>Hotel Fire</objSlug>
                <mosAbstract>
                        <b>Hotel Fire</b>
                        <em>vo</em>:30</mosAbstract>
                <objGroup>Show 7</objGroup>
                <objType>VIDEO</objType>
                <objTB>60</objTB>
                <objRev>1</objRev>
                <objDur>1800</objDur>
                <status>NEW</status>
                <objAir>READY</objAir>
                <createdBy>Chris</createdBy>
                <created>1998-10-31T23:39:12</created>
                <changedBy>Chris</changedBy>
                <changed>1998-10-31T23:39:12</changed>
                <description>
                        <p>Exterior footage of <em>Baley Park Hotel</em>on fire with natural sound. Trucks are visible for the first portion of the clip.
<em>CG locator at 0:04 and duration 0:05, Baley Park Hotel.</em>
                        </p>
                        <p>
                                <tab/>Cuts to view of fire personnel exiting hotel lobby and cleaning up after the fire is out.</p>
                        <p>
                                <em>Clip has been doubled for pad on voice over.</em>
                        </p>
                </description>
                <mosExternalMetadata>
                        <mosScope>STORY</mosScope>
                        <mosSchema>http://MOSA4.com/mos/supported_schemas/MOSAXML2.08</mosSchema>
                        <mosPayload>
                                <Owner>SHOLMES</Owner>
                                <ModTime>20010308142001</ModTime>
                                <mediaTime>0</mediaTime>
                                <TextTime>278</TextTime>
                                <ModBy>LJOHNSTON</ModBy>
                                <Approved>0</Approved>
                                <Creator>SHOLMES</Creator>
                        </mosPayload>
```

```
                </mosExternalMetadata>
            </mosObj>
        <ncsInformation>
                <userID>JOHNDOE</userID>
                <runContext>BROWSE</runContext>
                <software>
                        <manufacturer>Good Software R Us</manufacturer>
                        <product>story composer</product>
                        <version>8.9.3</version>
                </software>
                <UImetric num="0">
                        <startx>690</startx>
                        <starty>230</starty>
                        <endx>690</endx>
                        <endy>230</endy>
                        <mode>CONTAINED</mode>
                        <canClose>FALSE</canClose>
                </UImetric>
                <UImetric num="1">
                        <startx>810</startx>
                        <starty>100</starty>
                        <endx>810</endx>
                        <endy>490</endy>
                        <mode>CONTAINED</mode>
                        <canClose>FALSE</canClose>
                </UImetric>
                <UImetric num="2">
                        <startx>1000</startx>
                        <starty>575</starty>
                        <endx>1000</endx>
                        <endy>575</endy>
                        <mode>NONMODAL</mode>
                        <canClose>TRUE</canClose>
                </UImetric>
                <UImetric num="3">
                        <startx>200</startx>
                        <starty>50</starty>
                        <endx>400</endx>
                        <endy>50</endy>
                        <mode>TOOLBAR</mode>
                        <canClose>FALSE</canClose>
                </UImetric>
                <UImetric num="4">
                        <startx/>
                        <starty/>
                        <endx/>
                        <endy/>
                        <mode/>
                </UImetric>
          </ncsInformation>
       </ncsAppInfo>
</mos>
```

Example – item form (note: no mosObj structure is included)

```
<mos>
        <mosID>aircache.newscenter.com</mosID>
        <ncsID>ncs.newscenter.com</ncsID>
        <messageID/>
        <ncsAppInfo>
        <roID/>
        <storyID/>
        <item>
                <itemID>2</itemID>
                <itemSlug>Cat bites dog VO</itemSlug>
```

```
                <objID>A0323H1233309873139AQz</objID>
                <mosID>aircache.newscenter.com</mosID>
                <mosAbstract>Cat Bites Dog VO :17</mosAbstract>
                <itemChannel>A</itemChannel>
                <itemEdStart>0</itemEdStart>
                <itemEdDur>510</itemEdDur>
                <itemUserTimingDur>310</itemUserTimingDur>
                <itemTrigger>MANUAL</itemTrigger>
                <macroIn>1;7;5</macroIn>
                <macroOut>2;8;4</macroOut>
                <mosExternalMetadata>
                        <mosScope>STORY</mosScope>
                        <mosSchema>http://ncsA4.com/mos/supported_schemas/NCSAXML2.08</mosSchema>
                        <mosPayload>
                                <Owner>SHOLMES</Owner>
                                <ModTime>20010308142001</ModTime>
                                <mediaTime>0</mediaTime>
                                <TextTime>278</TextTime>
                                <ModBy>LJOHNSTON</ModBy>
                                <Approved>0</Approved>
                                <Creator>SHOLMES</Creator>
                        </mosPayload>
                </mosExternalMetadata>
        </item>
        <ncsInformation>
                <userID>JOHNDOE</userID>
                <runContext>BROWSE</runContext>
                <software>
                        <manufacturer>Good Software R Us</manufacturer>
                        <product>story composer</product>
                        <version>8.9.3</version>
                </software>
                <UImetric num="0">
                        <startx>690</startx>
                        <starty>230</starty>
                        <endx>690</endx>
                        <endy>230</endy>
                        <mode>CONTAINED</mode>
                        <canClose>FALSE</canClose>
                </UImetric>
                <UImetric num="1">
                        <startx>810</startx>
                        <starty>100</starty>
                        <endx>810</endx>
                        <endy>490</endy>
                        <mode>CONTAINED</mode>
                        <canClose>FALSE</canClose>
                </UImetric>
                <UImetric num="2">
                        <startx>1000</startx>
                        <starty>575</starty>
                        <endx>1000</endx>
                        <endy>575</endy>
                        <mode>NONMODAL</mode>
                        <canClose>TRUE</canClose>
                </UImetric>
                <UImetric num="3">
                        <startx>200</startx>
                        <starty>50</starty>
                        <endx>400</endx>
                        <endy>50</endy>
                        <mode>TOOLBAR</mode>
                        <canClose>FALSE</canClose>
                </UImetric>
                <UImetric num="4">
                        <startx/>
                        <starty/>
                        <endx/>
                        <endy/>
```

```
                    <mode/>
                </UImetric>
        </ncsInformation>
        </ncsAppInfo>
</mos>
```

5.3.4 ncsReqAppMode – Request to run Plug-In in different size window

Purpose

This allows the ActiveX Plug-In to request the NCS Host to allow it to run in a different mode. This mode must be chosen from an enumerated list provided by the NCS Host in the ncsAppInfo message. If the NCS Host can fulfill the request, it returns ncsAppInfo containing a list of display metrics as described in part 3 of the "Additional Functionality" section. If it cannot fulfill the request, it returns ncsAck with a status of "ERROR."

Response

ncsAppInfo

or

ncsAck

Structural Outline

mos
ncsReqAppMode
UImetric num="x"

Syntax

```
<!ELEMENT ncsReqAppMode (UImetric)>
```

Example

```
<mos>
        <ncsReqAppMode>
                <UImetric num="1"/>
        </ncsReqAppMode>
</mos>
```

5.3.5 ncsStoryRequest – Request the NCS Host to send a Story

Purpose

This allows the ActiveX Plug-In to request the NCS Host to send it the body of a Story. The NCS Host must determine which Story to send and whether this message is allowed in a specific context. If there is an error or the message is not allowed, then an ncsAck message must be sent with a status of "ERROR".

Response

roStorySend
or
ncsAck

Structural Outline

```
mos
    ncsStoryRequest
```

Syntax

```
<!ELEMENT ncsStoryRequest EMPTY>
```

Example

```
<mos>
    <ncsStoryRequest/>
</mos>
```

5.3.6 ncsItemRequest – Request the NCS Host or ActiveX Plug-In to send an Item

Purpose

This allows either application to request the other application to send an Item. The requested application must determine which Item to send and whether this message is allowed in a specific context. If there is an error or the message is not allowed, then an ncsAck message must be sent with a status of "ERROR".

Response

ncsItem

or
ncsAck

Structural Outline

mos
ncsItemRequest

Syntax

<!ELEMENT ncsItemRequest ()>

Example

```
<mos>
	<ncsItemRequest/>
</mos>
```

5.3.7 roStorySend – Allows the NCS Host to send a Story Body to the ActiveX Plug-In

Purpose

This allows the NCS Host to send the body of a story and associated metadata to the ActiveX Plug-In. This can be done in an unsolicited manner or as a result of ncsStoryRequest.

Response

None if as a response to ncsStoryRequest

or

ncsAck if sent as an unsolicited message

Structural Outline

mos
mosID
ncsID
messageID
roStorySend

roID
storyID
storySlug?
storyNum?
storyBody
storyPresenter*
storyPresenterRR*
p*
em*
tab*
pi*
pkg*
b*
I*
u*
storyItem*
itemID
itemSlug?
mosID
mosPlugInID?
objID
objSlug?
objDur
objTB
mosAbstract?
itemChannel?
itemEdStart?
itemEdDur?
itemUserTimingDur?
itemTrigger?
macroIn?
macroOut?
mosExternalMetadata*
mosExternalMetadata*

Syntax

```
<!ELEMENT roStorySend (roID, storyID, storySlug?, storyNum?, storyBody, mosExternalMetadata*)>
<!ELEMENT storyBody (storyPresenter*, storyPresenterRR*, p*, storyItem*)>
<!ELEMENT storyPresenter (#PCDATA)>
<!ELEMENT storyPresenterRR (#PCDATA)>
<!ELEMENT storyItem (itemID, itemSlug?, mosID, mosPlugInID, objID, objSlug, objDur, objTB, mosAbstract?, itemChannel?,
 itemEdStart?, itemEdDur?, itemUserTimingDur?,  itemTrigger?, macroIn?, macroOut?, mosExternalMetadata*)>
<!ELEMENT p (#PCDATA | em | tab | pi | pkg | b | i | u)*>
<!ELEMENT pi (#PCDATA |  b | i | u)*>
<!ELEMENT pkg (#PCDATA |  b | i | u)*>
<!ELEMENT b (#PCDATA |  i | u)*>
<!ELEMENT i (#PCDATA |  b | u)*>
<!ELEMENT u (#PCDATA |  b | i)*>
```

Example

```
<mos>
  <mosID>prompt.station.com</mosID>
  <ncsID>ncs.station.com</ncsID>
  <messageID/>
  <roStorySend>
    <roID>96857485</roID>
    <storyID>5983A501:0049B924:8390EF1F</storyID>
    <storySlug>Show Open</storySlug>
    <storyNum>C8</storyNum>
    <storyBody>
      <storyPresenter>Suzie</storyPresenter>
      <storyPresenterRR>10</storyPresenterRR>
      <p>
        <pi> Smile </pi>
      </p>
      <p> Good Evening, I'm Suzie Humpries </p>
      <storyPresenter>Chet </storyPresenter>
      <storyPresenterRR>12</storyPresenterRR>
      <p> - and I'm Chet Daniels, this is the 5PM news on Monday November 5th.</p>
      <p>First up today - a hotel fire downtown</p>
                    <storyItem>
                          <itemID>2</itemID>
                          <itemSlug>Cat bites dog VO</itemSlug>
                          <objID>A0323H1233309873139AQz</objID>
                          <mosID>aircache.newscenter.com</mosID>
                          <mosAbstract>Cat Bites Dog VO :17</mosAbstract>
                          <itemChannel>A</itemChannel>
                          <itemEdStart>0</itemEdStart>
                          <itemEdDur>510</itemEdDur>
                          <itemUserTimingDur>400</itemUserTimingDur>
                          <itemTrigger>MANUAL</itemTrigger>
                          <macroIn>1;7;5</macroIn>
                          <macroOut>2;8;4</macroOut>
                          <mosExternalMetadata>
                                <mosScope>STORY</mosScope>
                                <mosSchema>http://ncsA4.com/mos/supported_schemas/NCSAXML2.08</mosSchema>
                                <mosPayload>
                                      <Owner>SHOLMES</Owner>
                                      <ModTime>20010308142001</ModTime>
                                      <mediaTime>0</mediaTime>
                                      <TextTime>278</TextTime>
                                      <ModBy>LJOHNSTON</ModBy>
                                      <Approved>0</Approved>
                                      <Creator>SHOLMES</Creator>
                                </mosPayload>
                          </mosExternalMetadata>
                    </storyItem>
      <p>...as you can see, the flames were quite high. </p>
    </storyBody>
    <mosExternalMetadata>
      <mosScope>PLAYLIST</mosScope>
      <mosSchema>http://NCSA4.com/mos/supported_schemas/NCSAXML2.08</mosSchema>
      <mosPayload>
        <Owner>SHOLMES</Owner>
        <changedBy>MPalmer</changedBy>
        <length>463</length>
        <show>10 pm</show>
      </mosPayload>
    </mosExternalMetadata>
  </roStorySend>
</mos>
```

5.3.8 ncsItem – Allows either the ActiveX Plug-In or NCS Host to send an Item Reference to the other application

Purpose

This allows either application to send an Item reference to the other application. The receiving application must determine how to best handle the data.

Note: ItemID is sent as the reserved value "0" which replaced with an actual value by the NCS Host before insertion into the body of a Story.

Response

none if in response to an ncsItemRequest message

or

ncsAck if sent as an unsolicited message

Structural Outline

mos
- ncsItem
  - item
    - itemID
    - itemSlug?
    - mosID
    - objID
    - mosPlugInID?
    - objSlug?
    - objDur
    - objTB
    - mosAbstract?
    - itemChannel?
    - itemEdStart?
    - itemEdDur?
    - itemUserTimingDur?
    - itemTrigger?
    - macroIn?
    - macroOut?
    - mosExternalMetadata*

Syntax

```
<!ELEMENT ncsItem (item)>
<!ELEMENT item (itemID, itemSlug?, mosID, mosPlugInID, objID, objSlug, objDur, objTB, mosAbstract?, itemChannel?, itemEdStart?,
 itemEdDur?, itemUserTimingDur?,  itemTrigger?, macroIn?, macroOut?, mosExternalMetadata*)>
```

Example

```
<mos>
	<ncsItem>
		<item>
			<itemID>2</itemID>
			<itemSlug>Cat bites dog VO</itemSlug>
			<objID>A0323H1233309873139AQz</objID>
			<mosID>aircache.newscenter.com</mosID>
			<mosPlugInID>Shell.Explorer.2</mosPlugInID>
			<objSlug>Cat Bites Dog VO</objSlug>
			<objDur>510</objDur>
			<objTB>30</objTB>
			<mosAbstract>Cat Bites Dog VO :17</mosAbstract>
			<itemChannel>A</itemChannel>
			<itemEdStart>0</itemEdStart>
			<itemEdDur>510</itemEdDur>
			<itemUserTimingDur>310</itemUserTimingDur>
			<itemTrigger>MANUAL</itemTrigger>
			<macroIn>1;7;5</macroIn>
			<macroOut>2;8;4</macroOut>
			<mosExternalMetadata>
				<mosScope>STORY</mosScope>
				<mosSchema>http://ncsA4.com/mos/supported_schemas/NCSAXML2.08</mosSchema>
				<mosPayload>
					<Owner>SHOLMES</Owner>
					<ModTime>20010308142001</ModTime>
					<mediaTime>0</mediaTime>
					<TextTime>278</TextTime>
					<ModBy>LJOHNSTON</ModBy>
					<Approved>0</Approved>
					<Creator>SHOLMES</Creator>
				</mosPayload>
			</mosExternalMetadata>
		</item>
	</ncsItem>
</mos>
```

5.3.9 mosItemReplace – Allows the ActiveX Plug-In to replace an existing Item in a Story

Purpose

This allows the ActiveX Plug-In to replace an Item Reference embedded in a Story. It is up to the NCS Host to determine which Story will be the target of this action. It is implied that the ActiveX Plug-In previously became aware of the contents of the Story through a preceding roStorySend message.

Note: The Media Object Server to which the ActiveX Plug-In is connected should never cause an ActiveX Plug-In to initiate this message. The MOS should send the mosItemReplace message from the MOS 2.8.1 Protocol instead.

Note: ItemIDs must match in order for the replace operation to take place.

Note: This is a strict replace; the existing Item Reference is removed and the new Item Reference is inserted. This is different from the action of the mosItemReplace message sent from a Media Object Server to a

Newsroom Computer System, where the new Item Reference is merged into the existing Item Reference.

Response

ncsAck

Structural Outline

mos
- mosID
- ncsID
- messageID
- mosItemReplace
  - roID?
  - storyID
  - item
    - itemID
    - itemSlug?
    - mosID
    - mosPlugInID?
    - objID
    - objSlug?
    - objDur
    - objTB
    - mosAbstract?
    - itemChannel?
    - itemEdStart?
    - itemEdDur?
    - itemUserTimingDur?
    - itemTrigger?
    - macroIn?
    - macroOut?
    - mosExternalMetadata*

Syntax

```
<!ELEMENT mosItemReplace (roID, storyID, item)>
<!ELEMENT item (itemID, itemSlug?, mosID, mosPlugInID, objID, objSlug, objDur, objTB, mosAbstract?, itemChannel?, itemEdStart?,
 itemEdDur?, itemUserTimingDur? , itemTrigger?, macroIn?, macroOut?, mosExternalMetadata*)>
```

Example

```
<mos>
  <mosID>aircache.newscenter.com</mosID>
  <ncsID>ncs.newscenter.com</ncsID>
  <messageID/>
  <mosItemReplace>
    <roID>5PM</roID>
    <storyID>HOTEL FIRE</storyID>
        <item>
                <itemID>2</itemID>
```

```
                    <itemSlug>Cat bites dog VO</itemSlug>
                    <objID>A0323H1233309873139AQz</objID>
                    <mosID>aircache.newscenter.com</mosID>
                    <mosPlugInID>Shell.Explorer.2</mosPlugInID>
                    <objSlug>Cat Bites Dog VO</objSlug>
                    <objDur>510</objDur>
                    <objTB>30</objTB>
                    <mosAbstract>Cat Bites Dog VO :17</mosAbstract>
                    <itemChannel>A</itemChannel>
                    <itemEdStart>0</itemEdStart>
                    <itemEdDur>510</itemEdDur>
                    <itemUserTimingDur>310</itemUserTimingDur>
                    <itemTrigger>MANUAL</itemTrigger>
                    <macroIn>1;7;5</macroIn>
                    <macroOut>2;8;4</macroOut>
                    <mosExternalMetadata>
                              <mosScope>STORY</mosScope>
                              <mosSchema>http://ncsA4.com/mos/supported_schemas/NCSAXML2.08</mosSchema>
                              <mosPayload>
                                        <Owner>SHOLMES</Owner>
                                        <ModTime>20010308142001</ModTime>
                                        <mediaTime>0</mediaTime>
                                        <TextTime>278</TextTime>
                                        <ModBy>LJOHNSTON</ModBy>
                                        <Approved>0</Approved>
                                        <Creator>SHOLMES</Creator>
                              </mosPayload>
                    </mosExternalMetadata>
          </item>
  </mosItemReplace>
</mos>
```

5.3.10 ncsReqAppClose – Request to close window for Plug-In

Purpose

This allows the ActiveX Plug-In to request the NCS Host to close the window in which it is hosted. If the NCS Host can fulfill the request, it closes the window and returns ncsAppInfo (if it moves the control to another window) or ncsAck with a status of ACK (if it releases its reference to the control) as described in part 4 of the "Additional Functionality" section. If it cannot fulfill the request, it returns ncsAck with a status of "ERROR."

Response

ncsAppInfo

or

ncsAck

Structural Outline

```
mos
    ncsReqAppClose
```

Syntax

```
<!ELEMENT ncsReqAppClose EMPTY>
```

Example

```
<mos>
        <ncsReqAppClose/>
</mos>
```

6. Field Descriptions

Many of the terminal elements are constrained in size and/or content as listed below. Character entities count as one character in size constraints. Numeric values may be provided in decimal or hexadecimal (when preceded by "0x", or "x"). Text constants are case sensitive.

b
Bold face type: Specifies that text between tags is in boldface type.

canClose
Indicates whether an NCS can close the window in which it is hosting an ActiveX control when the control sends an ncsReqAppClose message. Permitted values are TRUE and FALSE.

changed
Changed Time/Date: Time the object was last changed in the MOS. Format is YYYY-MM-DD'T'hh:mm:ss[,ddd]['Z'], e.g. 1999-04-11T14:22:07,125Z or 1999-04-11T14:22:07,125-05:00. Parameters displayed within brackets are optional. [,ddd] represents fractional time in which all three digits must be present. [‘Z’] indicates time zone which can be expressed as an offset from UTC in hours and minutes. Optionally, the time zone may be replaced by the character ‘Z’ to indicate UTC.

changedBy
Last Changed by: Name of the person or process that last changed the object in the MOS. This can be stored in a language other than English.

command
roItemCtrl command: The commands READY, EXECUTE, PAUSE and STOP, as well as general indicator, SIGNAL, can be addressed at each MOS Structure level. In other words, a single command can begin EXECUTION of an entire Running Order, of a Story containing multiple Items, or of a single Item.

created
Creation Time/Date: Time the object was created in the MOS. Format is YYYY-MM-DD'T'hh:mm:ss[,ddd]['Z'], e.g. 1999-04-11T14:22:07,125Z or 1999-04-11T14:22:07,125-05:00. Parameters displayed within brackets are optional. [,ddd] represents fractional time in which all three digits must be present. [‘Z’] indicates time zone which can be expressed as an offset from UTC in hours and minutes. Optionally, the time zone may be replaced by the character ‘Z’ to indicate UTC.

createdBy

Created by: Name of the person or process that created the object in the MOS. This can be stored in a language other than English. 128 chars max.

description

Object Description: Text description of the MOS object. No maximum Length is defined. This can be stored in a language other than English.

DOM

Date of Manufacture.

em

Emphasized Text: markup within description and p to emphasize text.

endx

Used in MOS ActiveX messages. The maximum width in pixels that the NCS Host allows for an ActiveX Plug-In in a particular metric in ncsAppInfo. 0XFFFFFFFF max.

endy

Used in MOS ActiveX messages. The maximum height in pixels that the NCS Host allows for an ActiveX Plug-In in a particular metric in ncsAppInfo. 0XFFFFFFFF max.

generalSearch

String used for simple searching in the mosReqObjList message. Logical operators are allowed. 128 chars max.

hwRev

HW Revision: 128 chars max.

I

Italics: Specifies that text between tags is in Italics.

ID

Identification of a Machine: text. 128 chars max.

item

Item: Container for item information within a Running Order message.

itemChannel

Item Channel: Channel requested by the NCS for MOS to playback a running order item. 128 chars max.

itemEdDur

Item Editorial Duration: in number of samples 0XFFFFFFFF max.

itemEdStart

Editorial Start: in number of samples 0XFFFFFFFF max.

itemID

Item ID: Defined by NCS, UID not required. 128 chars max.

itemSlug

Item Slug: Defined by NCS. 128 chars max

itemTrigger

Item Air Trigger: "MANUAL", "TIMED" or "CHAINED".
CHAINED (sign +/-) (value in # of samples)
CHAINED -10 would start the specified clip 10 samples before the proceeding clip ended. CHAINED 10 would start the specified clip 10 samples after the preceding clip ended, thus making a pause of 10 samples between the clips. There is a space character between the word CHAINED and the value.

itemUserTimingDur

Item User Timing Duration: If the NCS extracts a duration value from a MOS item for show timing, and this field has a value, then the NCS must use this value. The value is in number of samples. 0XFFFFFFFF max.

listReturnEnd

Integer value used in mosReqObjList commands to specify the index of the last mosObj message

requested or returned in a message. 0xFFFFFFFF max.

listReturnStart
Integer value used in mosReqObjList commands to specify the index of the first mosObj message requested or returned in a message. 0xFFFFFFFF max.

listReturnStatus
Optional string value in the mosObjList message that specifies the reason for a zero value in the <listReturnTotal> tag. 128 chars max.

listReturnTotal
Integer value in the mosObjList message specifying how many mosObj messages are returned. 0xFFFFFFFF max.

macroIn
Macro Transition In: Defined by MOS. 128 chars max.

macroOut
Macro Transition Out: Defined by MOS. 128 chars max.

manufacturer
Used in MOS ActiveX messages. Manufacturer: Text description. 128 chars max.

messageID
Unique identifier sent with requests. See chapter 4.1.6 for a detailed description.
Format: signed integer 32-bit, value above or equal to 1, decimal or hexadecimal.
An empty messageID tag is allowed for messages when used in the ActiveX interface.

mode
Used in MOS ActiveX messages. How the ActiveX Plug-In window appears in the NCS Host window: MODALDIALOG, MODELESS, CONTAINED, TOOLBAR.

model
Model: Text description. 128 chars max.

modifiedBy
Modified by: Name of the person or process that last modified the object in the MOS. This can be stored in a language other than English. 128 chars max.

mosAbstract
Abstract of the Object intended for display by the NCS. This field may contain HTML and DHTML markup. The specific contents are limited by the NCS vendor's implementation. Length is unlimited but reasonable use is suggested.

mosActiveXversion
Used in MOS ActiveX messages. String indicating the version of the ActiveX Plug-In. 128 chars max

mosID
MOS ID: Character name for the MOS unique within a particular installation.

mosGroup
This field is intended to imply the name of a destination, group or folder for the Object pointer to be stored in the NCS. 128 chars max.

mosMsg data type

Used in MOS ActiveX messages. Clipboard format used for OLE drag and drop from the ActiveX Plug-In.

mosPlugInID

Used in MOS ActiveX messages. ID that the NCS Host can use to instantiate the ActiveX Plug-In. 128 chars max.

mosRev
MOS Revision: Text description. 128 chars max.

mosScope
This field implies the extent to which the mosExternalMetadata block will move through the NCS workflow. Accepted values are "OBJECT" "STORY" and "PLAYLIST"

ncsID
NCS ID: Character name for the NCS unique within a particular installation. 128 chars max.

objAir
Air Status: "READY" or "NOT READY".

objDur
Object Duration: The number of samples contained in the object. For Still Stores this would be 1. 0XFFFFFFFF MAX

objID
Object UID: Unique ID generated by the MOS and assigned to this object. 128 chars max.

objRev
Object Revision Number: 999 max.

objSlug
Object Slug: Textual object description. 128 chars max.

objTB
Object Time Base: Describes the sampling rate of the object in samples per second. For still stores this is 0. For PAL Video this would be 50. For NTSC it would be 60. For audio it would reflect the audio sampling rate. Object Time Base is used by the NCS to derive duration and other timing information. 0XFFFFFFFF MAX

objType
Object Type: Choices are "STILL", "AUDIO", "VIDEO".

opTime
Operational Time: date and time of last machine start. Format is YYYY-MM-DD'T'hh:mm:ss[,ddd]['Z'], e.g. 1999-04-11T14:22:07,125Z or 1999-04-11T14:22:07,125-05:00. Parameters displayed within brackets are optional. [,ddd] represents fractional time in which all three digits must be present. ['Z'] indicates time zone which can be expressed as an offset from UTC in hours and minutes. Optionally, the time zone may be replaced by the character 'Z' to indicate UTC.

p
Paragraph: Standard html delimitation for a new paragraph.

pause
Item Delay: Requested delay between items in ms 0XFFFFFFFF MAX.

pi
Presenter instructions: Instructions to the anchor or presenter that are not to be read such as "Turn to 2-shot."

pkg
Package: Specifies that text is verbatim package copy as opposed to copy to be read by presenter.

product
Used in MOS ActiveX messages. String indicating the product name of the NCS Host. 128 chars max.

queryID
Unique identifier used in mosReqObjList and mosObjList to allow the MOS to do cached searching. 128 chars max.

roAir
Air Ready Flag: "READY" or "NOT READY".

roChannel
Running Order Channel: default channel requested by the NCS for MOS to playback a running order. 128 chars max.

roCtrlCmd

Running Order Control Command: READY, EXECUTE, PAUSE and STOP, as well as general

indicator, SIGNAL, can be addressed at each level. In other words, a single command can begin EXECUTION of an entire Running Order, of a Story containing multiple Items, or of a single Item.

roCtrlTime

Running Order Control Time: roCtrlTime is an optional field which provides a mechanism to time stamp the time of message transmission, or optionally, to provide a time in the immediate future at which the MOS should execute the command. Format is YYYY-MM-DD'T'hh:mm:ss[,ddd]['Z'], e.g. 1999-04-11T14:22:07,125Z or 1999-04-11T14:22:07,125-05:00. Parameters displayed within brackets are optional. [,ddd] represents fractional time in which all three digits must be present. ['Z'] indicates time zone which can be expressed as an offset from UTC in hours and minutes. Optionally, the time zone may be replaced by the character 'Z' to indicate UTC.

roEdDur

Running Order Editorial Duration: duration of entire running order. Format in hh:mm:ss, e.g. 00:58:25.

roEdStart

Running Order Editorial Start: date and time requested by NCS for MOS to start playback of a running order. Format is YYYY-MM-DD'T'hh:mm:ss[,ddd]['Z'], e.g. 1999-04-11T14:22:07,125Z or 1999-04-11T14:22:07,125-05:00. Parameters displayed within brackets are optional. [,ddd] represents fractional time in which all three digits must be present. ['Z'] indicates time zone which can be expressed as an offset from UTC in hours and minutes. Optionally, the time zone may be replaced by the character 'Z' to indicate UTC.

roEventTime

Running Order Event Time: Time of the time cue sent to the parent MOS by the NCS for a specific item event.

roEventType

Running Order Event Type: The type of event that is being queued in the Running order.

roID

Running Order UID: Unique Identifier defined by NCS. 128 chars max.

roSlug

Running Order Slug: Textual Running Order description. 128 chars max.

roStatus

Running Order Status: Options are: "OK" or error description. 128 chars max.

roTrigger

Running Order Air Trigger: "MANUAL", "TIMED" or "CHAINED".
CHAINED (sign +/-) (value in # of samples)
CHAINED -10 would start the specified clip 10 samples before the proceeding clip ended. CHAINED 10 would start the specified clip 10 samples after the preceding clip ended, thus making a pause of 10 samples between the clips. There is a space character between the word CHAINED and the value.slug Textual Object ID: This is the text slug of the object and is stored in the native language. This can be stored in a language other than English. 128 chars max.

runContext

Used in MOS ActiveX messages. Specifies the context in which the NCS Host is instantiating the ActiveX Plug-In: BROWSE, EDIT, CREATE.

SN

Serial Number: text serial number. 128 chars max.

startx

Used in MOS ActiveX messages. The minimum width in pixels that the NCS Host allows for an ActiveX Plug-In in a particular metric in ncsAppInfo. 0XFFFFFFFF max.

starty

Used in MOS ActiveX messages. The minimum height in pixels that the NCS Host allows for an ActiveX Plug-In in a particular metric in ncsAppInfo. 0XFFFFFFFF max.

status

Status: Options are "NEW" "UPDATED" "MOVED" "BUSY " "DELETED", "NCS CTRL", "MANUAL CTRL", "READY", "NOT READY", "PLAY," "STOP".

statusDescription

Status Description: textual description of status. 128 chars max.

statusDescription

String describing the error in an ncsAck message that contains a status of ERROR. 128 chars max.

story

Story: Container for story information in a Running Order message.

storyBody

Story Body: The actual text of the story within a running order.

storyID

Story UID: Defined by the NCS. 128 chars max.

storyItem

Story Item: An item imbedded into a story that can be triggered when that point in the story is reached in the teleprompter.

storyNum

Story Number: The name or number of the Story as used in the NCS. This is an optional field originally intended for use by prompters. 128 chars max.

storyPresenter

Story Presenter: The anchor or presenter of a story within an running order.

storyPresenterRR

Story Presenter Read Rate: The read rate of the anchor or presenter of a story within a running order.

storySlug

Story Slug: Textual Story description. 128 chars max.

swRev

Software Revision: (MOS) Text description. 128 chars max.

tab

Tab: tabulation markup within description and p.

time

Time: Time object changed status. Format is YYYY-MM-DD'T'hh:mm:ss[,ddd]['Z'], e.g. 1999-04-11T14:22:07,125Z or 1999-04-11T14:22:07,125-05:00. Parameters displayed within brackets are optional. [,ddd] represents fractional time in which all three digits must be present. [‘Z’] indicates time zone which can be expressed as an offset from UTC in hours and minutes. Optionally, the time zone may be replaced by the character ‘Z’ to indicate UTC.

u

Underline: Specifies that text between tags is to be underlined.

version

Used in MOS ActiveX messages. String indicating the version of the NCS Host. 128 chars max.

7. Recent Changes

7.1 Changes from MOS version 2.8 to 2.8.1

1. A new family of messages has been added to allow the NCS to query the MOS for object metadata meeting certain criteria. The mosReqObjList message supports a subset of the XPath query language for mosObj message retrieval. They have been added to Profile 3 – Advanced Object-Based Workflow.
2. A new messageID element has been added to all the MOS device messages to support intelligent retry if a connection times out before a response to a message has been received. Three ActiveX messages also had the messageID tag added:
   ncsAppInfo
   mosItemReplace
   roStorySend
   The ActiveX messages do not require a value for the tag, as the communication happens within a single process.
3. Added a new ActiveX message ncsReqAppClose. A hosted ActiveX Plugin can send this message to the NCS Host when it wants to shut itself down.
4. The roElementAction message description has been rewritten to make its use clearer. More examples have also been added. The message syntax and functionality have not changed.
5. All messages are now included in the DTD.
6. All the example messages have been validated by the DTD.

7.2 Changes from MOS version 2.6 to 2.8

The following command messages have been added:

1) A single preferred command message, roElementAction, is now available for manipulation of story and item sequences. This command message can effectively replace all existing use of roStory and roItem commands.

2) Five additional commands were added as alternates to roElementAction to ease transition and provide backward compatibility. These commands are allowed in MOS v2.8 but will be phased out at some point in the future

   a. roItemInsert
   b. roItemReplace
   c. roItemMoveMultiple
   d. roItemDelete
   e. roStoryMoveMultiple – allows multiple stories to be moved in a single command.

   Item level commands are analogous to the story-level commands with similar names.

   The item-level Append and Swap commands can be performed by using roItemInsert and roItemMoveMultiple, respectively, and provide no extra performance gain. Hence roItemAppend and roItemSwap are not needed.

   Note: these commands are used to manage items within a single story; these cannot be used to copy or move items between stories.

The following command messages have been changed:

1) mosReqAll - command has reversed the meaning of the <pause> tag, so that a pause value greater than zero means that individual mosObj messages are sent by the MOS. A pause value of zero means that a single mosListAll message is sent by the MOS.

2) mosListAll - command now contains <mosObj> tags, instead of all the object fields appearing as direct tags inside <mosListAll>.

3) roListAll - command now contains <ro> tags, instead of all the ro fields appearing as direct tags inside roListAll.

4) listMachInfo - command contains additional tags to define the MOS profiles supported by the MOS, and optional ActiveX configuration information.

7.3 Changes to MOS version 2.6 WD-2001-08-09

1. Removed second reference to mosExternalMetadata in roStorySend description
2. DTD reworked with suggestions from Jiri Basek Aveco s.r.o (Jiri.Basek@aveco.com)
3. DTD also posted to web site at http://www.mosprotocol.com/mos2dot601.dtd

7.4 Changes from MOS version 2.5 to 2.6 WD-2001-06-06

1. Added message for mosItemReplace.
2. Added message for roMetadataReplace.
3. Added message for roStoryMove.
4. Added structure for mosExternalMetadata.
5. mosExternalMetadata structure added to many common elements
6. DTD was restructured to include new and changed elements. The sequence of definitions was also changed.

7.5 Changes from MOS version 2.0 to 2.5 WD-2000-08-01

7. Added tag for mosObjCreate.
8. Added tag for roStorySend.
9. Added tag for roItemCue.
10. Added tag for roCtrl.
11. Clarified Definition of roList and *roReq*
12. Fixed errors and syntax errors in documentation.
13. Modified Item structure. Description is as follows:

Modified Item Structure

Purpose

Allows for identification of the parent Media Object Server when the play list is forcefully constructed on a machine other than an object's parent Media Object Server, i.e. when a play list is constructed in a MOS Prompter.

Description

The format of the message remains the same. The only change is the optional addition of the

existing mosID tag. The value of this tag, if appropriately used, is set to the ID of the referenced object's parent Media Object Serve. This change is reflected throughout all MOS messages which include the <item> structure.

Structural Outline

- item
  - itemID
  - itemSlug
  - mosID
  - objID
  - itemChannel
  - itemEdStart
  - itemEdDur
  - itemTrigger
  - macroIn
  - macroOut

7.6 Changes for MOS 2.0 WD-1999-05-12

1. Restructured Running Order messages so that running orders contain "stories", which contain "items".
2. Added optional channel assignment tag "itemChannel" following objID in item tag in Running Order messages.
3. Added story tag "story" to Running Order messages to contain storyID and Items.
4. Changed the name of message types so that messages that are exclusive to the Media Object Metadata socket start with "mos" and messages exclusive to the Running Order socket start with "ro".
5. Eliminated Running Order item messages. The same effect can be achieved using Running Order story messages.
6. Changed item slug tag from "slug" to "itemSlug" and added optional "storySlug" tag following story tag.
7. Added message "mosReqObj" to allow NCS to request MOS object information for a specific object by objID. Response is "mosObj".
8. Added optional tags for Running Order channel, start, duration and trigger (roChannel, roEdStart, roEdDur, roTrigger). Renamed item tags (itemChannel, itemEdStart, itemEdDur, itemTrigger).
9. Added message "roReqAll" for MOS to request list of all known running orders from an NCS. Response is "roListAll" message.
10. Changed "metadata" tag to "statusDescription" in mosAck and to "description" in mosObj message.
11. Added itemID to "roItemStat" message.

7.7 Changes from MOS version 1.52 to 2.0 WD-1999-03-17

1. Changed tags to XML format.
2. Added root tag "mos".
3. Removed space in tags.
4. Change MOS ID and NCS ID to always be in the same sequence.
5. Tags must be used in the sequence defined.
6. Changed capitalization of tags for readability.
7. Added optional formatting to metadata.

8. MOS 2.8.1 DTD

```
<!-- MOS.DTD version 2.8.1 June 3, 2004-->
<!--Thanks to Jiri Basek Aveco s.r.o (Jiri.Basek@aveco.com) and the Octopus Team for the following list of modifications due to original DTD bugs :
      1:  element "roStorySend" : comma added between "mosExternalMetadata*" and "storyBody"
      2:  element "roStorySend" old version : deactivated
      3:  element "item" : ending bracket added after "mosExternalMetadata*"
      4:  element "command" : "READY", "EXECUTE", "PAUSE", "STOP", "SIGNAL" are not elements but predefined strings
      5:  element "mosObj" : "externalData" changed to "mosExternalData"
      6:  ATTLIST metadata : deactivated. ? should it be changed to ATTLIST mosExternalMetadata ?
      8:  element "mosScope" : "object, "story", "playlist" are not elements but predefined strings
      A9: element "mos" : "reqMachineInfo" changed to "reqMachInfo"
      A10:  element "storyNum" defined

   Additional changes have been made to the formatting and sequence of the elements in order to better match the order of structures presented in the MOS
    Protocol document.

-->
<!-- MOS Message - One message type per message -->
<!ELEMENT mos ((ncsAck | ncsReqAppInfo | ncsReqAppMode | ncsStoryRequest | ncsItemRequest | ncsItem | ncsReqAppClose | (mosID, ncsID,
 messageID, (mosAck | mosObj | mosReqObj | mosReqAll | mosListAll | mosObjCreate | mosItemReplace | ncsAppInfo | roAck | roCreate | roReplace |
 roMetadataReplace | roDelete | roReq | roList | roReqAll | roListAll | roStat | roReadyToAir | roStoryAppend | roStoryInsert | roStoryReplace | roStoryMove |
 roStorySwap | roStoryDelete | roStoryMoveMultiple | roItemInsert | roItemReplace | roItemMoveMultiple | roItemDelete | roElementAction | roItemStat |
 roItemCue | roCtrl | roStorySend | heartbeat | reqMachInfo | listMachInfo | mosReqSearchableSchema | mosListSearchableSchema | mosReqObjList |
 mosObjList))))>
<!ELEMENT mosAck (objID, objRev, status, statusDescription)>
<!ELEMENT mosObj (objID, objSlug, mosAbstract?, objGroup?, objType, objTB, objRev, objDur, status, objAir, createdBy, created, changedBy, changed,
 description, mosExternalMetadata*)>
<!ELEMENT mosReqObj (objID)>
<!ELEMENT mosReqAll (pause)>
<!ELEMENT mosListAll (mosObj*)>
<!ELEMENT mosReqSearchableSchema EMPTY>
<!ELEMENT mosListSearchableSchema (mosSchema)>
<!ELEMENT mosReqObjList (queryID, listReturnStart, listReturnEnd, generalSearch, mosSchema, searchGroup*)>
<!ELEMENT searchGroup (searchField+)>
<!ELEMENT searchField EMPTY>
<!ATTLIST searchField
            XPath CDATA #REQUIRED
            sortByOrder CDATA #IMPLIED
            sortType CDATA #IMPLIED
>
<!ELEMENT mosObjList (queryID, listReturnStart, listReturnEnd, listReturnTotal, listReturnStatus?, list?)>
<!ELEMENT list (mosObj+)>
<!ELEMENT generalSearch (#PCDATA)>
<!ELEMENT listReturnStart (#PCDATA)>
<!ELEMENT listReturnEnd (#PCDATA)>
<!ELEMENT listReturnTotal (#PCDATA)>
<!ELEMENT listReturnStatus (#PCDATA)>
<!ELEMENT queryID (#PCDATA)>
<!ELEMENT mosObjCreate (objSlug, objGroup?, objType, objTB, objDur?, time?, createdBy?, description?, mosExternalMetadata*)>
<!ELEMENT mosItemReplace (roID, storyID, item)>
<!ELEMENT ncsAck (status, statusDescription?)>
<!ELEMENT ncsAppInfo (mosObj?, roID?, storyID?, item?, ncsInformation)>
<!ELEMENT roAck (roID, roStatus, (storyID, itemID, objID, status)*)>
<!ELEMENT roCreate (roID, roSlug, roChannel?, roEdStart?, roEdDur?, roTrigger?, macroIn?, macroOut?, mosExternalMetadata*, story*)>
<!ELEMENT roReplace (roID, roSlug, roChannel?, roEdStart?, roEdDur?, roTrigger?, macroIn?, macroOut?, mosExternalMetadata*, story*)>
<!ELEMENT roMetadataReplace (roID, roSlug, roChannel?, roEdStart?, roEdDur?, roTrigger?, mosExternalMetadata?)>
<!ELEMENT roDelete (roID)>
<!ELEMENT roReq (roID)>
<!ELEMENT roList (roID, roSlug, roChannel?, roEdStart?, roEdDur?, roTrigger?, macroIn?, macroOut?, mosExternalMetadata*, story*)>
<!ELEMENT roListAll (ro*)>
<!ELEMENT ro (roID, roSlug?, roChannel?, roEdStart?, roEdDur?, roTrigger?, mosExternalMetadata*)>
<!ELEMENT roStat (roID, status, time)>
<!ELEMENT roReadyToAir (roID, roAir)>
<!ELEMENT roStoryAppend (roID, story+)>
<!ELEMENT roStoryInsert (roID, storyID, story+)>
```

```
<!ELEMENT roStoryReplace (roID, storyID, story+)>
<!ELEMENT roStoryMove (roID, storyID, storyID)>
<!ELEMENT roStorySwap (roID, storyID, storyID)>
<!ELEMENT roStoryDelete (roID, storyID+)>
<!ELEMENT roStoryMoveMultiple (roID, storyID+ )>
<!ELEMENT roItemInsert (roID, storyID, itemID, item+)>
<!ELEMENT roItemReplace (roID, storyID, itemID, item+)>
<!ELEMENT roItemMoveMultiple (roID, storyID, itemID+ )>
<!ELEMENT roItemDelete (roID, storyID, itemID+ )>
<!ELEMENT roElementAction (roID, element_target?, element_source)>
<!ELEMENT element_target  (storyID, itemID?)>
<!ELEMENT element_source  (story+ | item+ | storyID+ | itemID+)>
 <!ATTLIST roElementAction operation CDATA #REQUIRED>
<!ELEMENT roStorySend (roID, storyID, storySlug?, storyNum?, storyBody, mosExternalMetadata*)>
<!ELEMENT roItemStat (roID, storyID, itemID, objID, status, time)>
<!ELEMENT roItemCue (mosID, roID, storyID, itemID, roEventType, roEventTime, mosExternalMetadata*)>
<!ELEMENT roCtrl (roID, storyID, itemID, command, mosExternalMetadata*)>
<!ELEMENT heartbeat (time)>
<!ELEMENT listMachInfo (manufacturer, model, hwRev, swRev, DOM, SN, ID, time, opTime?, mosRev, mosProfile0, mosProfile1, mosProfile2, mosProfile3,
 mosProfile4, mosProfile5, mosProfile6, defaultActiveX*, mosExternalMetadata*)>
<!ELEMENT defaultActiveX (mode, controlFileLocation, controlSlug, controlName, controlDefaultParams)>
<!ELEMENT story (storyID, storySlug?, storyNum?, mosExternalMetadata*, item*)>
<!ELEMENT description (#PCDATA | p | em | tab)*>
<!ELEMENT storyBody ((storyPresenter*, storyPresenterRR*, p*, storyItem*)*)>
<!ELEMENT storyItem (itemID, itemSlug?, objID, mosID, mosAbstract?, itemChannel?, itemEdStart?, itemEdDur?, itemUserTimingDur?, itemTrigger?,
 macroIn?, macroOut?, mosExternalMetadata*)>
<!ELEMENT item (itemID, itemSlug?, objID, mosID, mosAbstract?, itemChannel?, itemEdStart?, itemEdDur?, itemUserTimingDur?, itemTrigger?, macroIn?,
 macroOut?, mosExternalMetadata*)>
<!ELEMENT mosExternalMetadata (mosScope?, mosSchema, mosPayload)>
<!ELEMENT ncsInformation (userID, runContext, software, UImetric*)>
<!ELEMENT UImetric ((startx, starty, endx, endy, mode, canClose?)?)>
<!ATTLIST UImetric num CDATA #REQUIRED>
<!ELEMENT ncsItem (item)>
<!ELEMENT ncsReqAppMode (UImetric)>
<!ELEMENT software (manufacturer, product, version)>
<!ELEMENT b (#PCDATA | i | u)*>
<!ELEMENT i (#PCDATA | b | u)*>
<!ELEMENT p (#PCDATA | em | tab | pi | pkg | b | i | u)*>
<!ELEMENT pi (#PCDATA | b | i | u)*>
<!ELEMENT pkg (#PCDATA | b | i | u)*>
<!ELEMENT u (#PCDATA | b | i)*>
<!ELEMENT mosID (#PCDATA)>
<!ELEMENT ncsID (#PCDATA)>
<!ELEMENT canClose (#PCDATA)>
<!ELEMENT changed (#PCDATA)>
<!ELEMENT changedBy (#PCDATA)>
<!ELEMENT command (#PCDATA)>
<!-- valid values for "command" are "READY", "EXECUTE", "PAUSE", "STOP", "SIGNAL" -->
<!ELEMENT controlDefaultParams (#PCDATA)>
<!ELEMENT controlFileLocation (#PCDATA)>
<!ELEMENT controlName (#PCDATA)>
<!ELEMENT controlSlug (#PCDATA)>
<!ELEMENT created (#PCDATA)>
<!ELEMENT createdBy (#PCDATA)>
<!ELEMENT DOM (#PCDATA)>
<!ELEMENT em (#PCDATA)>
<!ELEMENT endx (#PCDATA)>
<!ELEMENT endy (#PCDATA)>
<!ELEMENT hwRev (#PCDATA)>
<!ELEMENT ID (#PCDATA)>
<!ELEMENT itemChannel (#PCDATA)>
<!ELEMENT itemEdDur (#PCDATA)>
<!ELEMENT itemEdStart (#PCDATA)>
<!ELEMENT itemID (#PCDATA)>
<!ELEMENT itemSlug (#PCDATA)>
<!ELEMENT itemTrigger (#PCDATA)>
<!ELEMENT itemUserTimingDur (#PCDATA)>
<!ELEMENT macroIn (#PCDATA)>
<!ELEMENT macroOut (#PCDATA)>
```

```
<!ELEMENT manufacturer (#PCDATA)>
<!ELEMENT mode (#PCDATA)>
<!ELEMENT model (#PCDATA)>
<!ELEMENT mosAbstract ANY>
<!ELEMENT mosPayload ANY>
<!ELEMENT mosProfile0 (#PCDATA)>
<!ELEMENT mosProfile1 (#PCDATA)>
<!ELEMENT mosProfile2 (#PCDATA)>
<!ELEMENT mosProfile3 (#PCDATA)>
<!ELEMENT mosProfile4 (#PCDATA)>
<!ELEMENT mosProfile5 (#PCDATA)>
<!ELEMENT mosProfile6 (#PCDATA)>
<!ELEMENT mosSchema (#PCDATA)>
<!ELEMENT mosScope (#PCDATA)>
<!-- valid values for "mosScope" are "OBJECT", "STORY", "PLAYLIST" -->
<!ELEMENT mosRev (#PCDATA)>
<!ELEMENT ncsItemRequest EMPTY>
<!ELEMENT ncsReqAppClose EMPTY>
<!ELEMENT ncsReqAppInfo EMPTY>
<!ELEMENT ncsStoryRequest EMPTY>
<!ELEMENT objAir (#PCDATA)>
<!ELEMENT objDur (#PCDATA)>
<!ELEMENT objGroup (#PCDATA)>
<!ELEMENT objID (#PCDATA)>
<!ELEMENT objRev (#PCDATA)>
<!ELEMENT objSlug (#PCDATA)>
<!ELEMENT objTB (#PCDATA)>
<!ELEMENT objType (#PCDATA)>
<!ELEMENT opTime (#PCDATA)>
<!ELEMENT pause (#PCDATA)>
<!ELEMENT product (#PCDATA)>
<!ELEMENT messageID (#PCDATA)>
<!ELEMENT reqMachInfo EMPTY>
<!ELEMENT roAir (#PCDATA)>
<!ELEMENT roID (#PCDATA)>
<!ELEMENT roChannel (#PCDATA)>
<!ELEMENT roCtrlCmd (#PCDATA)>
<!ELEMENT roCtrlTime (#PCDATA)>
<!ELEMENT roEdDur (#PCDATA)>
<!ELEMENT roEdStart (#PCDATA)>
<!ELEMENT roEventTime (#PCDATA)>
<!ELEMENT roEventType (#PCDATA)>
<!ELEMENT roReqAll EMPTY>
<!ELEMENT roSlug (#PCDATA)>
<!ELEMENT roStatus (#PCDATA)>
<!ELEMENT roTrigger (#PCDATA)>
<!ELEMENT runContext (#PCDATA)>
<!ELEMENT SN (#PCDATA)>
<!ELEMENT startx (#PCDATA)>
<!ELEMENT starty (#PCDATA)>
<!ELEMENT status (#PCDATA)>
<!ELEMENT statusDescription (#PCDATA)>
<!ELEMENT storyID (#PCDATA)>
<!ELEMENT storyNum (#PCDATA)>
<!ELEMENT storyPresenter (#PCDATA)>
<!ELEMENT storyPresenterRR (#PCDATA)>
<!ELEMENT storySlug (#PCDATA)>
<!ELEMENT swRev (#PCDATA)>
<!ELEMENT tab (#PCDATA)>
<!ELEMENT time (#PCDATA)>
<!ELEMENT userID (#PCDATA)>
<!ELEMENT version (#PCDATA)>
<!-- Attributes -->
<!ATTLIST mos
      version CDATA #FIXED "-//MOS Group//DTD MOS 2.8.1//EN"
      changeDate CDATA #FIXED "09 April 2004"
>
<!-- <!ATTLIST metadata xml:space (default | preserve) 'preserve'>  -->
```

9. References and Resources

9.1

The primary site for MOS protocol information is http://www.mosprotocol.com.

9.2 XML FAQ

http://www.ucc.ie/xml/ contains an extensive document of Frequently Asked Questions regarding XML.

9.3 Recommended Reading

Just XML: John E. Simpson; 1999. Prentice Hall PTR. ISBN 0-13-9434417-8. ($34.99)
XML for Dummies Quick Reference: Mariva H. Aviram; 1998. IDG Books Worldwide, Inc. ISBN 0-7645-0383-9. ($14.99)

The Unicode Standard, Version 2.0: The Unicode Consortium. Addison-Wesley. ISBN 0-201-48345-9. ($62.95)

9.4 XML Web Sites

The mission of XML.com is to help you discover XML and learn how this new Internet technology can solve real-world problems in information management and electronic commerce. http://www.xml.com/xml/pub/

Robin Cover's XML resource page is perhaps the most useful and extensive available on the Web. http://www.oasis-open.org/cover/xml.html