# MOS Agent



$Date: 2023-03-31 12:34:20 +0200 (Fri, 31 Mar 2023) $

---

**Table of Contents**



# Preface

Following guide describes the installation and configuration of Octopus MOS Agent.

# Chapter 1. Introduction

This guide describes how to install the Octopus MOS Agent, how to configure the service, what has to be configured in the Octopus client and basic troubleshooting.

MOSAgent exchanges data between Octopus and third-party systems (MOS devices) using the MOS protocol (see http://mosprotocol.com/). Rundowns and story folders flow from Octopus to MOS devices. MOS objects flow from MOS devices to Octopus. MOS protocol messages are in Unicode XML format and sent over TCP connections.

## 1.1. Network communication basics

### 1.1.1. Network connections

MOSAgent opens two independent TCP connections to the MOS device (unless their host and port fields are omitted in the MOS devices config):

- Connection to media port (a.k.a. lower port, default 10540) - it is mainly used to send requests to create/modify/delete MOS objects and receive MOS objects and their updates from the MOS device.
- Connection to rundown port (a.k.a. upper port, default 10541) - it is mainly used to send rundowns and stories to the MOS device and receive their statuses from the MOS device.

Ports are hooked to a specific messages. NRCS cannot send rundowns on the Media port and vice versa a MOS Device cannot send MOS objects on the rundown port.

MOSAgent also accepts TCP connections incoming from the MOS device on the "Octopus media port" and "Octopus rundown port" TCP ports in config.

MOSAgent keeps its TCP connections open until the MOS device stops responding or until the connection is reconfigured to another address.

### 1.1.2. Sending messages

In general, MOSAgent sends a MOS protocol message to the MOS device and then it waits for the MOS device to send back a particular response message. MOSAgent does not send any other requests on this TCP connection until the MOS device responds or until too much time elapses ("Response timeout" in config). Any other requests that MOSAgent would send through this connection are waiting in a queue and they will be sent after the MOS device responds. Waiting for a response is limited to the one TCP connection, the other TCP connections are not affected by it i.e. MOSAgent can still send requests through them.

If the MOS device does not respond in time then MOSAgent closes the TCP connection and empties the queue of waiting requests i.e. the requests are discarded and will not be sent after reconnection. This is not a problem for the flow of rundown from Octopus to the MOS device because MOSAgent resends them in full when it establishes a connection to the rundown port.

If this connection was opened from MOSAgent to the MOS device then MOSAgent attempts to reconnect and keeps doing so until it succeeds. Otherwise, the TCP connection is just closed and it is up to the MOS device to restore it, MOSAgent cannot do it (it is impossible).

## 1.2. MOS objects

MOS objects are media objects that can be aired by MOS devices - video clips, animations, pictures, sounds. When a MOS object is created (changed, deleted) at a MOS device then the MOS device is supposed to send a mosObj message with status NEW (UPDATED, DELETED). Each MOS object has a unique objID at the MOS device.

Octopus database contains two types of MOS objects

- Public MOS objects - They are visible in the Media section. They may be used by multiple stories at the same time. They are supposed to mirror the MOS objects in the MOS device database.
- Private MOS objects - They are NOT visible in the Media section. They may be used by one story only. They can be different from the MOS objects in the MOS device database, there can even be multiple private MOS objects with the same objID. They are used for archiving of historical data and for MOS objects that Octopus received from MOS device plugins.

### 1.2.1. Lowres

Lowres files are files that describe the contents of a MOS object in some way - a small part of the MOS object or the whole thing in lower quality. Paths to lowres files (URLs or UNC paths) are either sent to Octopus in <mosObj> messages in <objProxyPath> tags or they are deduced by Octopus from properties of MOS objects.

There are several types of lowres:

**Keyframes**

Still pictures from a video. One MOS object can have a number of keyframes. The first keyframe is used as a thumbnail to represent the whole video in Media.

**Lowres**

A low-quality version of a video.

**Mobile lowres**

Lowres that is supposed to be played on mobile devices.

### 1.2.2. Content sync

The database of MOS objects at the MOS device may not always be the same as the database of public MOS objects in Octopus because for example:

- this is the first time we are connected to the MOS device
- MOSAgent is not necessarily always connected to MOS device so it could miss some mosObj messages
- because of network outages, bugs etc.

You can make sure that Octopus contains the same public MOS objects as the MOS device by synchronizing the MOS object databases with the MOS device. Synchronization is started by the "Content sync" button in the MOS devices section. Also, the MOS device could initiate the synchronization instead of Octopus. Synchronization is only concerned with public MOS objects, it does not affect private MOS objects.

There are two methods of synchronization:

- MOSAgent sends mosReqAll to the MOS device requesting a single mosListAll message. This message contains all MOS objects stored in the MOS device. MOSAgent then does the following:
  - MOS objects which are not present in the Octopus database will be inserted there
  - MOS objects which are already present in the Octopus database will be overwritten - but only if their <changed> timestamp is more recent in the mosListAll message
  - MOS objects which are present in the Octopus database but not in the mosListAll message will be deleted from the Octopus database

  This is the only proper method of synchronization.
- MOSAgent sends mosReqAll to the MOS device requesting multiple mosObj messages - one mosObj message for each MOS object stored in the MOS device. MOSAgent then stores all these MOS objects in the Octopus database. But it cannot delete those MOS objects from the Octopus database which are no longer stored in the MOS device database because it does not know their list. Use this method of synchronization only if the first method does not work.

### 1.2.3. Deletion of MOS objects

MOS objects become candidates for deletion when a <mosObj> message with <status>DELETED</status> is received. If there is no non-archived story using this MOS object then the MOS object is deleted immediately. Otherwise, it is marked as ***DISCONNECTED***. The Media section does not show ***DISCONNECTED*** objects by default (see the option ***Show disconnected MOS objects*** in ***MOS*** tab in System Setup). If all the stories that were using the particular MOS object stop using it then the ***DISCONNECTED*** MOS object is deleted.

### 1.2.4. Creation of placeholders

Octopus can ask a MOS device to create a new MOS object with properties specified by Octopus. MOSAgent sends a mosObjCreate message to the MOS device which is then supposed to

- immediately reply with a mosAck message which contains objID of the newly created MOS object
- later send a mosObj message with the new MOS object (with the objID supplied in the previous step)

The Octopus dialog where users request the creation will not close until both steps are completed.

## 1.3. Sending of stories and their (sub)elements

The MOS protocol specifies how to send "Running orders" ("RO" in short) which are sequences of stories. MOSAgent can send both rundowns and story folders as ROs. It only sends ROs that are considered to be MOS-active. Rundowns can be activated manually by users or automatically according to their scheduled start, see ***Activation*** tab in the config of the particular MOS device. Automatic activation is preferred to manual activations because users often forget to deactivate rundowns that are no longer needed and this could cause performance problems. Story folders can only be activated individually. When a RO becomes active MOSAgent sends either roCreate or roReplace (according to ***Refresh method***). When a RO becomes inactive MOSAgent sends roDelete and if the RO is a rundown then

- it deletes MOS status of the rundown and its slugs if ***Accepts on-air status*** is enabled in config
- it marks the rundown as ***AIRED*** if ***Mark rundowns aired*** is set to ***When deactivated*** in System Setup.

When a connection to the rundown port of a MOS device is established (either for the first time or after MOSAgent reconnects) we need to send active ROs to the MOS device. There are two possible ways of doing that:

- either MOSAgent sends all active rundowns using either roCreate or roReplace (according to ***Refresh method***). This is how we work with most MOS devices.
- or MOSAgent does not send these messages (if ***Refresh method*** is set to ***None***) and it relies on the MOS device to take initiative and send roReqAll and roReq messages to MOSAgent. This is how we work with Astra.

When a user makes changes to stories in an active RO then MOSAgent notifies the MOS device about these changes using these messages: roStoryAppend, roStoryInsert, roStoryReplace, roStoryMove, roStoryDelete.

### 1.3.1. Visibility of stories

MOSAgent does not have to send all stories with all of their elements to the MOS device because it is not appropriate to send data which are not relevant for the MOS device. Story element is visible for the MOS device if

- it contains visible redirected MOS object inside (even in a script subelement)
- or it contains a CG and sending of QuickCGs or subelements is enabled
- or it is an element of basic type Insert and sending of Inserts is enabled

The story is visible for the MOS device if

- it is not a special story type like Segment, Text, Off-air
- and it is not skipped (or below the Off-air line) or sending of skipped stories is enabled
- and it contains a visible story element or sending of empty stories is enabled

Stories that are not visible are not sent to MOS devices and MOSAgent pretends that they are not present in the RO. If they become visible then they are sent to the MOS device. If they become invisible then they are deleted from the MOS device.

### 1.3.2. Prompting

### 1.3.3. Redirection of MOS objects

Redirection is a way of taking MOS objects that belong to one MOS device and making them visible to another but related MOS device. Those two MOS devices must have "similar" mosIDs which mean that the parts of their mosIDs before the first "." character are the same. For example VIZRT.PRIMARY and VIZRT.SECONDARY are considered to be similar.

Redirected version of a mosID is the mosID of another MOS device with similar mosID that has ***Device is the target of redirection*** enabled or that was chosen for redirection for the particular rundown in the rundown editing dialog or in the scheduler.

### 1.3.4. Alternative script versions

Alternative scripts can be sent to MOS devices in the form of virtual rundowns located in virtual channels. It means that a rundown with multiple versions will appear as multiple independent rundowns to the MOS device. If an alternative version of a story does not exist then its content is taken from the default version (this does not apply to stories of type Insert).

MOSAgent only sends those versions which were selected in BOTH the MOS device config AND in the rundown editing dialog or scheduler. Why does it have to be selected from two places ? Because you may want to have several MOS devices - one for each language - instead of one MOS device for all languages. Or because you may want to have rundowns that contain stories with multiple versions but you only want to broadcast some of them.

Selecting no alternatives is the same as selecting the Default alternative alone.

<roID> of these virtual rundowns have the short name of the alternative appended e.g. <roID>12345-USA</roID>

<roSlug> of these virtual rundowns are generated according to the pattern in the System Setup variable called "MOS alternative channel pattern". The pattern can have these variables: %CHANNEL, %SHORTNAME, %LONGNAME. For example "%CHANNEL %SHORTNAME" could be "AT USA" where "AT" is the name of the channel and "USA" is short name of the script alternative.

## 1.4. Status of story elements

MOS devices can send status information about story elements of stories that were sent to it by MOSAgent.

The statuses are sent by the MOS device using the <roItemStat> message.

The statuses say if the MOS object is ready to be aired or if there is something wrong with it or if it is being played out right now etc.

The list of statuses that Octopus should recognize is defined in the MOS devices section.

These statuses tell the users something about a MOS object in a particular location - as opposed to the location-independent status in the <objAir> tag in <mosObj> messages which is a different kind of status. The same MOS object often has different <roItemStat> statuses in different stories and rundowns.

MOSAgent ignores any <roItemStat> messages for ROs that are not MOS-active.

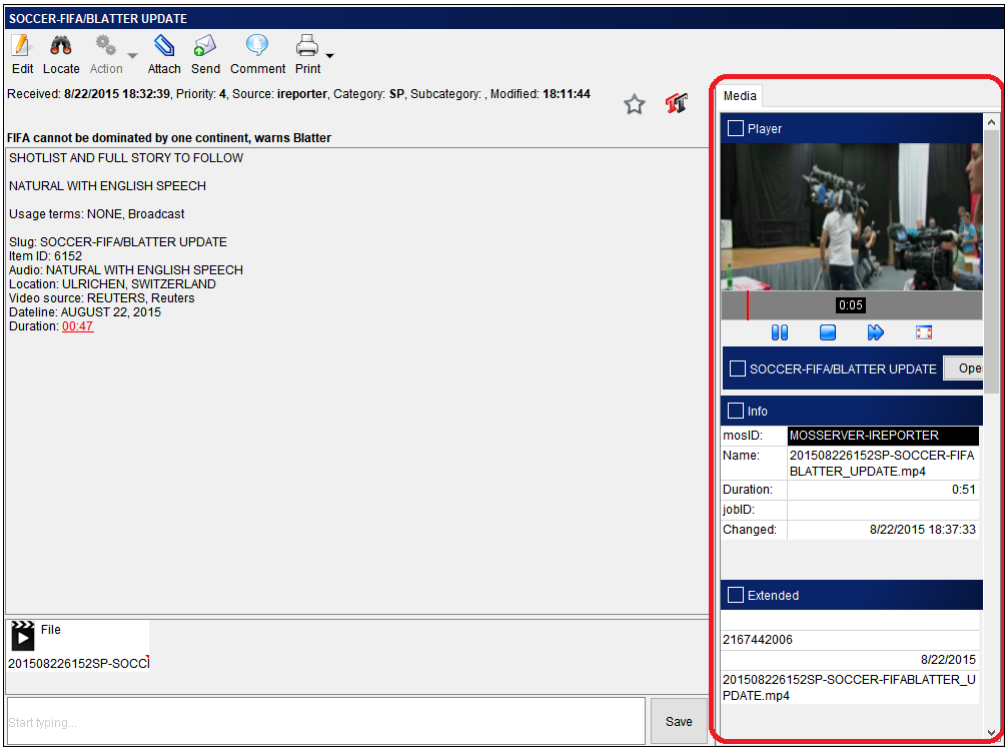
## 1.5. Link with wires

### 1.5.1. Basics

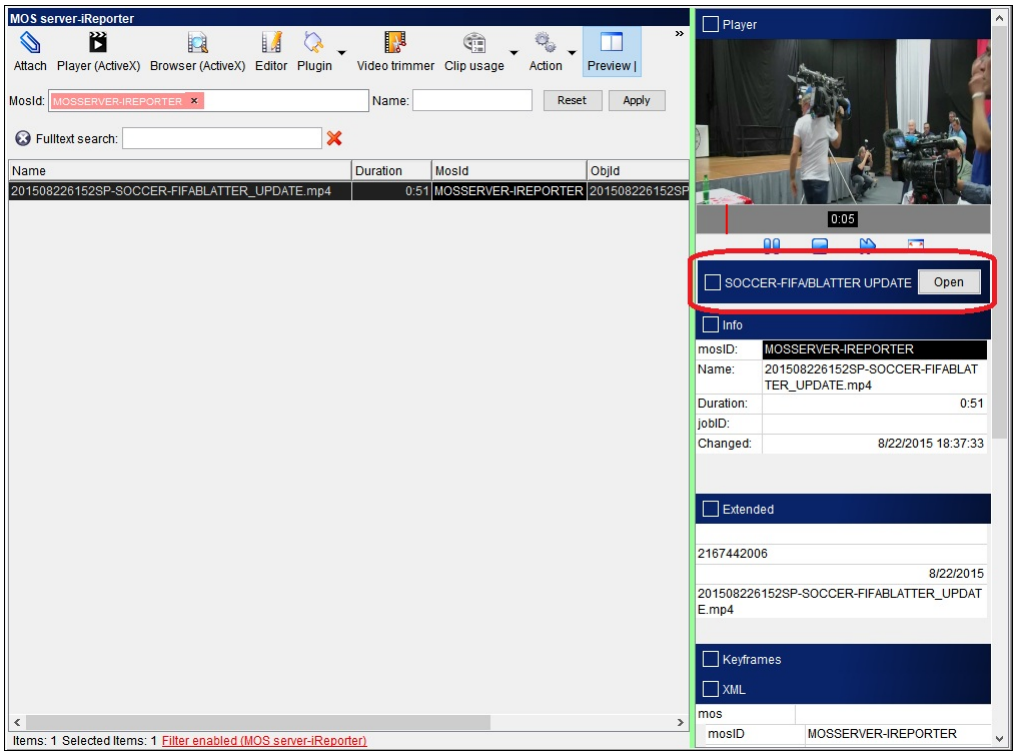
Octopus can link wires messages with related MOS objects (and vice versa). This might be helpful in case of ingested raw material, where there is important information about the footage in the related wire message. In order for this to work properly, there needs to be a unique identifier shared (GUID) with both wire message and MOS object. The link between the two is "lazy", meaning that it does not matter what comes first into the system but once there are two objects sharing this identifier they will be shown as related.

On top of automatic linking, it is also possible to link wire to media (or vice versa) by drag and drop (action link).

Although technically this link in 1: N (ie. there might be more wires per one MOS object), there is no limit restricting more MOS objects from having equal identifiers (more objects with different bitrate, for example).



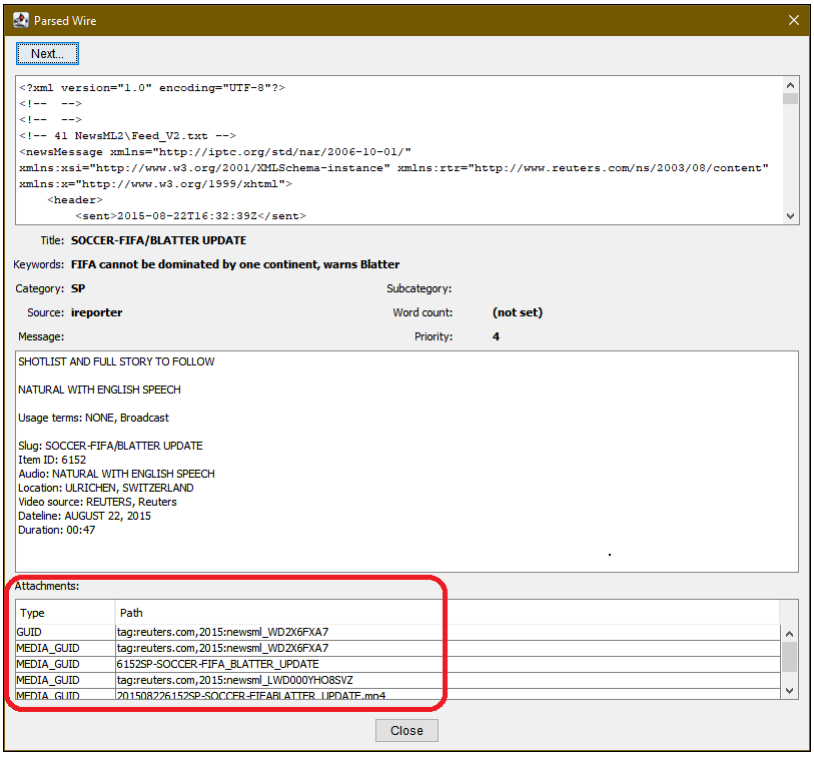

*Related MOS object shown in wire preview*




*Related wires shown in MOS object preview*

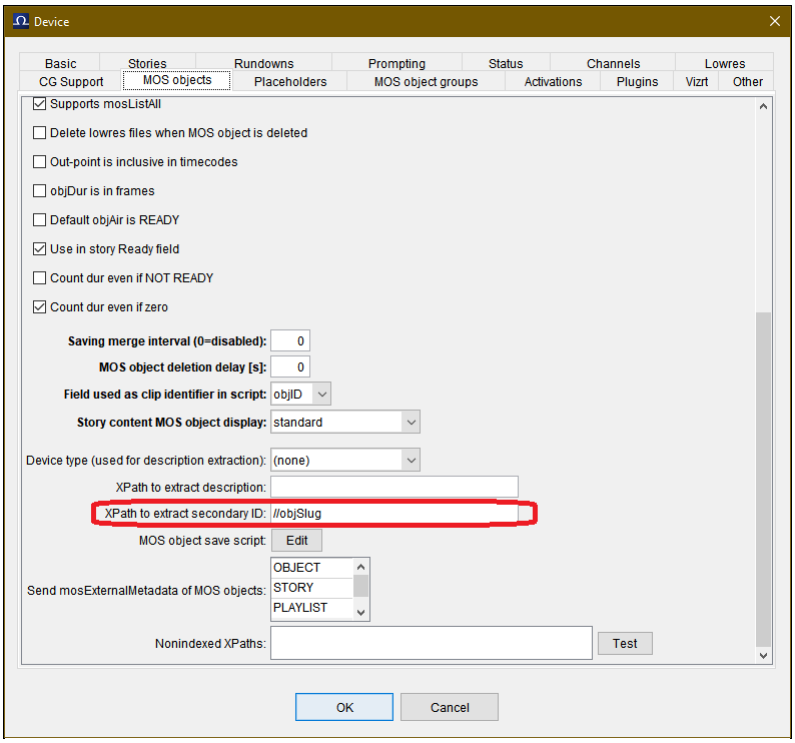
### 1.5.2. Configuration

As far as InjectAgent (the wires part) is concerned, this functionality is supported out-of-the-box by NewsML2 parser with Reuters agency. The agent is automatically creating wire attachments of type MEDIA_GUID from every news items guid attribute and every attached file name. The attachments have to have the guid specified in the XML, the parsed guid can be seen in the InjectAgent serverConnection.log when parsing the xml files or in the InjectAgent Testing dialogue:




*Parsed guid from the wire's XML*

On the other side (MOSAgent config part), there needs to be an XPath defining where this kind of identifier can be found in MOS XML. This option can be found in MOS device edit dialogue under ***MOS Objects*** tab and it is called ***XPath to extract secondary ID***.




*Secondary id configuration*

# Chapter 2. MOS Agent Installation

## 2.1. On Microsoft Windows

1. MOSAgent can be installed during the Octopus server installation or you can do it manually later. Copy the files MOSAgent.jar, MOSAgent.xml, MOSAgent.exe and the ***lib*** folder to $octopus$\services\MOSAgent-%MOSID%.

   There might be just one MOSAgent handling multiple MOS devices, or one MOSAgent per MOS device. First solution is less RAM consuming and also is used when some MOS devices should work on the same ports as you can not have two or more services running on the same ports. Advantage of the second solution is that you can start/stop or restart service only for one particular MOS device, but it consume more RAM. So it is up to your preferences. The second solution is recommended.

2. Edit file MOSAgent.xml if necessary. Something like this:

   ```
   <service>
    <id>OctopusMOSAgentService-%MOSID%</id>
    <name>Octopus MOS Agent-%MOSID%</name>
    <description>Octopus MOS Agent-%MOSID%</description>
    <mainClass>octopus.agents.mosagent.Main</mainClass>
    <javaArguments>-XX:MinHeapFreeRatio=10 -XX:MaxHeapFreeRatio=25 -XX:+HeapDumpOnOutOfMemoryError -XX:-OmitStackTraceInFastThrow --add-opens=java.base/java.lang=ALL-UNNAMED -XX:MaxGCPauseMillis=200 -Xmx
    <arguments>%MAIN_SERVER_IP%,%BACKUP_SERVER_IP% MOSAgentAccountName MOSAgentAccountPassword MOSAgentServiceName MOSAgentServiceInstanceName commaSeparatedMOSIDs</arguments>
    <javaHome>C:\octopus\jdk</javaHome>
   </service>
   ```

   Where -Xmx is maximum amount of the assigned memory.

   In case you would like to enable the service to provide metrics of the machine-level resources you have to add***Doctopus.statistics.reportMachineValues=true*** to the javaArguments. This is not needed in case the service is running on the same machine as the Octopus Server service as that service does it by default, or if other service on the same machine already has it enabled, so that only one service per machine provides those metrics.

   Should you want to change the default maintenance of the logs, you can add the following javaArguments:

   - -Doctopus.logger.archive=1d - defines after how many days the logs will be compressed, default is 3 days
   - -Doctopus.logger.purge=2d - defines after how many days the logs will be deleted, default is 14 days

   MOSAgent_password, MOSAgent_ServiceName, MOSAgent_InstanceName are CASE SENSITIVE!

   > **Tip**
   >
   > You can also use different port like this: 192.168.1.10:8443,192.168.1.11:8443 in arguments. Do not put space after the comma. For example <arguments>192.168.1.10:8443,192.168.1.11:8443 mos_agent password mos-%MOSID% A %MOSID1%,%MOSID2%

3. Install agent as a service by executing

   **MOSAgent.exe -install**

   in the command line in the location of the file. It will use MOSAgent.xml to set up the parameters.

   > **Warning**
   >
   > You can not edit the first four lines in the xml once the MOSAgent service is installed. You have to uninstall the service, modify the xml and install the service again (remember the recovery actions).

   Following logs are created once the service is installed:

   - systemErr.log - critical errors are logged here
   - systemOut.log - every start of the service is logged here

   > **Warning**
   >
   > Be aware of the Java Wrapper (.exe file) version. It might be 32bit or 64bit, depends on your OS version. 64bit Java Wrapper should be used on 64bit Operating System and you can not use 64bit java wrapper on 32bit operating system. The path to the jdk in the xml file depends on the used Java Wrapper.

   > **Tip**
   >
   > Uninstallation is straightforward:
   >
   > **MOSAgent.exe -uninstall**

> **Warning**
>
> Do not start the Octopus MOS Agent service yet. You can start it only after complete configuration.

4. Set the recovery actions in the ***Recovery*** tab in the properties of the Octopus MOS Agent service in the windows service manager to***Restart the Service*** on ***the First failure*** and ***Second failure*** and ***Reset fail count after*** 1 ***day*** and ***Restart service after*** 0 ***minutes***.

## 2.2. On Linux - Ubuntu

1. Download the following files and copy/extract them to /octopus/services/MOSAgent/ folder. If you do not have access to the URLs, ask Octopus Support for help.

   - The latest version of the archive (.zip file) fromhttp://download.octopus-news.com/install/10.0/ or from http://hudson.octopus-news.com/job/Octopus-10.0-STABLE/ws/dist/MOSAgent/. You can use the latest version of the ***Upgrade tool*** available in http://download.octopus-news.com/install/tools/ which can download all the required files when you provide correct credentials.
   - The latest version of services' XML files fromhttps://download.octopus-news.com/install/tools/XMLs/XMLs-Linux.zip or http://hudson.octopus-news.com/job/Octopus-TRUNK/ws/installer/service/MOSAgent/ when in internal octopus network.
   - The latest version of the startup scriptOctopusDaemon from https://download.octopus-news.com/install/tools/OctopusDaemon or from the scripts folder on Hudsonhttp://hudson.octopus-news.com/job/Octopus-TRUNK/ws/scripts/ when in octopus internal network and rename it toMOSAgent (without extensions). This daemon can be used for any Octopus services, just the name of the file has to follow the name of the .jar file without the extension.

     In case a different user is used to handle the services than***octopus***, you have to change the user in the daemon to the particular user that will handle the services. Simply change the user in the following line:

     ```
     USER="octopus"
     ```

2. Open the .xml for editing and configure javaHome to link to JDK, /octopus/jdk . Change Java Max memory parameter (-Xmx) to be 1g. It is maximum amount of the assigned memory. Edit id, name and description of the service - add MOSID after dash if necessary. Edit arguments to %MAIN_SERVER_IP%,%BACKUP_SERVER_IP% MOSAgentAccountName MOSAgentAccountPassword MOSAgentServiceName MOSAgentServiceInstanceName commaSeparatedMOSIDs

   Something like this:

   ```
   <service>
    <id>OctopusMOSAgentService-%MOSID%</id>
    <name>Octopus MOS Agent-%MOSID%</name>
    <description>Octopus MOS Agent-%MOSID%</description>
    <mainClass>octopus.agents.mosagent.Main</mainClass>
    <javaArguments>-XX:MinHeapFreeRatio=10 -XX:MaxHeapFreeRatio=25 -XX:+HeapDumpOnOutOfMemoryError -XX:-OmitStackTraceInFastThrow --add-opens java.base/java.lang=ALL-UNNAMED -XX:MaxGCPauseMillis=200 -Xmx1
    <arguments>%MAIN_SERVER_IP%:8443,%BACKUP_SERVER_IP%:8443 MOSAgentAccountName MOSAgentAccountPassword MOSAgentServiceName MOSAgentServiceInstanceName commaSeparatedMOSIDs</arguments>
    <javaHome>/octopus/jdk</javaHome>
   </service>
   ```

   Where -Xmx is maximum amount of the assigned memory.

   In case you would like to enable the service to provide metrics of the machine-level resources you have to add***Doctopus.statistics.reportMachineValues=true*** to the javaArguments. This is not needed in case the service is running on the same machine as the Octopus Server service as that service does it by default, or if other service on the same machine already has it enabled, so that only one service per machine provides those metrics.

   Should you want to change the default maintenance of the logs, you can add the following javaArguments:

   - -Doctopus.logger.archive=1d - defines after how many days the logs will be compressed, default is 3 days
   - -Doctopus.logger.purge=2d - defines after how many days the logs will be deleted, default is 14 days

   MOSAgent_password, MOSAgent_ServiceName, MOSAgent_InstanceName are CASE SENSITIVE!

   > **Tip**
   >
   > You can also use different port like this: 192.168.1.10:8443,192.168.1.11:8443 in arguments. Do not put space after the comma. For example <arguments>192.168.1.10:8443,192.168.1.11:8443 mos_agent password mos-%MOSID% A %MOSID1%,%MOSID2%

3. Check the permissions of the daemon with

   **ls -al /octopus/services/MOSAgent/MOSAgent**

   the file should have execute rights - x three times - for the owner, group, and others and be listed with green colour in the command output). Something like rwxrwxr-x. Change it if necessary

   **chmod +x /octopus/services/MOSAgent/MOSAgent**

   or

   **chmod 775 /octopus/services/MOSAgent/MOSAgent**

4. Create a symbolic link to /etc/init.d/ that will point toMOSAgent daemon:

   **sudo ln -s /octopus/services/MOSAgent/MOSAgent /etc/init.d/octopus-mos-$mosid**

   > **Tip**
   >
   > Keep in mind to have the same naming convention for all symbolic links for all Octopus services starting withoctopus-$ suffix.
   >
   > With correct naming of links you can simply start/stop/restart all Octopus services by one command
   >
   > **ls /etc/init.d/octopus-* | xargs -i$ sudo $ start/stop/restart**
   >
   > which takes the results of the **ls** command and replaces (that is what -i parameter does) the $ in the start/stop/restart command with them.

5. Set octopus-mos-$mosid to start automatically when your system boots. Put record to /etc/rc$runlevel.d folders for each service. To add service use the following command, just replace $SERVICENAME in command by proper name of the link in /etc/init.d/$servicename, for Octopus server typically octopus-server):

   **sudo update-rc.d $SERVICENAME defaults**

   You can double-check if service has been added in /etc/rc0.d folder:

   **ls -al /etc/rc0.d**

   Do not forget to add every octopus service you want to start after OS startup.

Should you ever need to remove service from OS startup, use command:

**sudo update-rc.d $SERVICENAME remove**

> **Note**
>
> MOSAgent can *only* be started by direct calling its script in /etc/init.d :
>
> **sudo /etc/init.d/octopus-mos-%MOSID% start**
>
> Same applies to restart|stop|status .

# Chapter 3. MOS Agent Configuration

## 3.1. General configuration

1. Launch Octopus client as administrator. Go to ***Administration / Users*** and create a user called ***octopus.mos*** of ***MOS Agent*** type with some password (remember that password). This will be the username and password used for the MOSAgent<->OctopusServer connection.
2. Go to the ***Administration / Admin services*** and create new service with type ***MOS Agent*** and set the name to something like MOS-%MOSID% (MOS-PROMPTER1 for example) (CASE SENSITIVE!). Leave ***IP address*** and ***Interface*** field blank - these fields are used only for the automatic failover of the Octopus MOS Agent - see the MOS Agent automatic failover section.
3. Create ***Instance*** inside of the service you have created in the previous step. Instances will be A, B, C, etc. depending on how many instances of the service you want to use (CASE SENSITIVE!).
4. Set up ***MOS devices*** in the Octopus client under ***Administration / MOS / Devices***. Configuration templates are possible to find on in https://confluence.octopus-news.com/display/SUP/Technology+Partners Technology Partners page (Confluence) or older configuration templates can be found in http://wiki.octopus-news.com/index.php/Inst:MOS_Device_-_konfigurace MOS Device configurations (Wiki). If you do not have access ask Octopus Support.
5. Once you have XML in your clipboard click on **Export** and paste the XML inside the new window, then click on **Import**. A new device will be created

   If you want to get the XML of the current MOS device just select desired MOS device and click on **Export** and then on **Export** in new dialogue. You will get the XML. You just have to change the MOSID to be able to import that XML back. Keep in mind that some information is not part of the XML, typically ***Activations*** and ***Prompting*** setting.
6. Select the newly created mos device, click on **Edit** and check the configuration (IPs, MOSID, Activation etc.).
7. Keep in mind that MOSID, MOSAgentUserName, MOSAgentPassword, ServiceName and InstanceName has to correspond with the setting in the MOSAgent.xml file for the particular service. So if you change something in the client configuration related to these values, you have to do the same changes in the MOSAgent.xml file.
8. You can launch the particular MOSAgent service if the configuration is done.

> **Warning**
>
> Any changes done in the configuration of the MOS Device in the client are applied by the restart of the particular service.

## 3.2. Toolbar

- **New** - you can create and configure new MOS device from scratch, but it is better to ask for the XML configuration or use already existing device configuration
- **Edit** - you can edit the configuration of the selected MOS device
- **Delete** - Only Octopus Support can delete the MOS device config as it might cause serious issues with the MOS objects that belong to the MOS device. MOS objects need the original MOSID as a reference. Deletion the MOSID should be possible only to fix a mistake, but you should never delete the MOSID that has been used for some time and already have existing MOS objects. Disabling the MOS device and creating a new one is the right way in that case. You can clone the config of the existing MOS device easily with the Export/Import button, just keep in mind that export/import does not cover the following MOS device config because they vary (different element labels and different channels) for all customers:
  - ***Prompter*** tab - the selection of the story elements
  - ***Activation*** tab - activation settings
- **Content sync** - you can run synchronization of the MOS objects with selected MOS device, Octopus supports two ways:
  - mosListAll sync - **Request mosListAll from the MOS device** sends mosReqAll with pause=0 command and MOS device should reply with list of MOS objects (mosListAll). This also generates new mosListAll.log file in the log folder of the corresponding MOSAgent service. You can find detailed description in https://mosprotocol.com/wp-content/MOS-Protocol-Documents/MOS_Protocol_Version_2.8.5_Final.htm#mosListAll.
  - Serial (mosObj) sync - **Request serial sync from the MOS device (this will send mosReqAll)** sends mosReqAll with pause=1 and MOS device should reply with MOS objects one by one. This can be seen in mos_%MOSID%.log file in the log folder of the corresponding MOSAgent service.

  ***pause*** indicates the number of seconds to pause between individual mosObj messages. More can be found in http://mosprotocol.com/wp-content/MOS-Protocol-Documents/MOS_Protocol_Version_2.8.5_Final.htm#mosReqAll
- **CG template sync** - works together with Vizrt. There are certain conditions that have to be met. QuickCG has to be enabled, templates have to contain externalId and edit field of type PLAIN. You can distinguish them by ***External Info*** of the template. ***VizDataServer*** in the MOS device configuration under ***CG Preview*** tab has to be set. To be able to see CG preview in Octopus ***Viz PreviewServer*** has to be set.
- **Export** - you can Export and Import configuration in XML format
- **MOS objects** - you can see the number of Public (available ) and Private (owned by certain story) MOS objects in the device. You can also delete MOS objects from Octopus database:
  - **Delete unused** - you can delete MOS objects that are not used in story nor story template. Those that are used somewhere will be marked as ***Disconnected***. Behaviour is the same as when we receive mosObj DELETED status.
  - **Delete all** - you can delete all MOS objects that are in Octopus database despite the fact that they are used in some stories.
- **Resend active rundowns** - all currently MOS active rundowns are sent again to the select MOS device.
- **Filter** - you can filter the MOS devices by their state - Disabled = Yes/No

## 3.3. MOS Agent automatic failover (optional)

You can have multiple instances of one service - A, B, C ... etc. it just depends on how many backup services you want to use. You need also one additional IP address per device, it will be called ***MOS Agent shared IP***. This IP will be used for the communication between Octopus MOS Agent and remote MOS gateway.

Octopus MOS Agent can control the assignment of the IP addresses. Once the particular instance becomes ***ACTIVE*** the MOS Agent assigns the configured ***MOS Agent shared IP*** to the configured network interface. The ***MOS Agent shared IP*** is released when the service goes down so DO NOT USE THE PHYSICAL SERVER IP. When the ***READY*** instance realizes that there is no ***ACTIVE*** instance it automatically assigns the configured ***MOS Agent shared IP*** to the specified network interface. Status of the service can show ***ERROR*** in case there is wrong name of the network interface or wrong IP without subnet - details can be found in the failover.log. Be aware that there might be the IP conflict when the server shuts down completely (power failure, etc.) and there was no time to release the ***MOS Agent shared IP*** and the other instance running on the other server is already using the same ***MOS Agent shared IP***. The service is designed to release the IP during the start if there is some unreleased IP and then assign it again if the service becomes ***ACTIVE***, so the IP conflict should be there only for a very short time.

Instance statuses

**ACTIVE**

Instance that currently communicates with the remote MOS gateway and has the MOSAgent cluster IP assigned. Command **netsh interface ipv4 add address %interface_name% %IP_address/subnet_mask%** is used to assign the IP.

**READY**

Instance that does not communicate to the remote MOS gateway and is waiting until no ***ACTIVE*** instance is present.

**ERROR**

Instance that has wrong name of the network interface or wrong IP format (without subnet) - details can be found in the failover.log or the IP is in use by other computer/service on the network.

> **Important**
>
> On Linux, **ip** command is used to manipulate addresses. It had been present for quite some time and it is the preferred way for the future (ifconfig had been deprecated a long time ago). In order to be able to run this command, we need to add octopus user to sudo configuration and allow execution of **ip** command without password. It is quite simple to set this all up, just follow the steps:
>
> 1. Create 10-octopus-ip-changes file:
>
>    **sudo nano /etc/sudoers.d/10-octopus-ip-changes**
> 2. put following lines into the file:

```
octopus ALL = (root) NOPASSWD: /sbin/ip
octopus ALL = (root) NOPASSWD: /usr/lib/heartbeat/send_arp
```

3. set permission to 0440 by running:

   **sudo chmod 0440 /etc/sudoers.d/10-octopus-ip-changes**

Example of the configuration:

MAIN MOS server - IP 192.168.1.10 and BACKUP MOS server IP 192.168.1.11 in 255.255.255.0 subnet which means netmask=24. We want three mos devices, all of them with two instances so if one service will be down the other one will act as the ***ACTIVE*** one. The priority is given by the order of the Instances under ***Administration / Admin services***.

> **Note**
>
> Octopus services are using **netsh interface ipv4 add address "%Name_of_the_network_interface%" %MOS_Agent_shared_IP%/%Netmask%** for IP assignment and **netsh interface ipv4 delete address "%Name_of_the_network_interface%" %MOS_Agent_shared_IP%** for IP deletion.

1. MOS Device PROMPTER

   There will be ***MOS Agent-PROMPTER*** service installed on both servers, the XML will differ only in one thing - the instance in the connection string. Instance ***A*** on ***MAIN MOS*** server and Instance ***B*** on ***BACKUP MOS*** server.

   The configuration in ***Administration / Admin services*** - fill in the ***IP address*** and ***Interface*** fields. The IP address is the %MOS_Agent_shared_IP_for_the_PROMPTER%/%SUBNETMASK% - it has to be unique IP. In our case 192.168.1.100/24 and Interface is the name of the network card where this IP will be added as the additional IP of the Interface - on Windows most typically Local Area Connection (2, 3, 4), on Linux eno1 (2, 3). The Interfaces have to have exactly same names on ***MAIN MOS*** server and on ***BACKUP MOS*** server. Remember that the remote MOS gateway of the PROMPTER has to be configured to communicate with the %MOS_Agent_shared_IP_for_the_PROMPTER%.

   You can start the Octopus ***MOSAgent-PROMPTER*** service. Typically start the MOS Agent service with instance ***A*** first and then the other services with other instances.

2. MOS Device GFX

   There will be ***MOS Agent-GFX*** service installed on both servers, the XML will differ only in one thing - the instance in the connection string. Instance ***A*** on ***MAIN MOS*** server and Instance ***B*** on ***BACKUP MOS*** server.

   The configuration in ***Administration / Admin services*** - fill in the ***IP address*** and ***Interface*** fields. The IP address is the %MOS_Agent_shared_IP_for_the_GFX%/%SUBNETMASK% - it has to be unique IP. In our case 192.168.1.101/24 and Interface is the name of the network card where this IP will be added as the additional IP of the Interface - on Windows most typically Local Area Connection (2, 3, 4), on Linux eno1 (2, 3). The Interface has to have exactly the same name on ***MAIN MOS*** server and on ***BACKUP MOS*** server. Remember that the remote MOS gateway of the GFX has to be configured to communicate with the %MOS_Agent_shared_IP_for_the_GFX%.

   You can start the Octopus ***MOSAgent-GFX*** service. Typically start the MOS Agent service with instance ***A*** first and then the other services with other instances.

3. MOS Device PLAYOUT

   There will be ***MOS Agent-PLAYOUT*** service installed on both servers, the XML will differ only in one thing - the instance in the connection string. Instance ***A*** on ***MAIN MOS*** server and Instance ***B*** on ***BACKUP MOS*** server.

   The configuration in ***Administration / Admin services*** - fill in the ***IP address*** and ***Interface*** fields. IP address is the %MOS_Agent_shared_IP_for_the_PLAYOUT%/%SUBNETMASK% - it has to be unique IP. In our case 192.168.1.102/24 and Interface is the name of the network card where this IP will be added as the additional IP of the Interface - on Windows most typically Local Area Connection (2, 3, 4), on Linux eno1 (2, 3). The Interface has to have exactly the same name on ***MAIN MOS*** server and on ***BACKUP MOS*** server. Remember that the remote MOS gateway of the PLAYOUT has to be configured to communicate with the %MOS_Agent_shared_IP_for_the_PLAYOUT%.

   You can start the Octopus ***MOSAgent-PLAYOUT*** service. Typically start the MOS Agent service with instance ***A*** first and then the other services with other instances.

You can monitor the state of the services under ***Administration / Admin services*** or under ***Administration / Online Users*** with enabled **Show agents** (you have to add additional columns like Service Name, Service Instance Name, and Status ( ***ACTIVE***/ ***READY***/ ***ERROR***).

***Summary***

We have two MOS Servers - ***MAIN MOS*** server (192.168.1.10) and ***BACKUP MOS*** server (192.168.1.11). We have three services ***MOS Agent-PROMPTER*** (shared IP 192.168.1.100), ***MOS Agent-GFX*** (shared IP 192.168.1.101) and ***MOS Agent-PLAYOUT*** (shared IP 192.168.1.102). All three services have two instances ***A*** - ***ACTIVE***, ***B*** - ***READY***). That means that if the same network interface is used for the configuration of all Octopus services the Local Area Connection interface will have four IP addresses on ***MAIN*** 192.168.1.10, 192.168.1.100, 192.168.1.101, 192.168.1.102 (command **netsh interface ipv4 add address "%Name_of_the_network_interface%" %MOS_Agent_shared_IP%/%Netmask%** is used to assign the IP) and just one on ***BACKUP*** 192.168.1.11. If one MOS Agent service on the ***MAIN*** server goes down it will release it's ***MOS Agent shared IP*** (command **netsh interface ipv4 delete address "%Name_of_the_network_interface%" %MOS_Agent_shared_IP%** is used to unassign the IP) and the instance ***B*** will set the ***MOS Agent shared IP*** on the network interface on the ***BACKUP*** server, so the communication between Octopus MOS Agent and remote MOS gateway is reestablished on the ***BACKUP MOS*** server. All active rundowns are re-send to the remote MOS gateway once the connection is established again.

## 3.4. MOS device configuration options

### 3.4.1. Basic

**mosID - case sensitive**

The unique ID of the MOS device (each MOS device must have a different one). The same mosID must be configured somewhere at the MOS device itself so that it sends this mosID to MOSAgent in its messages. If you decide to change the MOSID later on all the MOS objects that belong to this MOSID will disappear from the script.

> **Warning**
>
> Only Octopus Support can change the MOSID as it might cause serious issues with the MOS objects that belong to the MOS device. MOS objects need the original MOSID as a reference. Changing the MOSID should be possible only to fix the typo, but you should never rename the MOSID that has been used for some time and already have existing MOS objects. Disabling the MOS device and creating a new one is the right way in that case. You can clone the config of the existing MOS device easily with the Export/Import button, just keep in mind that export/import does not cover the following MOS device config because they vary (different element labels and different channels) for all customers:
>
> - ***Prompter*** tab - the selection of the story elements
> - ***Activation*** tab - activation settings

**ncsID - case sensitive**

Our own ID. It can be the same for every MOS device, for example, "Octopus". The same ncsID must be configured somewhere at the MOS device itself so that it sends this ncsID to MOSAgent in its messages.

**Version**

Octopus support MOS version 2.6, 2.8, 2.8.1 and 2.8.4. The version of MOS protocol used to communicate with the remote MOS gateway. The differences reflect the differences between different versions of the MOS protocol specification.

**2.6**

differences in behaviour from 2.8.4:

- different meaning of the <pause> tag in the <mosReqAll> message
- different format of the <mosListAll> and <roListAll> messages
- <messageID> is not sent

**2.8**

differences in behaviour from 2.8.4:

- <messageID> is not sent

**2.8.1**

same as 2.8.4

**Disabled**

Hides the MOS device from users, use it if you need to hide the MOS device temporarily instead of deleting it.

**Addresses**

MOSAgent can open one TCP connection for exchange of media data (Media host & port) and one TCP connection for exchange of rundown data (Rundown host & port). Rundown host and media host are almost always the same. You can omit configuration of a connection by leaving it empty, MOSAgent will not attempt to open the connection.

You can specify multiple addresses (by clicking on **New** ) if you want failover. MOSAgent will try to connect to the first address and if no exchange of messages with the current address happens for more than 30 seconds then MOSAgent will switch to the next address and it will try to communicate with it. Heartbeats must be enabled if you specify multiple addresses for the MOS device !

**Octopus media port, Octopus rundown port**

The two ports that MOSAgent listens for incoming TCP connections at. Different MOS devices can share the same listening ports as long as they are managed by the same instance of MOSAgent. Otherwise, one MOSAgent instance would prevent the other MOSAgent instances from using it if they run on the same machine. MOS device can connect to any port that a MOSAgent listens to as long as the MOSAgent was configured to communicate with this MOS device.

**Octopus IP address**

If the server, where MOSAgent service is running, has multiple network interfaces, you can specify the IP and/or the network interface that you want to use to communicate with the MOS device. Some network interfaces may not be able to communicate with the MOS device (different subnet) or it may be insecure. Keep in mind that MOSAgent service always listens on all interfaces so keep in mind the ports conflict. This setting is affected by the IP and interface set for the particular service in the Admin services:

- If no IP is set in Admin services and Octopus IP is not set then physical IP is used.
- If IP is set in Octopus IP in the MOS device config and no IP is set in the Admin services then IP in Octopus IP is used.
- If IP is set in the Admin services and Octopus IP is configured in MOS device config the IP in Admin services is used.
- If IP is set in the Admin services and Octopus IP is not configured then IP in Admin services is used.

**Response timeout [s]**

How much time should MOSAgent waits for a response after it sends a request to the MOS device. The connection on that particular socket (Rundown or Media port) is reset if the MOSAgent does not receive the answer in time.

**Response timeout for mosListAll [s]**

The response timeout to be used when MOSAgent is waiting for a <mosListAll> message to arrive. It takes much more time to get an answer to this request as it tends to be huge. You may need to increase it in case the MOS device needs more time to send <mosListAll>. You can find detailed description in https://mosprotocol.com/wp-content/MOS-Protocol-Documents/MOS_Protocol_Version_2.8.5_Final.htm#mosListAll.

**Final response timeout [s]**

How much time MOSAgent waits for a response after it sends a request to the MOS device. The full connection is reset if the MOSAgent does not receive the answer in time.

**The interval between heartbeats [s]**

<heartbeat> messages are used to identify dead connections. Their exchange confirms that a connection between MOSAgent and a MOS device is working. They are sent only if there are no other messages being exchanged through the connection. This config option says how often are the heartbeat messages sent. Entering of 0 disables heartbeats. They must be enabled if you specify multiple pairs of addresses for failover purposes.

### 3.4.2. Stories

**Send MOS objects that belong to other devices**

It makes all MOS objects visible for the MOS device regardless of the mosID of the MOS object. Otherwise, the redirected mosID of the MOS object must be the same as the mosID of the MOS device in order for the MOS object to be visible to the MOS device. This is mostly enabled for MOS devices that are supposed to play MOS objects from other MOS devices e.g. Astra and Fork.

**Send empty story elements**

If enabled then script elements will be visible for the MOS device even if they contain no visible MOS object or visible subelements. For example studio elements. Standard MOS protocol does not expect <item> tags without a MOS object inside. Enable this for Astra.

**Send empty stories**

If enabled then stories will be visible for the MOS device even if they contain no visible script elements.

**Send skipped stories**

For the purpose of this option "skipped" story means that it is marked as skipped in the rundown or it is located below its off-air line. If enabled then skipped stories will be visible for the MOS device - storySlug will contain a suffix ***[SKIP]***. Enable this for Astra. If disabled then skipped stories will not be visible for the MOS device.

**Use octext_roStorySkip**

Available only when ***Send skipped stories*** is enabled. If enabled all stories will contain the nonstandard <octext_roStorySkip> tag saying if the story is skipped (<skip>=1) or not (<skip>=0) instead of the suffix ***[SKIP]*** in storySlug. Skipping or unskipping a story will cause sending of the nonstandard <octext_roStorySkip> message. Enable this for Astra.

**Send MANUAL itemTrigger**

If enabled then MOSAgent will always send "MANUAL" in the <itemTrigger> tag, regardless of what kind the previous story element is. Enable this if "CHAINED" <itemTrigger> causes the MOS device to play clips automatically and this behaviour is undesired.

**Send subitems**

If enabled then MOSAgent will send script subelements as <subItem> tags located inside of the corresponding <item> tag which represents the script element that owns the subelement. If disabled then MOSAgent will send the subelements as independent <item> tags. The <subItem> tag in nonstandard and its point is to preserve the structure of stories - to show which subelements belong to which elements. It is supported by Astra.

**Send QuickCG**

If enabled then MOSAgent will send script subelements of type CG in the nonstandard QuickCG format instead of in the <item> or <subItem> tags (where the CG data are also stored in a non-standard way). This extension was invented by Vizrt.

**Default for QuickCG**

The enabled device will be preselected automatically when creating new QuickCG in the story script. Similar option is available in the CG device config.

**Hide from CG device list**

If enabled it hides the device from CG device list, this is useful for automation-type devices that do send QuickCG via MOS but those objects belong to another MOS device

**Send inserts**

If enabled then MOSAgent will send script subelements of type Insert (a.k.a. Commercial).

**Send inserts as MOS objects**

If enabled then MOSAgent will send script subelements of type Insert as <item> or <subItem> tags which contain standard <mosID> and <objID> tags instead of containing nonstandard <octext_insclipExtID> and <octext_insclipCrtNO> tags which are only supported by Astra.

**Send inserts with CHAINED trigger**

If enabled the subelements of the elements of type INSERT are sent with itemTrigger=CHAINED, otherwise they are sent as itemTrigger=MANUAL. The commercials should be usually triggered automatically so this is likely to be often enabled in the MOS device config.

**Send production requirements**

If enabled then MOSAgent will send information saying if a story is committed and the committed MOS object in <mosExternalMetadata> in all messages containing the <story> tag. Confusingly on top of that, it will also send custom columns of the story and in the <mosObjCreate> message, again in <mosExternalMetadata>. Supported by Astra and Primestream.

**Ignore production requirements**

If enabled the committed clip will not completely zero the text in storyBody in roStorySend message. This is useful in case there is a story with committed clip but you still need to see the story script text on the Telemprompter or Subtitler kind of MOS device. It sends the story in the same way as there was no committed clip at all. If disabled the storyBody will be just blank and neither the script text nor the presenter instructions will be sent over.

**Send story custom fields**

If enabled MOSAgent will send all created custom story/slug fields to the mos device.

**Send alternative texts**

**Send assets**

If enabled then MOSAgent will send list of story assets in <mosExternalMetadata> in all messages containing the <story> tag. This is a proprietary extension invented by Sienna.

**Search fields**

You can multi-select the fields that are used by the whisperer to search the mos objects in the story script.

**MOS command CUE IN/OUT mode**

You can define options for CUE IN/OUT of MOS command in the story script per device. Options are: Editable - ; Read-only - ; Hidden - ;

**Element itemEdStart**

Item/element Editorial Start in number of samples. Options are: Never - ; Override - ; Always

**Element itemEdDur**

Item/element Editorial Duration in number of samples. Options are: Never - ; Override - ; Always

**Subelement itemEdStart**

Command/subitem/subelement editorial start. Options are: Never - ; Override - ; Always

**Subelement itemEdDur**

Command/subitem/subelement editorial duration. Options are: Never - ; Override - ; Always

**itemEdStart means IN**

If enabled itemEdStart should contain the value of "IN" (from story script editor) rather than containing when does the subelement start relative to the start of its parent element ("CUE IN")

### 3.4.3. Rundowns

**Refresh method**

It says what kind of message is MOSAgent supposed to send for each active RO when MOSAgent connects to the rundown port of the MOS device. The point of refreshing the ROs is to ensure that the ROs at the MOS device are up to date after a time of disconnection when updates of the ROs could not be sent to the MOS device. Most of the time the best choice is "roCreate". Choose "None" if the MOS device asks for active ROs using the <roReqAll> and <roReq> messages after MOSAgent establishes a connection to it - we often do this with Astra.

**Interval of automatic rundown refresh [s]**

It tells MOSAgent how often should it send a <roReplace> message for each active RO. This is only meant for situations when the delivery of RO updates to the MOS device is unreliable for some reason and we want to periodically refresh all active ROs to ensure that they are up to date at the MOS device. Entering of zero disables this feature.

**Send roMetadataReplace**

If enabled then MOSAgent will send a <roMetadataReplace> message when a RO attribute (e.g. scheduled start, name) is modified.

**Send broadcast channel name in roChannel**

If enabled then MOSAgent will send the name of the rundown broadcast channel in the <roChannel> tag. This would violate the MOS protocol because the <roChannel> tag is supposed to have a different meaning. But for example, Astra interprets <roChannel> as the name of rundown broadcast channel. If disabled then the <roChannel> tag is not sent at all.

**Send rundown custom fields**

If enabled MOSAgent will send all created custom rundown fields to the mos device.

**Avoid roReplace**

If enabled then MOSAgent ???

**Resend only replaces stories**

If enabled then MOSAgent ???

**Detect all story modifications when rundown changes**

**roSlug pattern**

Pattern saying what is MOSAgent supposed to send in the <roSlug> tag in case of timed rundowns. The pattern can contain references to these variables:

**%TYPE%**

type of the rundown

**%START%**

scheduled start of rundown - time format is fixed - hh:mm; date format is based on the server locale setting

**%CHANNEL%**

broadcast channel of the rundown

For example, the pattern ***%TYPE% at %START%*** could generate this in <roSlug>: ***Sport at 11:00***

**Script alternatives**

It says which script alternatives are supposed to be sent to the MOS device.

**Send script alternatives selected on rundown or slug level**

### 3.4.4. Prompting

**Send story text**

The story text will be sent to the prompter MOS gateway

**Send story element labels**

The labels of the element will be sent to the prompter MOS gateway

**Keep sending roStoryReplace or roElementAction**

**Send colours**

Colours applied in the script will be sent to the prompter MOS gateway

**Send First/Last words**

First and Last words will be visible on the prompter MOS gateway. First/Last words are sent only if the script element text is NOT sent i.e. the script element type is not selected in Administration / Mos device / Devices / Edit / Prompting tab.

**Send script tags**

Story tags like Duration tag or MAR tag will be sent to the prompter MOS gateway

**Send video durations**

The duration of the clips will be visible on the prompter MOS gateway

**Send presenters**

The content of the presenter tags are sent in the <pi> (presenter instruction) tags and have usually inverted colours on the prompter software.

**Send ignored MOS objects**

If enabled even the ignored MOS objects (MOS objects that are in between double round brackets in the script) will be sent over as part of the roStorySend message. You have to enable it on the device that owns the object, not on the device that sends it.

**Send ignored text**

If enabled even the ignored text (text that are in between double round brackets in the script) will be sent in the <pi> (presenter instruction) tags.

**Send all story text elements**

You can enable to send whatever element exists to the remote MOS gateway, so when you create a new one it will be automatically involved. This is useful in case of the automation devices.

**Multiselect of the elements**

The content of the selected elements will be sent to the prompter MOS gateway

**Maximum empty lines**

You can configure maximum empty lines between the text. Other empty lines will be reduced.

### 3.4.5. Status

Definition of the roItemStat http://mosprotocol.com/wp-content/MOS-Protocol-Documents/MOS_Protocol_Version_2.8.5_Final.htm#roItemStat or roElementStatus http://mosprotocol.com/wp-content/MOS-Protocol-Documents/MOS_Protocol_Version_2.8.5_Final.htm#roElementStat messages. Depends on the version of the MOS protocol and on the implementation of the vendor of the MOS device.

**Accepts on-air status for**

If enabled then MOSAgent will save status coming from this MOS device. Otherwise, the <roItemStat> (or roElementStat) message will be ignored. Only one MOS device should have this option enabled so that multiple MOS devices do not overwrite statuses of each other in our rundown. But if you have multiple MOS devices receiving statuses for different rundowns then all of them may have this option enabled because their statuses won't collide.

MOS device can accept statuses on different levels. The statuses can be accepted for the following or the combination of the following:

- Rundown - MOS status for the rundowns/shows are accepted. It can look like this in case of <roElementStat> message:

```
2020-10-27 15:26:18.382    87 LogTopic.Agent.MOS.MSG INFO    Received message
<roElementStat element="RO">
 <roID>188894909</roID>
 <status>PLAY</status>
 <time>2020-10-27T15:26:19,978Z</time>
</roElementStat>. {addr=/192.168.101.30:54289} {mosId=mosart.mos} .
```

- Story - MOS statuses for the slugs are accepted. It can look like this in case of <roElementStat> message:

```
2020-10-27 15:26:18.385    87 LogTopic.Agent.MOS.MSG INFO    Received message
<roElementStat element="STORY">
 <roID>188894909</roID>
 <storyID>189267938</storyID>
 <status>PLAY</status>
 <time>2020-10-27T15:26:19,978Z</time>
</roElementStat>. {addr=/192.168.101.30:54289} {mosId=mosart.mos} .
```

- Element - MOS statuses for the elements/items are accepted. It can look like this in case of <roElementStat> message:

```
2020-10-27 15:26:18.388    87 LogTopic.Agent.MOS.MSG INFO    Received message
<roElementStat element="ITEM">
 <roID>188894909</roID>
 <storyID>189267938</storyID>
 <itemID>1--1</itemID>
 <objID>1</objID>
 <itemChannel/>
 <status>PLAY</status>
 <time>2020-10-27T15:26:19,978Z</time>
</roElementStat>. {addr=/192.168.101.30:54289} {mosId=mosart.mos} .
```

- Subelement - MOS statuses for the commands/subitems/subelements are accepted. It can look like this in case of <roItemStat message>:

```
2020-10-27 15:52:39.665   243 LogTopic.Agent.MOS.MSG INFO    Received message
<roItemStat>
 <roID>188894909</roID>
 <storyID>189267939</storyID>
 <itemID>1-2</itemID>
 <objID>1675</objID>
 <itemChannel>_PROGRAM_</itemChannel>
 <status>READY</status>
 <time>2020-10-27T16:52:41</time>
</roItemStat>. {addr=/192.168.101.31:56633} {mosId=PILOT} .
```

**Sets hit time to time of the first status in the story**

If enabled then statuses of type "On air" delivered for the first element in a story will set hit time of the story to the time specified in the <time> tag of the <roItemStat> (roElementStat) message. This way Octopus can display the time when each story really started to be played out and it can recalculate the estimated starts of the upcoming stories. Again, if you enable this option for multiple MOS devices then they may collide with each other.

**Calculates real slug duration from statuses**

Available only together with the ***Sets hit time to the time of the first status in the story*** . We set slug realdur (value in the Duration field with "v" suffix) to the time that elapsed since the manual start of the slug when following conditions are met:

1. If we receive one of the messages:
   - octext_storyAired
   - octext_storyOnAir with <airState>0</airState>
   - roElementStat with element=STORY
   - roElementStat with element=ITEM where <itemID> is empty, zero or not a valid number:

     ```
     2023-04-05 10:56:16.503 18991 LogTopic.Agent.MOS.MSG INFO    Received message
     <roElementStat element="ITEM">
      <roID>36386410</roID>
      <storyID>39539332</storyID>
      <itemID/>
      <objID>165</objID>
      <itemChannel>A</itemChannel>
      <status>PLAY</status>
      <time>2023-04-05T10:56:16</time>
     </roElementStat> {addr=/172.27.8.227:62384} {mosId=EXPLORER}
     ```

     and following with STOP

     ```
     2023-04-05 10:56:40.876 18991 LogTopic.Agent.MOS.MSG INFO    Received message
     <roElementStat element="ITEM">
      <roID>36386410</roID>
      <storyID>39539332</storyID>
      <itemID/>
      <objID>165</objID>
      <itemChannel>A</itemChannel>
      <status>STOP</status>
      <time>2023-04-05T10:56:40</time>
     </roElementStat> {addr=/172.27.8.227:62384} {mosId=EXPLORER}
     ```

     which simply means that when the itemID is empty the status is for the whole story and not just for one item in this case.
   - roItemStat where <itemID> is empty, zero or not a valid number

     ```
     2018-09-06 09:46:06.983    62 LogTopic.Agent.MOS.MSG INFO    Received message
     <roItemStat>
      <roID>49960345</roID>
      <storyID>55676807</storyID>
      <itemID/>
      <objID/>
      <itemChannel/>
      <status>PLAY</status>
      <time>2018-09-06T11:46:06,160</time>
     </roItemStat>. {addr=/192.168.1.100:3795} {mosId=astra}
     ```

and following with STOP

```
2018-09-06 09:46:11.165    62 LogTopic.Agent.MOS.MSG INFO    Received message
<roItemStat>
 <roID>49960345</roID>
 <storyID>55676807</storyID>
 <itemID/>
 <objID/>
 <itemChannel/>
 <status>STOP</status>
 <time>2018-09-06T11:46:10,800</time>
</roItemStat>. {addr=/192.168.1.100:3795} {mosId=astra}
```

which simply means that when the itemID is empty the status is for the whole story and not just for one item in this case.

2. status type in MOS device config = STOP
3. manual start (a.k.a. hit time) of the slug is not empty

**Accept status for slugs in not-ready rundowns**

If enabled then status coming from the MOS device will be saved even in case of rundowns that are not marked as ready to air. If disabled then status will be saved only in the case of ready to air rundowns.

**Rundown Stop status deletes statuses from all devices**

If disabled only MOS statuses of the MOS objects that belong to this MOS device are removed when rundown STOP status is received, otherwise MOS statuses of all MOS objects are removed.

This is where you define the list of statuses delivered in the <status> tag of the <roItemStat> (roElementStat) message. They must be ordered by priority - from the most important status to the least important status. When a story contains multiple MOS objects with different statuses then only the most important status will be shown in the rundown. Each status has

**Name**

that we expect in the <status> tag and that we display in the rundown

**FG (foreground colour)**

used when the status is displayed in the rundown

**BG (background colour)**

used when the status is displayed in the rundown

**Meaning (of items in drop-down list)**

- ***None*** (0) - status has no special meaning
- ***On air*** (1) - the story is on-air and it is locked by MOSAgent so the users can not do changes in such stories. In cooperation with automation we can calculate and show the real duration (the duration will have suffix "v" in the Duration column) of the whole story as it was broadcasted - check also ***Calculates real slug duration from statuses***.
- ***On air unlocked*** (4) - the story is on-air and it is not locked by MOSAgent so the users can do changes in such stories. In cooperation with automation we can calculate and show the real duration (the duration will have suffix "v" in the Duration column) of the whole story as it was broadcasted - check also ***Calculates real slug duration from statuses***.
- ***Cue*** (2) - the story is being cued and it is locked by MOSAgent so the users can not do changes in such stories.
- ***Stop*** (3) - story has been played. In cooperation with automation we can calculate and show the real duration (the duration will have suffix "v" in the Duration column) of the whole story as it was broadcasted - check also ***Calculates real slug duration from statuses***. This status is used to mark the rundown as aired - check ***Mark rundows aired*** in the ***MOS*** options in the ***System Setup***. Check the Admin Guide for more information about that option.

**Checkbox**

MOS Status is enabled or disabled.

You have to find out which statuses the MOS device may send and then enter them here. As it is case sensitive they have to match exactly. You could ask the vendor of the MOS device about this. Standard statuses ordered by priority are: NOT READY (set colour to red on yellow background), PLAY (set meaning to ***On air***), CUE (set meaning to ***Cue***), STOP (set meaning to ***Stop***), READY.

### 3.4.6. Channels

You have to first enable the MOS channels under Administration / System Setup / Story tab / Using MOS channels to be able to use the MOS channels.

**Accept channels incoming from MOS device**

Octopus will show the received channel information in the Channel column in the rundown.

**RO channels**

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

If you want to add an RO channel click on **Add** then select newly created white line and press **F2**, write the name of the RO channel (News, Sport ...) and press **Enter**. It has to be exactly the same (it is case sensitive) as on playout.

**Item channels**

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

If you want to add an Item channel click on **Add** then select newly created white line and press **F2**, write the name of the Item channel (A, B, C, ... 1, 2, 3 ...) and press **Enter**. It has to be exactly the same (it is case sensitive) as on playout.

**Item channels rules**

Rules specify the order how the Item channels will be assigned to the clips in the rundown.

If you want to add an Item channel rule click on **Add**, then select the newly created white line, then select the Item channels (A, B ...) which you want to add into the rule (you can use multi-select with Ctrl or Shift) and click on **Add channels to rule**. You can have multiple rules. **Clear rule** will erase the selected rules, **Delete** will delete the selected rule.

### 3.4.7. Lowres

**Expiration**

You can specify when are the thumbnails of the lowres files refreshed, because they are cashed on the client side and might happen that when they were not created when the MOS object arrived they were not shown later on at all.

- (none) - they are never refreshed
- Maximum age - you can set the refresh period is seconds
- Using "changed" - client watches the "changed" field/tag of the MOS object XML and acts if it has changed.

**Path to keyframes**

If paths to keyframe files are relative paths instead of absolute paths then this field must contain the beginning of the absolute path to be inserted before the relative paths.

**Automatic keyframe pattern**

If the MOS device does not send keyframes in the <mosObj> messages but the keyframes exist on a path that can be deduced from the MOS object according to a pattern then you can enter the pattern here. The pattern is the path which contains references to these (case-sensitive!) properties of the MOS object: objId, objSlug, jobId.

For example the pattern

**\\lowres-server\keyframes\%objId%.jpg**

will generate this path to keyframe

\\lowres-server\keyframes\CLIP123.jpg

for the MOS object with <objID>CLIP123</objID>.

There is no way to deduce multiple keyframes for one MOS object.

**Automatic keyframe width [px]**

Width of the pattern-deduced keyframe in pixels.

**Automatic keyframe height [px]**

Height of the pattern-deduced keyframe in pixels.

**Path to lowres video**

Similar to "Path to keyframes".

**Automatic lowres pattern**

Similar to "Automatic keyframe pattern".

**Automatic lowres type**

Type of the pattern-deduced lowres. WMV; QT; MPEG1; MPEG-DASH; RM, BROWSER (allows to launch an external browser with custom URL specified in ***Automatic lowres pattern***). This pattern is used in case MAM does not provide ***objProxyPath*** with ***techDescription***.

**Path to mobile lowres video**

Similar to "Path to keyframes".

**Automatic mobile lowres pattern**

Similar to "Automatic keyframe pattern".

**Path to hires video**

Similar to "Path to keyframes" but it points to the original full quality video of the MOS object.

**Automatic hires pattern**

Similar to "Automatic lowres pattern".

**Path to image of missing thumbnail**

Path to the image to be displayed as the thumbnail image in case the MOS object lacks a thumbnail.

**Lowres signing**

***Public key ID*** and ***Private key*** needs to be specified in case of secured access to the lowres paths.

**Type patterns**

they are case sensitive, comma separated (no space after comma!) and are used to match the value in techDescription. Client searches for techDescription and its value in objProxyPath and matches the exact string. If techDescription is not part of the objProxyPath the ***Automatic lowres type*** is used instead. Sometimes you need to check the MOS object's XML for the exact string in the techDescription (relevant part of the XML can be found in Clips tab in the XML section, or the whole xml of the MOS object can be found in the database, or in the logs or if you drag and drop the MOS object into a WordPad for example). You can find examples and details about objPath, objProxyPath and techDescritpion in https://mosprotocol.com/wp-content/MOS-Protocol-Documents/MOS_Protocol_Version_2.8.5_Final.htm#objPaths

**MPEG-DASH lowres patterns**

MPD,DASH,MPEG-DASH

**WMV lowres patterns**

WM,wm,WMV,wmv,MP4,mp4,MP3,mp3,MXF,mxf,video/mp4

**QT lowres patterns**

QT,qt,MOV,mov

**H264 lowres patterns**

264

**MPEG1 lowres patterns**

MPG,mpg,MPEG,mpeg

**RM lowres patterns**

RM,rm,REM,rm

**JPG lowres patterns**

JPG,jpg,JPEG,jpeg,PNG,png

**BMP lowres patterns**

BMP,bmp

### 3.4.8. CG Support

**All MOS objects are CGs**

When enabled all MOS objects that belong to this MOS device will be considered as CGs and they will be visible in the CG tab in the story script assets, when CG filter enabled in the rundown Buddy. If disabled you can configure more sophisticated rules under the ***Rules exist*** `...` to detect which MOS object should be considered as CG, for example:

```
<script>
 <equals>
  <getVar name="objDur"/>
  <string>1</string>
 </equals>
</script>
```

will mark as CGs only the MOS objects with objDur=1.

**CG Type**

- (none) - CG Preview is disabled
- Vizrt

  **Viz DataServer**

  IP, hostname or FQDN of the viz database server (always lowercase as it is case sensitive, this value is used to adjust the external info value of the CG templates by removing this prefix out of it, case matters, it has to match to whatever we receive from Pilot Data Server). Default port is 8177.

  **Viz PreviewServer**

  IP, hostname or FQDN of the viz preview server (always lowercase as it is case sensitive, this value is used to adjust the external info value of the CG templates by removing this prefix out of it). Default port is 21098.

  **Save CGs as objects on Vizrt side**

  QuickCGs added to the story script created by the CG templates received over API trigger a MOS object creation in Octopus and on Vizrt side as well, so from the user perspective it looks there is just a CG, but on the background there are MOS objects created on both sides.

  **Create object copy on Vizrt when command/story is copied**

  Copy CG command/story will create a new object on Vizrt side, which means original CG and copied CG are not the same object. If this option is disabled, copied CG is the same object as the original CG.

  **Update CGs when saving on Vizrt side (if not, make sure we receive them over MOS)**

  QuickCGs are originally created via the API call to Pilot Data Server and in case of changes Vizrt triggers update of that MOS object over TCP/IP socket which results in another API request from Octopus to Vizrt which is not necessary and might be slow. If you enable this option we will request the info about the MOS object again over API which returns the same

thing as we have already received over API previously which is not needed, so this option is usually not needed.

**Synchronize all templates (not only those having an external_id)**

never mix templates synchronized in ***external_ids*** mode and ***all*** mode. In case this is option is changed you have to delete the existing CG templates and sync them again, because the CG templates with external_id contains additional information about the variant in their URI (Uniformed Resource Identifier), which is URL. Combination of syncing only the CG templates with external_id and mode with QuickCG does not make sense as the CG templates received over API are stored as MOS objects on the background in the database and not as a simple CG template as in case of a simple CG device config integrated via CII.

**Import image fields in templates -**

**Import rich text fields in templates**

allows to import the graphic templates that contain rich text fields (this is default format of the fields in VIZ tempalte designer, but it does not mean the content is really rich text)

- MOS proxy - used with the graphic devices that can send the objProxyPath to the rendered image as part of the MOS object's XML, so you can see the preview in the CG tab in the story script's assets, as part of the subelement w(when enabled) and in the Buddy with CG toggle enabled.

### 3.4.9. MOS objects

**Update private objects**

If enabled then incoming <mosObj> messages received over the TCP/IP socket will overwrite not only existing public MOS objects that have the same <objID> and <mosID> but also existing private MOS objects located in stories that are not archived (usually the ones received over the plugin if the plugin is set that the MOS objects received over the plugin are private).

**Allow mosItemReplace**

If enabled then incoming <mosItemReplace> messages will be processed, otherwise, they will be ignored. mosItemReplace has affect on the following information: mosAbstract, objPaths, itemSlug, mosExternalMetadata, itemEdStart, itemEdDur, itemChannel.

**Allow mosItemReplace start change**

Enables translation of itemEdStart from the mosItemReplace to the manual MOS IN point.

> **Note**
>
> You must have ***Update private objects*** enabled, as the mosItemReplace will make the item in the story PRIVATE. If you don't enable it, you won't receive the duration and objAir status update to the object in the story.

**Allow mosItemReplace dur change**

Enables translation of itemEdDur from mosItemReplace to the manual MOS OUT/DUR point.

> **Note**
>
> You must have ***Update private objects*** enabled, as the mosItemReplace will make the item in the story PRIVATE. If you don't enable it, you won't receive the duration and objAir status update to the object in the story.

**Allow bound MOS objects**

MOS objects that belongs to this MOS device will be available for the bound MOS object actions (a story element can have bounded automation template preassigned). This is most typical for the automation MOS devices (Astra, Mosart, etc.).

**Bound MOS objects fill script**

**Merge mosItemReplace EMD tags**

Merges mosExternalMetadata received in mosItemReplace message over the socket connection. Usually applicable on devices that send mosItemReplace messages (one MOS object can have multiple items/versions with different values for duration for example), for example Sonaps.

**Merge plugin EMD**

Merges mosExternalMetadata received over the plugin. This makes sure that Octopus updates the metadata of the item when changed.

**Device is target of redirection**

When enabled the MOSAgent sends the MOS objects that belong to the MOS devices selected in the field below. Change requires restart of the particular MOSAgent service.

**Translate redirected IDs**

If enabled then MOSAgent will send all mosIDs as if they were redirected. If disabled then redirection still influences visibility of MOS objects but their mosIDs are sent unaltered. It depends on if the MOS device expects original mosIDs or its own mosID.

**Supports mosListAll**

Listing of All Object Data from MOS, send MOS object descriptions in a format similar to mosObj messages from the MOS to the NRCS. mosListAll is initiated by a properly ack'd mosReqAll message from the NRCS. You can find detailed description in https://mosprotocol.com/wp-content/MOS-Protocol-Documents/MOS_Protocol_Version_2.8.5_Final.htm#mosListAll.

**Delete lowres files when MOS object is deleted**

If enabled then when a MOS object is being deleted from the Octopus DB then all of its lowres files stored on disk (or network) drives will be deleted also. But only if they are not being referenced by another still existing MOS objects. This is mostly needed when Octopus is responsible for the creation of the lowres files and so it is also responsible for the deletion of the files.

**Out-point is inclusive in timecodes**

If enabled then the lengths of time intervals defined by IN and OUT points will be calculated as OUT - IN + 1. If disabled then they will be calculated as OUT - IN. This is an Astra oddity, enable it for Astra if they need it.

**objDur is in frames**

If enabled then the <objDur> tag in the <mosObj> message is assumed to contain frames instead of fields (one frame consists of two fields). Sending of <objDur> in frames is a bug. This was a fix for Quantel device.

**Default objAir is READY**

If enabled then <mosObj> messages which lack the <objAir> tag are assumed to be READY, otherwise NOT READY. Sending of <mosObj> without an <objAir> is a bug.

**Use in story Ready field**

objAir status of the MOS objects that belong to this MOS device will be considered in overall objAir status in Ready column.

**Count dur even if NOT READY**

The duration of the clip (can be manually adjusted with Manual clip IN/OUT) will be considered in the overall duration of the whole story even when the MOS object has objAir ***NOT READY***.

**Count dur even if zero**

The duration will be considered in the overall duration of the whole story even when the MOS object has duration equal to zero.

**Story unarchive replaces private objects with public**

When restoring a story from archive, Octopus will check for the presence of the public MOS object copies of the private objects inside the story and if they exist, Octopus will replace them with public MOS object.

**Allowed types**

List of allowed types that will be available in the placeholder creation dialogue. Options are ***AUDIO***, ***STILL*** and ***VIDEO***. Type field is available in the dialogue only if more than one type is allowed, otherwise, the drop down field is hidden.

**Saving merge interval (0=disabled)**

**MOS object deletion delay [s]**

**Field used as clip identifier in script**

- objID -
- Name -
- jobID -

**Story content MOS object display**

The look of the MOS object in the story content column can be specified:

- Standard -
- Mosart - Used in case of plugin-less integration with Mosart. Based on this setting we know that we should group all the MOS objects together based on the objGroup (template set in mosart)
- Astra - Used in case of plugin-less integration with Astra.

  There are two types of Astra MOS objects - templates and instances:

  - Templates are supplied by Astra and they describe what kind of editable fields is a MOS object supposed to have.

    Templates have a <mosExternalMetadata> tag with <mosSchema>astra://OBJECT/ASD_VARIANT</mosSchema> where the fields are defined.
  - Instances are created by Octopus, they are based on a template and they contains the values entered by users to the fields defined by the template.

    Instances have a <mosExternalMetadata> tag with <mosSchema>astra://OBJECT/ASD</mosSchema> where the values of the fields are stored.

  When you try to insert a template MOS object into script editor as a bound MOS object, it is automatically replaced with a new instance MOS object based on this template. Modifications made to the instance by the user in the script editor are saved to the instance, not to the template.

  The new instance has objID equal to the objID of the template and its mosID is taken from attribute "mosId" of the <model> tag in the template XML.

  MosID and objID of the template, that an instance is based on, is saved in the <mosPayload> tag in the XML of the instance.

  If the template that an instance is based on no longer exists, the instance cannot be edited by the user and an error will be shown.

  If a form exists that has name which is equal to "AstraTemplate" followed by the objID of the Astra template that we are editing, then the layout of the editor will be created according to this form. Otherwise it will be created according to the layout specified in the Astra template XML under the <view> tag.

  To create such a form, you first have to bind some MOS object custom fields to fields (tags <inputElement>) in Astra template XML: in AS fields/MOS object, set type of the custom field to ***Astra*** and enter ID of the Astra field that the field is supposed to represent. The Astra field could e.g. look like this <inputElement id="duration"> in the template XML. So the value in the MOS object custom field should be ***duration***.

**Device type (used for description extraction)**

It extracts and display the text from the MOS objects that is send in mosExternalMetadata/mosPayload/objects/object/name

- (none) - nothing is used as predefined structure of the description and the value in ***XPath to extract description*** is used instead.
- ROSS Xpression - we use the known structure of the XPression MOS objects instead of the value in ***XPath to extract description***.
- Harris Inscriber - we use the known structure of the Inscriber MOS objects instead of the value in ***XPath to extract description***.

**XPath to extract description - case sensitive**

//mosAbstract (option for vizrt to extract and display content of the graphic MOS objects); //name | //data (option for inscriber where "|" means AND).

More information about XPath can be found in https://developer.mozilla.org/en-US/docs/Web/XPath. More complex XPath expressions can be used, you can use https://extendsclass.com/xpath-tester.html and the MOS object's XML in order to test/validate or generate the xpath expression.

**XPath to extract secondary ID - case sensitive**

You can link together the MOS object and the wire as far as the wire and the MOS object contains some shared identifier GUID (can be found in the InjectAgent ***serverConnection.log***, in the InjectAgent Test parser dialogue and in the guid table in the database). For example, if the wire has video attachment called "Octopus_wire_link.mp4" and then you received the MOS object called exactly the same "Octopus_wire_link" then when you open the wire you will see the link to the MOS object in it when this field is set to ***//objSlug*** (eventually ***//objID*** or other unique field that is part of the XML). Keep in mind that this field has to be set before you receive MOS object from MAM as it is parsing the secondary ID during the reception of the object.

More information about XPath can be found in https://developer.mozilla.org/en-US/docs/Web/XPath. More complex XPath expressions can be used, you can use https://extendsclass.com/xpath-tester.html and the MOS object's XML in order to test/validate or generate the xpath expression.

**MOS object save script**

There might be a completely configurable MOS object save script.

**Send mosExternalMetadata of MOS objects**

MOSAgent only sends those <mosExternalMetadata> blocks which have their <mosScope> selected here.

- OBJECT -
- STORY -
- PLAYLIST -

**Nonindexed XPaths**

**Story clips**

You can configure the look of the Clips column, you can define what information, related to the particular MOS object, are shown.

**MOS object**

Here you can define what will represent MOS object n the clips column:

- objID - unique object ID received from MAM (Media Asset Management)
- Name - name of the MOS object - name can be anything and also can have predefined patterns. See next section related to the Placeholders and Naming patterns
- jobID - only Astra related, it has been introduced as proprietary identifier way before MOS protocol

**Status**

Here you can define what will be part of the MOS status shown in the clips column:

- None - MOS status (roItemStat or roElementStat) will not be shown at all
- Without device - only MOS status (roItemStat or roElementStat) will be shown
- mosID - MOSID of the device that has sent the MOS status (roItemStat or roElementStat) will be shown together with the MOS status
- shortName - Short name defined in the Other tab of the device that has sent the MOS status (roItemStat or roElementStat) will be shown together with the MOS status

**Channel**

Here you can define what will be part of the MOS status shown in the clips column:

- None - channel (mosItemChannel) will not be shown at all
- Without device - only channel (mosItemChannel) will be shown
- mosID - MOSID of the device that has set the channel (mosItemChannel) will be shown together with the channel
- shortName - Short name defined in the Other tab of the device that has set the channel (mosItemChannel) will be shown together with the channel

**Show MOS object status**

Keep in mind that there are another two configuration options that are not MOS related. You can set ***Story clip ids element format*** and ***Story clip ids tape element format*** in ***System Setup*** under ***Story*** tab.

To be able to use the mosItemChannel you have to enable ***Using MOS channels*** in ***System Setup*** under ***Story*** tab and configure the channels in the particular MOS device config under ***Channels*** tab.

### 3.4.10. Placeholders

**Allow MOS object creation**

If enabled then users will be allowed to create new MOS objects on this device using <mosObjCreate> messages.

**Default MOS object creation device**

If enabled then this MOS device will be offered to users as the default MOS device to create new MOS objects at.

**Allow automatic MOS object creation**

If enabled then users will be allowed to create new MOS objects on this device without specifying the attributes of the new MOS object. The attributes will have default values according to the settings below and the user will not be able to change them. It saves the users one mouse click and prevents them from entering incorrect attributes.

**Use the <mosObjCreate> message**

Whether mosObjCreate message or mosReqObjAction operation="NEW" is used to request the remote MOS device to create a placeholder. More can be found in http://mosprotocol.com/wp-content/MOS-Protocol-Documents/MOS_Protocol_Version_2.8.5_Final.htm#mosObjCreate and http://mosprotocol.com/wp-content/MOS-Protocol-Documents/MOS_Protocol_Version_2.8.5_Final.htm#_3.3.3_mosReqObjAction_%E2%80%93_NCS_request. mosReqObjAction has been implemented since version 2.8.2.

**Allow Video Trimmer**

Easy Video Trimmer allows simple cutting directly from the Octopus Client. This works with Tools On Air currently.

**Object (auto) create shortcut number**

Here you specify which number will be associated with the Create placeholder and Auto-create placeholder action.

**Default duration of created MOS objects**

This applies to both automatic and non-automatic creation.

**Number of digits of the autoincrement fields**

How many digits will be in the number when %INC is used in the MOS object naming pattern.

**Starting value of the autoincrement field**

Starting value of the autoincrement parameter when %INC is used in the MOS object naming pattern.

**Naming pattern of created MOS objects**

The pattern is a line of text which can contain references to variables. Preferably it should attempt to generate unique names. This applies to both automatic and non-automatic creation. You can use the following variables:

**%END**

is the ***END*** mark in case the variable ends with a number. Typically STORYCOL*, SHOWCOL* and NAME*

**%NAME**

name of the parrent entity (Story, Assignment, etc.), it can be immediately followed by the maximum length of the substituted text e.g. %NAME5

**%INC**

auto-incremented MOS object number, for this MOS device. The number of digits of the autoincrement field and the starting value of the autoincrement field can be set. If autoincrement field overpasses it's number of digits, it's current value returns to 1.

**%OBJGROUP**

you can choose the objGroup of the MOS device

**%SYEAR %SYEAR2 %SMONTH %SDAY %SDOWD %SDOWS %SHOUR %SMIN %SSEC**

components of the scheduled start of rundown timestamp. SDOWD - rundown timestamp day of the week as a digit, SDOWS - the same in the common short string form, in accordance to server settings.

**%CYEAR %CYEAR2 %CMONTH %CDAY %CDOWD %CDOWS %CHOUR %CMIN %CSEC**

components of the current timestamp (according to server clock). CDOWD - current day of the week as a digit, CDOWS - the same in the common short string form, in accordance to server settings.

**%CHANNEL**

name of the channel the placeholder is created in, if there is no channel then the pattern is not used

**%SEGMENT**

name of the segment the placeholder is created in, if there is no segment then the pattern is not used

**%ELABEL**

label of the story element e.g. PKG

**%EID**

ID of script element (it is unique within the story)

**%ENTITYID**

ID of the slug, story or assignment

**%UNAME**

username of the user creating the new MOS object

**%SNAME**

shortname of the user

**%LNAME**

longname of the user

**%STORYCOL***

content of the custom story field can be part of the pattern, just replace the * by number of the custom field (1-64)

**%SHOWTYPE**

show/rundown type (name of the show/rundown)

**%SHOWCOL***

content of the custom show field can be part of the pattern, just replace the * by number of the custom field (1-10)

For example the pattern **%NAME %SYEAR-%SMONTH-%SDAY %SHOUR:%SMIN:%SSEC %ELABEL** could generate this MOS object name: Elvis lives 2016-02-22 12:34:08 PKG Pattern is only applied to newly generated MOS objects.

**Naming pattern filters**

Some MAMs can not handle certain Unicode letters so as a workaround you can configure filters in Octopus. Filters can be combined and are applied in the order as they are in the list.

1. Cyrillic to Latin - transforms cyrillic letters to latin letters
2. Alphanumeric, hyphen and comma only - allows only latin alphabet, arabic numbers, hyphens and commas, and removes everything else
3. All letters UPPERCASE - transforms all letters to uppercase
4. All letters lowercase - transforms all letters to lowercase
5. Remove diacritics - removes diacritic marks
6. Remove whitespace - removes all whitespaces

7. Whitespace to underscore - replaces white spaces with underscores
8. Translation table - you can define the translation table. You have to define the ***Unicode hexadecimal*** interpretation of the letter you want to be replaced and ***Ascii decimal*** interpretation of the letter you want to replace it with.

**Form**

You can configure the form that will be used in the new placeholder dialog.

**Send <createdBy>**

You can configure the format of the username that is sent to the remote MOS device when requesting creation of the placeholder

- (none)
- User name
- Short name
- Long name

### 3.4.11. MOS object groups

objGroups are used to distinguish the clips by the object group. You can have different groups based on the channel, language or topic. The groups have to be supported by the Media Asset management software as it is setting this value. The value has to be exactly the same on both sides.

objGroups:

- Add - create a new object group
- Use by default - set the selected object group as the default one
- Delete - delete selected object group

Rule-based groups:

- Add - create a new object group
- Edit - you have to define an algorithm in the XML minilanguage
- Delete - delete selected object group

### 3.4.12. Activations

**Activate folders**

If enabled then users will be allowed to manually activate story folders for this MOS device.

**Mark rundowns ready on activation**

Activation of rundowns is configurable for each channel separately:

**Enabled**

It enables the possibility to activate rundowns from this channel (both manually and automatically).

**Activate range**

It enables automatic activation of rundowns from this channel based on the ***Rundown range*** (1 ***Back*** and 1 ***Ahead*** means that 3 rundowns will be MOS active - 1 past rundown, 1 future rundown, and 1 current rundown) or ***Time-based range*** (keep in mind that the duration of the rundown is considered as well so in case it is set 60 minutes ***Back*** and rundown has 60 minutes, it will be mos deactivated in 120 minutes, this is precaution in case that the ***Back*** value is too small as the rundown could be MOS deactivated during broadcasting). You have to specify how many rundowns/minutes preceding the current rundown in ***Back*** field and how many rundowns/minutes succeeding the current rundown in ***Ahead*** field will be considered active in this channel. Entering unnecessarily high numbers could cause performance problems. The current rundown is the rundown which is supposed to be on-air right now according to it's scheduled start and duration. If there is no such rundown then the current rundown is the first rundown in the future. If there are no rundowns in the future then the current rundown is the last rundown in channel. Only non-archived rundowns can be current rundowns.

### 3.4.13. Plugins

**MOS objects from plugin are private**

MOS objects inserted from the ActiveX plugins will be considered as private. Most typical for the graphic MOS devices.

**Send user domain to plugin**

Some plugins require authentication and client can pass the domain of the currently logged in user to the plugins.

**Accept itemEdStart/Dur from plugin**

We receive and accept the itemEdStart/Dur values from the plugin and it is visible in the GUI.

**Send itemEdStart/Dur to plugin**

We send the itemEdStart/Dur values to the plugins when enabled.

**Accept itemChannel from plugin**

We send the itemChannel values to the plugin - keep in mind that the ***Using MOS channels*** have to be enabled in the ***System Setup/Story*** tab and that the MOS channels have to be set in the ***Channels*** tab of the MOS device.

**Send itemChannel to plugin**

We receive and accept itemChannel values from the plugin - keep in mind that the ***Using MOS channels*** have to be enabled in the ***System Setup/Story*** tab and that the MOS channels have to be set in the ***Channels*** tab of the MOS device.

**Plugins**

- Add - create a new plugin config
  - Short name - a short name of the plugin
  - Long name - a long name of the plugin; you can use special parameters in parenthesis such as: (OOPC) = out of process, (AS) = autosave and (AC) = auto close
  - Width - width of the window of the plugin
  - Height - height of the window of the plugin
  - Type
    - Player (ActiveX) -
    - Browser (Active X) -
    - Editor -
  - Version
    - 1.0 ENPS -
    - 1.0 iNews -
    - 2.8 -
    - 2.8_Web - this is the recommended version when using CEF platform. Should you experience some issues you might try version 2.8
    - WebForm -
    - 2.8_IE -
  - Platform
    - ActiveX -

        - Javascript -
        - Chrome (CEF - Chromium Embedded Framework) -
    - Placement
        - Modal - plugin will run in a modal window/dialogue
        - Modeless - plugin will run in a modeless window/dialogue
        - Modeless reused -
        - Vertical split - plugin will run in a vertical split
        - Horizontal split - plugin will run in a horizontal split
        - None - there is no particularly defined placement, this is used together with the ***OOPC (Out Of Process)*** mode.
    - Shortcut number - you can override the automatically assigned shortcut number
    - Stretch when inline -
    - Allow creation in media - you can enable the plugin availability in the Media section
    - Run in separate thread -
    - Run in separate process -
    - Run in native window -
    - Reuse running plugin - if the client should not open the already opened and running plugin again and again. This is helpful also when the plugin is set to run in the split.
    - Use <ncsAppInfo> - when launching the plugin following information are passed to the plugin - userId, runContext (CREATE, EDIT, BROWSE), and software info (manufacturer, product and version)
    - <ncsAppInfo> sends <item> - ncsAppInfo 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 represent an "empty" control. If enabled we send the item when the plugin is launched from within a story. If the plugin is launched from the Media Library we send mosObj.
    - Show use -
    - Show close -
    - Close when object arrives -
    - Primary editor - Option to configure the mos plugin to define which plugin (in case two or more plugins are available) should be opened if they Ctrl+double-click on the MOS Command anchor or launch the plugin when you click on cogwheel.
    - Allow in read-only script -
    - Implementation
        - ActiveX plugin - enter GUID (starts and ends with curly brackets, it is the key/location in the Windows registry)
        - HTML plugin - enter the URL of the plugin.
            - When using plain URL the whole host page is pulled from our server (http:/octopus.somedomain.lan:$port/mos/plugin?pluginUrl=http://viz-gw.somedomain.lan/pilotedge - this URL can be found in the client's browser.log) as the iframe is used to host the plugin's URL and our Web Server hosts it. Values in listenOn and port are used for the server part of the URL (http:/octopus.somedomain.lan:$port), it can be overridden/replaced with a different FQDN, for instance the cluster one, by configuring ***Javascript platform hostpage url override*** in the ***Administration/System Setup/MOS***) tab, this is necessary to solve cross-origin-domain issues with clipboard. The protocols must match as well, you can not host URL with http on https and vice versa.
            - When you start the URL with "|" (a pipe sign) at the beginning it means the whole host page is stored on the provided URL (http://viz-gw.somedomain.lan/pilotedge), in such case the plugin must be really fully hosted on that Web Server. This is typical for the cloud-based plugins, for instance Mimir.
    - Separate process directory
- Edit - edit the existing selected plugin config
- Delete - delete the existing selected plugin config

### 3.4.14. Vizrt

See the GatewayAgent guide for more information.

**Vizrt API URL**

**Viz DataServer user**

**Viz DataServer password**

**Vizrt search**

**Field mapping**

Field set -

### 3.4.15. EVS

**API URL**

**API key**

RabbitMQ parameters:

**Host**

**Port**

**Virtual host**

**Username**

**Password**

### 3.4.16. Other

**Short name**

This name will be label visible in the story script instead of generic ***MOS*** .

**Foreground**

This colour will be used in the story script ***MOS*** tag instead of default white colour.

**Background**

This colour will be used in the story script ***MOS*** tag instead of default green colour.

**Rundown logging interval [s]**

How often the MOS Agent logs the changes/updates related to the slugs and story attributes in the MOS active rundowns, 0 means instantly.

**Send octext_tags**

If enabled then MOSAgent will send the "octopus extension" tags i.e. nonstandard tags invented by Octopus and Astra. Their names begin with <octext_...>. Enable this for Astra or other devices which support and need these extensions.

**Send octext_tags in EMD**

**Send and receive messages in UTF8**

If enabled then MOSAgent will send and receives XML in the UTF8 encoding instead of the Unicode encoding required by the MOS protocol. UTF8 is much more efficient for Latin scripts and less efficient for exotic scripts. Enable only if the MOS device is not only capable but also configured to use UTF8 otherwise the whole MOS communication will fail completely without clear error messages. Astra supports UTF8.

**Omit milliseconds when sending time**

If enabled then MOSAgent will send date and time data without the millisecond part. This affects format of the roSlug as well.

**Omit timezone when sending time**

If enabled then MOSAgent will send date and time data without the timezone part. This affects format of the roSlug as well.

**Log heartbeats**

**Send heartbeats on incoming sockets**

**Use roElementAction**

If enabled then MOSAgent will send the <roElementAction> messages instead of the equivalent but older <roStoryInsert/Append/Replace/Move/Delete> messages. Enable it only if the MOS device does not support these old messages.

**Send and receive times in UTC**

If enabled then MOSAgent will send date and time data in the UTC timezone. Also, it will assume that any received date and time data that lacks timezone will be UTC. If disabled then the local time zone will be used.

**Receive nested messages**

**Message transmit script**

**GUI script**

**Server script**

### 3.4.17. Protocol

#### 3.4.17.1. Supported messages

MOSAgent is able to receive:

- <mosObj> <mosListAll> <mosItemReplace> <roAck> <roReq> <roReqAll> <roItemStat> <roElementStat> <heartbeat> <reqMachInfo>

MOSAgent is not able to receive:

- <mosReqObj> <mosListSearchableSchema> <mosObjList> <roStat> <roReqStoryAction> <listMachInfo>

MOSAgent sends:

- <mosReqAll> <mosObjCreate> <mosReqObjAction> <mosAck>
- <roCreate> <roReplace> <roMetadataReplace> <roDelete> <roList> <roListAll> <roReadyToAir>
- <roStoryAppend> <roStoryInsert> <roStoryReplace> <roStoryMove> <roStoryDelete> <roElementAction> <roStorySend>
- <heartbeat> <listMachInfo>

MOSAgent does not send:

- <mosReqSearchableSchema> <mosReqObjList> <roStorySwap> <roStoryMoveMultiple>
- <roItemInsert> <roItemReplace> <roItemMoveMultiple> <roItemDelete>
- <roItemCue> <roCtrl>
- <reqMachInfo>

#### 3.4.17.2. Custom story columns

MOSAgent sends custom story columns in <mosExternalMetadata> in all messages containing the <story> tag and in the <roStorySend> and <mosObjCreate>. Each custom column XML tag has a "name" attribute which contains the label of the column i.e. the name of the column visible in the Octopus client.

# Chapter 4. MOS Agent Troubleshooting

The MOSAgent service creates the logs under log folder by default. The logs are divided based on the date. There are a few logs created:

- logging.log - log of the MOSAgent logging
- mos.log - memory usage
- mos_%MOSID%.log - you can see the details of the communication between Octopus MOS Agent and remote mos gateway
- mosActor_%MOSID%.log -
- serverConnection.log - messages exchanged between the MOSAgent and Octopus Server
- serverConnection_net.log - socket log
- threads.log - thread activity log

## 4.1. How to restart MOSAgent service on Windows?

Open the Windows Services, run **service.msc** from the command line or search for services in the Start menu, locate the ***Octopus MOS Agent %MOSID%*** service and right click on it and choose the action you want to do.

## 4.2. How to restart MOSAgent service on Linux?

Log in to the server where it runs as sudoer and run following command **sudo /etc/init.d/octopus-mos-%mosid% restart** where %mosid% is the unique name of the device. You can list all devices when you press TAB on the keyboard twice. You can use also stop / start commands instead of restart.

## 4.3. Newly created MOS device is not listed for the activation under `MOS` button in the Rundown.

You have to enable the activation for the particular channel, go to the Administration / MOS Devices / Devices / select particular MOS device, click on `Edit` go to the ***Activation*** tab, select the channel, click on `Edit` and check ***Enabled***.

## 4.4. Connection status shows only *Partial*.

There are three possible connection statuses:

- ***Yes*** with green background - all four ports are connected, most typical for playouts, the communication works on rundown and media port (lower and upper port) in both ways, so Octopus is sending rundowns and clips information to the device and we're also receiving the information about the rundown and clips from the device.
- ***Partial*** with orange background - just some ports are connected, most typical for the prompters and some graphics devices. There is just partial communication, typically on rundown port, as the graphics objects are received via ActiveX most of the time and the prompters do not send anything back to Octopus. It is absolutely fine in these cases.
- ***No*** with red background - there is no connection at all - the MOSAgent service is not running, the IPs or the ports are not configured properly or something is blocking the network communication (OS firewall, network firewall, router setting, switch setting, group policy).

You can see more detailed info (queue and status of each port and the duration of the connection) if you hover the mouse cursor over the connection status.

## 4.5. Too many MOS activated rundowns

You can see what rundowns are MOS Active under ***Administration / MOS Devices / Activation*** and select the MOS Device. There are two types of active rundowns - ***manual***ly activated and ***in range*** (automatically activated). You can simply select manually activated rundowns and deactivate them by clicking on `Deactivate`. If you want to adjust the range of automatically activated

rundowns you have to go to the MOS Device config under ***Administration / MOS Devices / Devices*** select the device and click on **Edit** in toolbar, go to the***Activations*** tab, choose the channel and click on **Edit** button on the right side and adjust the range back and ahead (see chapter 3.3.12 of this guide).

## 4.6. Changes in Octopus are not reflected on the remote side immediately.

- Check that there are maximum 20 MOS active rundowns, in case that you are using manual activation. Go to Administration / MOS Devices / Activations / select particular device and you can see all active rundowns on the right side.
- Check that the communication is not overwhelmed between the Octopus MOS Agent and remote MOS gateway. Other devices can sometimes send several thousands of updates in one minute and that can significantly slow down the response to the updates.
- Try to resend the rundown - click on **MOS** / **Activate** in the rundown and then click **Resend** next to the device where do you want to resend the rundown (configured refresh method will be used and content of the rundown will be resent)
- Deactivate and Activate the rundown again - in case of manual activation click on **MOS** / **Activate** in the rundown and then uncheck particular mos device and press **OK**, then open the same dialogue again and check the particular device and click **OK**. The rundown will be sent again to the remote mos gateway.

## 4.7. How to enable and assign/receive the MOS channels - the roItemChannel

- Open the client as the administrator and tick the option Using MOS channels under Administration / System Setup / Story tab. Press **Save**.
- Open the configuration of the MOS Device under Administration / MOS Device / Devices / select the particular device and click on **Edit**, switch to ***Channels*** tab. See the chapter 3.3.6 about the Channels in this guide.
- If you want to see the channels in the rundown you have to add a new column into the Rundown form. Open some rundown, right click on the name of any column and enable **Channels**. Or if you want to add the column to all users you have to modify the Rundown form under Administration / List / Forms
- A new sub-menu item called **Assign MOS item channels** under **MOS** button is available when the MOS channels are enabled in the System Setup. You have two options how to assign the channel. You can assign the channel based on the specified rule. You can specify that rule when you click on **MOS** / **Activation**. Choose the RO channel and/or Rule. Once this setting is selected hit the **OK** and then **MOS** / **Assign MOS item channels**. The channels will be assigned to the clips in that particular rundown based on the specified rules. When the rundown is MOS active the channel is sent via MOS.
- The other option is that you can assign the channels manually. Open the story with the clip you want to assign the channel to. Click on **Edit** and right click on the white***No channel*** and choose the channel you want to assign and press **Save**. When the rundown is MOS active the channel information is sent via MOS.

## 4.8. Element labels are not visible on the prompter

You can configure what is visible on the prompter side under Administration / MOS Device / Devices / select the prompter and go to the Prompter tab and tick ***Send story element labels***. See the Configuration chapter, section Prompting for more information. You can check the mos_%MOSID%.log under /octopus/services/log/%date%/ and search for some particular story. In this log file, you can see complete communication between remote MOS gateway and Octopus MOSAgent.

Sample how does the story look like in the logs:

```
2016-03-24 12:50:58.159   45 LogTopic.Agent.MOS.MSG DEBUG   TCPChannel sending <mos><mosID>PROMPTER</mosID><ncsID>OCTOPUS</ncsID><roStorySend><roID>69821277</roID><storyID>70098402</storyID><storySlu
```

Where you can find following tags:

- <mosID> - unique MOSID of the MOS device
- <ncsID> - NCSID set in the mos device config - most of the time OCTOPUS
- <roStorySend> - name of the command
- <roID> - unique rundown identifier in Octopus database
- <storyID> - unique story identifier in Octopus database
- <storySlug> - name of the story
- <storyNum> - position of the story in the rundown
- <storyBody> - actual content of the story
- <pi> - element labels
- <p> - paragraphs

## 4.9. Content of some elements is not visible on the prompter.

You can configure which elements are sent to the particular MOS device under ***Administration / MOS Device / Devices /*** select the ***Prompter*** device and go to the ***Prompter*** tab and multi-select (press and hold **Ctrl** and mouse click) the list of the elements. See theSection 3.4.4, “Prompting”

## 4.10. The content of some elements is not printed when printing prompter.

You can configure which elements are printed when using default Prompter print form or Custom one. Go to the ***Administration / Story / Story element sets*** and create a new entry by clicking on **New** or modify the current one by selecting an item in the list and clicking on **Edit**. Multiselect elements in the ***Story element set*** dialogue. These elements will be printed when particular ***Element set*** is selected when printing ***Prompter***.

## 4.11. How to change MOSID?

> **Warning**
>
> If you change MOSID current mos objects with that MOSID will not be visible in the stories anymore. They will simply disappear as they are bounded to the MOSID and if it is not present in the list of the MOS devices it is not possible to show them.

1. Stop the Octopus MOS Agent service of the particular MOS device on the server.
2. On Windows modify MOSAgent.xml of the particular MOS Agent service, change MOSID - it is the last parameter in the lines with the arguments - see chapter 2.1.2.

   On Linux edit the daemon and change the MOSID parameter in the arguments in line PARGS - see chapter 2.2.2.
3. Change the MOSID in the client configuration under Administration / MOS Devices / Devices, select desired device, click on **Edit** and change MOSID.
4. Start the Octopus MOS Agent service of the particular MOS device on the server.

> **Tip**
>
> It might be good to rename the folder where the MOSAgent is installed. In that case you have to unregister the service by executing **MOSAgent.exe -uninstall**, rename the folder and register the service again by executing **MOSAgent.exe -install**. Keep in mind that you have to set recovery actions again - described in chapter 2.1.4. If the change is significant you should change other parameters of the service in the MOSAgent.xml as id, name, and description, keep in mind to make it clear for somebody who will connect there in the future.

## 4.12. Missing msvcr71.dll

If you can not launch the MOSAgent service and you are getting the message with ***Missing msvcr71.dll*** you have to install appropriate (32bit or 64bit) version of the Microsoft Visual C++ Redistributable package.