mosromgr

Release 0.10.0

unknown

Feb 02, 2023

CONTENTS

Python library for managing MOS running orders. Pronounced *mos-ro-manager*.

The library provides functionality for classifying MOS file types, processing and inspecting MOS message files, as well as merging MOS files into a running order, and providing a "completed" programme including all additions and changes made between the first message (`roCreate`) and the last (`roDelete`).

This can be used as a library, using the utilities provided in the *mosromgr* module, and the command line command *CLI* can be used to process either a directory of MOS files, or a folder within an S3 bucket.

This library was developed by the BBC News Labs team.

Warning: Note that the library is currently in beta. The API and CLI are not yet stable and may change. Once the library reaches v1.0, it will be considered stable. Please consider giving *Feedback* to help stabilise the API.

CHAPTER

ONE

EXAMPLE USAGE

1.1 Command line

Inspect the contents of a MOS file:

```
$ mosromgr inspect -f roCreate.mos.xml
roCreate.mos.xml: RunningOrder
RO: 22:30 NEWSNIGHT 54D CORE Tue, 09.11.2021
STORY: OPENMEDIA_NCS.W1.BBC.MOS;OM_4.15749118;OM_4.15749119,4.15749118.1
STORY: OPENMEDIA_NCS.W1.BBC.MOS;OM_4.15749118;OM_4.15749121,4.15749118.3
STORY: OPENMEDIA_NCS.W1.BBC.MOS;OM_4.15749118;OM_4.15749123,4.15749118.5
```

```
$ mosromgr inspect -f roElementAction.mos.xml
roElementAction.mos.xml: EAStoryReplace
REPLACE STORY: OPENMEDIA_NCS.W1.BBC.MOS;OM_4.15749118;OM_4.15749156,4.15749118.67 WITH:
  STORY: OPENMEDIA_NCS.W1.BBC.MOS;OM_4.15749118;OM_4.15749156,4.15749118.67
```

Merge all MOS files in directory *newsnight* and save in `FINAL.xml`:

```
$ mosromgr merge -f newsnight/* -o FINAL.xml
```

1.2 Library

Load a `roCreate` file and view its stories:

```
from mosromgr.mostypes import RunningOrder

ro = RunningOrder.from_file('roCreate.mos.xml')

for story in ro.stories:
    print(story.slug)
```

Merge a single `roStorySend` (*StorySend*) into a `roCreate` (*RunningOrder*) and output the file to a new file:

```
from mosromgr.mostypes import RunningOrder, StorySend

ro = RunningOrder.from_file('roCreate.mos.xml')
ss = StorySend.from_file('roStorySend.mos.xml')
```

(continues on next page)

(continued from previous page)

```
ro += ss

with open('final.mos.xml', 'w') as f:
    f.write(str(ro))
```

If you're automating this process you won't necessarily know which MOS Type to use, so you can construct an object from the base class *MosFile* which will automatically classify your file:

```
>>> from mosromgr.mostypes import MosFile
>>> mf1 = MosFile.from_file('roCreate.mos.xml')
>>> mf1
<RunningOrder 1000>
>>> mf2 = MosFile.from_file('roStorySend.mos.xml')
>>> mf2
<StorySend 1001>
```

Using *MosCollection* will sort and classify multiple MOS types of all given files, allowing you to process a collection of MOS files within a complete or partially complete programme:

```
from mosromgr.moscollection import MosCollection

mos_files = ['roCreate.mos.xml', 'roStorySend.mos.xml', 'roDelete.mos.xml']
mc = MosCollection.from_files(mos_files)

mc.merge()
with open('final.mos.xml', 'w') as f:
    f.write(str(mc))
```

CHAPTER

TWO

DOCUMENTATION

This documentation follows the Diátaxis system, so is split between four modes of documentation: tutorials, how-to guides, technical reference and explanation.

2.1 Getting started

This section shows you how to get started with *mosromgr*.

2.1.1 Installing

Install with pip:

```
$ pip install mosromgr
```

2.1.2 Command line interface check

After installing the module, a simple way to verify it's working is by using the *CLI*. First of all, open a terminal and run the command `mosromgr` to be sure it's installed. You should see output like so:

```
$ mosromgr
optional arguments:
  -h, --help            show this help message and exit
  --version             show program's version number and exit

commands:
  {help,detect,inspect,merge}
    help                Displays help about the specified command
    detect              Detect the MOS type of one or more files
    inspect             Inspect the contents of a roCreate file
    merge               Merge the given MOS files
```

Now start by obtaining the MOS files for a single complete programme. In a terminal window, enter the directory containing the MOS files and run the command `mosromgr detect` on a single roCreate file, for example:

```
$ mosromgr detect 123456-roCreate.mos.xml
123456-roCreate.mos.xml: RunningOrder
```

The output shows that it's identified the `roCreate` file as a *RunningOrder*. Try it with some other files to check it can correctly identify a *MosFile* subclass to represent the file.

2.1.3 Using the module in Python code

Now you've tested the ready-made command line program is working with your MOS file, try using the module in some custom Python code.

Open a Python shell and try creating a MOS object from your `roCreate` file:

```
>>> from mosromgr.mostypes import RunningOrder
>>> ro = RunningOrder.from_file('123456-roCreate.mos.xml')
>>> ro
<RunningOrder 123456>
```

This shows you've successfully loaded a MOS file and created a *RunningOrder* from it. The output shows the object representation (`__repr__`) which includes the class name and message ID (this is from the XML contents, not the filename).

The next page will walk through the functionality provided by the module.

2.2 Introduction

This section is a walkthrough of the contents of the module, intended to explain how *mosromgr* works and introduce the concepts.

2.2.1 MOS Types

The *MOS Types* section of the module provides a collection of classes for dealing with individual MOS messages. The classes provide easy access to some of the elements within a MOS file, such as a list of stories within a running order, the transmission time of a programme, or its duration.

For example, you can load a running order from a `roCreate` file, print the RO Slug and access some details:

```
>>> from mosromgr.mostypes import RunningOrder
>>> ro = RunningOrder.from_file('123456-roCreate.mos.xml')
>>> ro.ro_slug
'22:45 NEWSNIGHT 54D CORE Thu, 08.04.2021'
>>> ro.message_id
123456
>>> ro.start_time
datetime.datetime(2021, 4, 8, 21, 46, 30)
>>> ro.duration
970.0
>>> len(ro.stories)
10
```

In the case of MOS messages which contain a *change* to a running order, the relevant details are exposed, for example a *StoryInsert* includes access to the *source_stories* and *target_story*.

When dealing with merging *MosFile* objects, this is done by "adding" each file to the *RunningOrder* object by using the + operator:

```
>>> from mosromgr.mostypes import RunningOrder, StoryInsert
>>> ro = RunningOrder.from_file('123456-roCreate.mos.xml')
>>> ss = StoryInsert.from_file('123457-roStoryInsert.mos.xml')
```

(continues on next page)

(continued from previous page)

```
>>> len(ro.stories)
10
>>> ro += ss
>>> len(ro.stories)
11
```

2.2.2 MOS Elements

The *MOS Elements* part of the module provides a collection of classes used to provide easy access to certain elements within a *MosFile* object, such as a list of stories within a running order, and the items within a story:

```
from mosromgr.mostypes import RunningOrder

ro = RunningOrder.from_file('123456-roCreate.mos.xml')

print(ro.ro_slug)
for story in ro.stories:
    print(story.slug)
```

Here, `ro.stories` is a list of *Story* objects. Each story has its own set of accessible properties, such as the story's *duration*, *start_time*, *end_time*, *offset* and *items*:

```
>>> story = ro.stories[0]
>>> story.duration
180.0
>>> story.start_time
datetime.datetime(2021, 4, 8, 21, 46, 30)
>>> len(story.items)
3
```

Here, the story contains 3 items, each of these is an *Item* object.

2.2.3 MOS Collection

The *MOS Collection* part of the module provides a wrapper class *MosCollection* which stores references to specified MOS files, strings or S3 object keys so the *MosFile* objects can be recreated when needed rather than kept in memory. Rather than using the + operator, a *merge()* method is provided:

```
from mosromgr.moscollection import MosCollection

mc = MosCollection.from_s3(bucket_name=bucket_name, prefix=prefix)

mc.merge()
```

The next page will cover some example problems and solutions to show you how you can use *mosromgr* in practice.

2.3 How-to guide

This section is a series of helpful recipes for how to do things and solve particular problems with *mosromgr*.

Note: For simplicity, these examples deal with MOS messages read from local files, but *MosFile* and *MosCollection* objects can also be constructed using *from_string* and *from_s3*. Refer to *MOS Types* and *MOS Collection* for more information.

2.3.1 Merging MOS files

When dealing with merging *MosFile* objects, this is done by "adding" each file to the *RunningOrder* object by using the + operator:

```
>>> from mosromgr.mostypes import RunningOrder, StoryInsert
>>> ro = RunningOrder.from_file('123456-roCreate.mos.xml')
>>> si = StoryInsert.from_file('123457-roStoryInsert.mos.xml')
>>> len(ro.stories)
10
>>> ro += si
>>> len(ro.stories)
11
```

To parse and merge a collection of MOS files, you could create a list of files (or use `glob()`), let *MosFile* classify each file, then merge each of them into the *RunningOrder*:

```
from mosromgr.mostypes import MosFile
from glob import glob

files = glob('*.mos.xml')

ro, *mosfiles = sorted(MosFile.from_file(f) for f in files)

for mf in mosfiles:
    ro += mf
```

To access the final XML, simply print the *RunningOrder* object or access the *__str__*:

```
>>> print(ro)
<mos>
  <mosID>MOS ID</mosID>
  <messageID>1234567</messageID>
  ...
>>> s = str(ro)
>>> s
<mos>
  <mosID>MOS ID</mosID>
  <messageID>1234567</messageID>
  ...
```

2.3.2 Merging MOS files using MOSCollection

The *MosCollection* class provides a wrapper for operations dealing with a collection of MOS files as part of one programme. So to merge files like in the previous example, you could do the following instead:

```
from mosromgr.moscollection import MosCollection
from glob import glob

files = glob('*.mos.xml')
mc = MosCollection.from_files(files)

mc.merge()
```

To access the final XML, simply print the *MosCollection* object or access the *__str__*:

```
>>> print(mc)
<mos>
  <mosID>MOS ID</mosID>
  <messageID>1234567</messageID>
  ...
>>> s = str(mc)
>>> s
<mos>
  <mosID>MOS ID</mosID>
  <messageID>1234567</messageID>
  ...
```

2.3.3 Accessing the properties of a running order

For example, a *RunningOrder* object could contain several *Story* objects, each containing a number of *Item* objects:

```
>>> from mosromgr.mostypes import RunningOrder
>>> ro = RunningOrder.from_file('roCreate.mos.xml')
>>> ro.stories
[<Story 1234>, <Story 1235>, <Story 1236>]
>>> [story.duration for story in ro.stories]
[10, 20, 30]
>>> ro.duration
60
>>> story = ro.stories[0]
>>> story.slug
'Some story'
>>> story.items
[<Item ITEM1>, <Item ITEM2>, <Item ITEM3>]
>>> item = story.items[0]
>>> item.slug
'Some item'
```

In the case of a *StoryAppend* object, this would contain a single story:

```
>>> from mosromgr.mostypes import StoryAppend
>>> sa = StoryAppend.from_file('roStoryAppend.mos.xml')
>>> sa.story
```

(continues on next page)

(continued from previous page)

```
<Story STORY1>
>>> sa.duration
20
```

If this *StoryAppend* object was merged with a *RunningOrder* object, the new story would be accessible in the *RunningOrder* *stories* property:

```
>>> from mosromgr.mostypes import RunningOrder, StoryAppend
>>> ro = RunningOrder.from_file('roCreate.mos.xml')
>>> sa = StoryAppend.from_file('roStoryAppend.mos.xml')
>>> len(ro.stories)
3
>>> ro += sa
>>> len(ro.stories)
4
```

Additional information may be contained within the XML, and not exposed by properties the way *slug* and *object_id* are. In the sprit of escape hatches and ejector seats, the original XML in which the element was found is accessible as an `xml.etree.ElementTree.Element` object for further introspection.

For example, if you knew some of the `<item>` tags in a running order contained an `<areaID>` field, and you wanted to access its value, you could do so by inspecting the *xml* property:

```
tag = item.xml.find('areaID')
if tag is not None:
    print(tag.text)
```

2.3.4 Handling Exceptions

This can be useful for handling exceptions in your own code. For example, to handle any exception generated by the library, you can catch the library's base exception *MosRoMgrException*:

```
try:
    main()
except MosRoMgrException as e:
    print(e)
```

To catch a specific exception known to be raised under certain circumstances, each exception can be imported and handled separately if required:

```
from mosromgr.mostypes import MosFile
from mosromgr.exc import MosInvalidXML, UnknownMosFileType

try:
    ro = MosFile.from_file(mosfile)
except MosInvalidXML as e:
    print("Invalid in", mosfile)
except UnknownMosFileType as e:
    print("Unknown MOS file type", mosfile)
```

In some cases, a warning is raised rather than an exception. This means that execution is continued but a warning is output, which can be suppressed using the `warnings` module.

2.3.5 Capturing warnings

If you want to catch warnings and log them (for example, during a merge), you can use `warnings.catch_warnings`:

```
with warnings.catch_warnings(record=True) as warns:
    mc.merge()

warning_messages = [str(w.message) for w in warns]
```

2.3.6 Suppressing warnings

If you are not interested in seeing or capturing warnings, you can either use a warning filter or use `warnings.catch_warnings`:

```
with warnings.catch_warnings() as warns:
    mc.merge()
```

2.4 API

2.4.1 MOS Types

This part of the module provides the classes required for classifying and managing MOS files.

MOS Type classes are typically imported like so:

```
from mosromgr.mostypes import MosFile
```

MOS objects are constructed using one of three classmethods. Either from a file path:

```
ro = RunningOrder.from_file('roCreate.mos.xml')
```

from an XML string:

```
with open('roCreate.mos.xml') as f:
    xml = f.read()

ro = RunningOrder.from_string(xml)
```

or from an S3 file key:

```
ro = RunningOrder.from_s3(bucket_name='newsnight', mos_file_key='20200101/roCreate.mos.
↪xml')
```

Similarly, objects constructed using these classmethods on the *MosFile* base class will be automatically classified and an instance of the relevant class will be created:

```
>>> ro = MosFile.from_file('roCreate.mos.xml')
>>> ro
<RunningOrder 1000>
>>> ss = MosFile.from_file('roStorySend.mos.xml')
>>> ss
<StorySend 1001>
>>> ro = MosFile.from_string(xml1)
>>> ro
<RunningOrder 1000>
>>> ss = MosFile.from_string(xml2)
>>> ss
<StorySend 1001>
```

Even `roElementAction` files, which require a number of different subclasses, can be classified this way:

```
>>> ea1 = MosFile.from_file('roElementAction1.mos.xml')
>>> ea1
<EAStorySwap 1012>
>>> ea2 = MosFile.from_string(xml)
>>> ea2
<EAItemMove 1013>
```

Note: Your AWS credentials must be configured to construct using the *from_s3()* classmethod. See https://boto3.amazonaws.com/v1/documentation/api/latest/guide/credentials.html

MOS message classes

The following classes are used to parse and manage specific types of MOS messages.

RunningOrder

class mosromgr.mostypes.**RunningOrder**

Bases: *MosFile*

A `RunningOrder` object is created from a `roCreate` MOS file and can be constructed using classmethods *from_file()*, *from_string()* or *from_s3()*.

Specification: Create Running Order

http://mosprotocol.com/wp-content/MOS-Protocol-Documents/MOSProtocolVersion40/index.html#calibre_link-32

__add__(*other:* MosFile)

`RunningOrder` objects can be merged with other MOS files which implement a `merge` method by using the + operator, for example:

```
ro = RunningOrder.from_file('roCreate.mos.xml')
ss = StorySend.from_file('roStorySend.mos.xml')
ro += ss
```

__lt__(*other*) → bool

Sort by *message_id* i.e. `ro < ss` or `sorted([ro, ss])`

__str__()

The XML string of the MOS file

classmethod from_file(*mos_file_path: Union[Path, str]*)

Construct from a path to a MOS file

Parameters
mos_file_path (*Union[pathlib.Path, str]*) – The MOS file path

classmethod from_s3(*bucket_name: str*, *mos_file_key: str*)

Construct from a MOS file in an S3 bucket

Parameters

- **bucket_name** (*str*) – The name of the S3 bucket
- **mos_file_key** (*str*) – A MOS file key within the S3 bucket

classmethod from_string(*mos_xml_string: str*)

Construct from an XML string of a MOS document

Parameters
mos_xml_string (*str*) – The XML string of the MOS document

inspect()

Print an outline of the key file contents

property base_tag: Element

The base tag within the *xml*, as determined by *base_tag_name*

property base_tag_name: str

The name of the base XML tag for this file type

property body: List[Union[Item, str]]

A list of elements found in the story bodies. Each item in the list is either a string (representing a <p> tag) or an *Item* object (representing an <item> tag). Unlike *script*, this does not exclude empty paragraph tags.

property completed: bool

Whether or not the running order has had a *RunningOrderEnd* merged

property dict: OrderedDict

Convert XML to dictionary using `xmltodict` library. Useful for testing.

property duration: Optional[float]

Total running order duration in seconds

property end_time: Optional[datetime]

Transmission end time (if present in the XML)

property message_id: int

The MOS file's message ID

property ro_id: str

The running order ID

property ro_slug: `str`

The running order slug

property script: `List[str]`

A list of strings found in paragraph tags within the story bodies, excluding any empty paragraphs or technical notes in brackets.

property start_time: `Optional[datetime]`

Transmission start time (if present in the XML)

property stories: `List[Story]`

A list of *Story* objects within the running order

property xml: `Element`

The XML element of the MOS file

StorySend

class mosromgr.mostypes.**StorySend**

Bases: *MosFile*

A `StorySend` object is created from a `roStorySend` MOS file and can be constructed using classmethods *from_file()*, *from_string()* or *from_s3()*.

`StorySend` objects can be merged with a *RunningOrder* by using the + operator. This behaviour is defined in the *merge()* method in this class.

Specification: Send Story information, including Body of the Story

http://mosprotocol.com/wp-content/MOS-Protocol-Documents/MOSProtocolVersion40/index.html#calibre_link-49

__lt__(*other*) → bool

Sort by *message_id* i.e. `ro < ss` or `sorted([ro, ss])`

__str__()

The XML string of the MOS file

classmethod from_file(*mos_file_path: Union[Path, str]*)

Construct from a path to a MOS file

Parameters
mos_file_path (*Union[pathlib.Path, str]*) – The MOS file path

classmethod from_s3(*bucket_name: str*, *mos_file_key: str*)

Construct from a MOS file in an S3 bucket

Parameters

- **bucket_name** (*str*) – The name of the S3 bucket
- **mos_file_key** (*str*) – A MOS file key within the S3 bucket

classmethod from_string(*mos_xml_string: str*)

Construct from an XML string of a MOS document

Parameters
mos_xml_string (*str*) – The XML string of the MOS document

inspect()

Print an outline of the key file contents

merge(*ro:* RunningOrder) → *RunningOrder*

Merge into the *RunningOrder* object provided.

property base_tag: Element

The base tag within the *xml*, as determined by *base_tag_name*

property base_tag_name: str

The name of the base XML tag for this file type

property dict: OrderedDict

Convert XML to dictionary using `xmltodict` library. Useful for testing.

property message_id: int

The MOS file's message ID

property ro_id: str

The running order ID

property story: *Story*

The *Story* object being sent

property xml: Element

The XML element of the MOS file

StoryReplace

class mosromgr.mostypes.**StoryReplace**

Bases: *MosFile*

A `StoryReplace` object is created from a `roStoryReplace` MOS file and can be constructed using classmethods *from_file()*, *from_string()* or *from_s3()*.

`StoryReplace` objects can be merged with a *RunningOrder* by using the + operator. This behaviour is defined in the *merge()* method in this class.

Specification: Replace a Story with Another in a Running Order

The roStoryReplace message replaces the referenced story with another story or stories. This messages also replaces all items associated with the original story or stories.

http://mosprotocol.com/wp-content/MOS-Protocol-Documents/MOS_Protocol_Version_2.8.5_Final.htm#roStoryReplace

__lt__(*other*) → bool

Sort by *message_id* i.e. `ro < ss` or `sorted([ro, ss])`

__str__()

The XML string of the MOS file

classmethod from_file(*mos_file_path: Union[Path, str]*)

Construct from a path to a MOS file

Parameters

mos_file_path (*Union[pathlib.Path, str]*) – The MOS file path

classmethod from_s3(*bucket_name: str*, *mos_file_key: str*)

Construct from a MOS file in an S3 bucket

Parameters

- **bucket_name** (*str*) – The name of the S3 bucket
- **mos_file_key** (*str*) – A MOS file key within the S3 bucket

classmethod from_string(*mos_xml_string: str*)

Construct from an XML string of a MOS document

Parameters

mos_xml_string (*str*) – The XML string of the MOS document

inspect()

Print an outline of the key file contents

merge(*ro:* RunningOrder) → *RunningOrder*

Merge into the *RunningOrder* object provided.

property base_tag: Element

The base tag within the *xml*, as determined by *base_tag_name*

property base_tag_name: str

The name of the base XML tag for this file type

property dict: OrderedDict

Convert XML to dictionary using `xmltodict` library. Useful for testing.

property message_id: int

The MOS file's message ID

property ro_id: str

The running order ID

property stories: List[*Story*]

A list of replacement *Story* objects

property story: *Story*

The *Story* object being replaced

property xml: Element

The XML element of the MOS file

StoryInsert

class mosromgr.mostypes.**StoryInsert**

Bases: *MosFile*

A `StoryInsert` object is created from a `roStoryInsert` MOS file and can be constructed using classmethods *from_file()*, *from_string()* or *from_s3()*.

`StoryInsert` objects can be merged with a *RunningOrder* by using the + operator. This behaviour is defined in the *merge()* method in this class.

Specification: Insert Stories in Running Order

This message inserts stories and all of their defined items before the referenced story in a Running Order.

http://mosprotocol.com/wp-content/MOS-Protocol-Documents/MOS_Protocol_Version_2.8.5_Final.htm#roStoryInsert

__lt__(*other*) → bool

Sort by *message_id* i.e. `ro < ss` or `sorted([ro, ss])`

__str__()

The XML string of the MOS file

classmethod from_file(*mos_file_path: Union[Path, str]*)

Construct from a path to a MOS file

Parameters
mos_file_path (*Union[pathlib.Path, str]*) – The MOS file path

classmethod from_s3(*bucket_name: str*, *mos_file_key: str*)

Construct from a MOS file in an S3 bucket

Parameters

- **bucket_name** (*str*) – The name of the S3 bucket
- **mos_file_key** (*str*) – A MOS file key within the S3 bucket

classmethod from_string(*mos_xml_string: str*)

Construct from an XML string of a MOS document

Parameters
mos_xml_string (*str*) – The XML string of the MOS document

inspect()

Print an outline of the key file contents

merge(*ro:* RunningOrder) → *RunningOrder*

Merge into the *RunningOrder* object provided.

property base_tag: Element

The base tag within the *xml*, as determined by *base_tag_name*

property base_tag_name: str

The name of the base XML tag for this file type

property dict: OrderedDict

Convert XML to dictionary using `xmltodict` library. Useful for testing.

property message_id: int

The MOS file's message ID

property ro_id: str

The running order ID

property source_stories: List[*Story*]

A list of *Story* objects to be inserted

property target_story: *Story*

The *Story* object above which the source stories are to be inserted

property xml: Element

The XML element of the MOS file

StoryAppend

class mosromgr.mostypes.**StoryAppend**

Bases: *MosFile*

A `StoryAppend` object is created from a `roStoryAppend` MOS file and can be constructed using classmethods *from_file()*, *from_string()* or *from_s3()*.

`StoryAppend` objects can be merged with a *RunningOrder* by using the + operator. This behaviour is defined in the *merge()* method in this class.

Specification: Append Stories to Running Order

The roStoryAppend message appends stories and all of their defined items at the end of a running order.

http://mosprotocol.com/wp-content/MOS-Protocol-Documents/MOS_Protocol_Version_2.8.5_Final.htm#roStoryAppend

__lt__(*other*) → bool

Sort by *message_id* i.e. `ro < ss` or `sorted([ro, ss])`

__str__()

The XML string of the MOS file

classmethod from_file(*mos_file_path: Union[Path, str]*)

Construct from a path to a MOS file

Parameters
mos_file_path (*Union[pathlib.Path, str]*) – The MOS file path

classmethod from_s3(*bucket_name: str*, *mos_file_key: str*)

Construct from a MOS file in an S3 bucket

Parameters

- **bucket_name** (*str*) – The name of the S3 bucket
- **mos_file_key** (*str*) – A MOS file key within the S3 bucket

classmethod from_string(*mos_xml_string: str*)

Construct from an XML string of a MOS document

Parameters
mos_xml_string (*str*) – The XML string of the MOS document

inspect()

Print an outline of the key file contents

merge(*ro:* RunningOrder) → *RunningOrder*

Merge into the *RunningOrder* object provided.

property base_tag: Element

The base tag within the *xml*, as determined by *base_tag_name*

property base_tag_name: str

The name of the base XML tag for this file type

property dict: OrderedDict

Convert XML to dictionary using `xmltodict` library. Useful for testing.

property message_id: *int*

The MOS file's message ID

property ro_id: *str*

The running order ID

property stories: *List[Story]*

A list of *Story* objects to be appended

property xml: *Element*

The XML element of the MOS file

StoryMove

class mosromgr.mostypes.**StoryMove**

Bases: *MosFile*

A `StoryMove` object is created from a `roStoryMove` MOS file and can be constructed using classmethods *from_file()*, *from_string()* or *from_s3()*.

`StoryMove` objects can be merged with a *RunningOrder* by using the + operator. This behaviour is defined in the *merge()* method in this class.

Specification: Move a story to a new position in the Playlist

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

http://mosprotocol.com/wp-content/MOS-Protocol-Documents/MOS_Protocol_Version_2.8.5_Final.htm#roStoryMove

__lt__(*other*) → bool

Sort by *message_id* i.e. `ro < ss` or `sorted([ro, ss])`

__str__()

The XML string of the MOS file

classmethod from_file(*mos_file_path: Union[Path, str]*)

Construct from a path to a MOS file

Parameters
mos_file_path (*Union[pathlib.Path, str]*) – The MOS file path

classmethod from_s3(*bucket_name: str*, *mos_file_key: str*)

Construct from a MOS file in an S3 bucket

Parameters

- **bucket_name** (*str*) – The name of the S3 bucket
- **mos_file_key** (*str*) – A MOS file key within the S3 bucket

classmethod from_string(*mos_xml_string: str*)

Construct from an XML string of a MOS document

Parameters
mos_xml_string (*str*) – The XML string of the MOS document

inspect()

Print an outline of the key file contents

merge(*ro:* RunningOrder) → *RunningOrder*

Merge into the *RunningOrder* object provided.

property base_tag: Element

The base tag within the *xml*, as determined by *base_tag_name*

property base_tag_name: str

The name of the base XML tag for this file type

property dict: OrderedDict

Convert XML to dictionary using `xmltodict` library. Useful for testing.

property message_id: int

The MOS file's message ID

property ro_id: str

The running order ID

property source_story: Optional[*Story*]

The *Story* object to be moved

property target_story: Optional[*Story*]

The *Story* object above which the source story is to be moved

property xml: Element

The XML element of the MOS file

StoryDelete

class mosromgr.mostypes.**StoryDelete**

Bases: *MosFile*

A `StoryDelete` object is created from a `roStoryDelete` MOS file and can be constructed using classmethods *from_file()*, *from_string()* or *from_s3()*.

`StoryDelete` objects can be merged with a *RunningOrder* by using the + operator. This behaviour is defined in the *merge()* method in this class.

Specification: Delete Stories from Running Order

The roStoryDelete message deletes the referenced Stories and all associated Items from the Running Order.

http://mosprotocol.com/wp-content/MOS-Protocol-Documents/MOS_Protocol_Version_2.8.5_Final.htm#roStoryDelete

__lt__(*other*) → bool

Sort by *message_id* i.e. `ro < ss` or `sorted([ro, ss])`

__str__()

The XML string of the MOS file

classmethod from_file(*mos_file_path: Union[Path, str]*)

Construct from a path to a MOS file

Parameters

mos_file_path (*Union[pathlib.Path, str]*) – The MOS file path

classmethod from_s3(*bucket_name: str*, *mos_file_key: str*)

Construct from a MOS file in an S3 bucket

Parameters

- **bucket_name** (*str*) – The name of the S3 bucket
- **mos_file_key** (*str*) – A MOS file key within the S3 bucket

classmethod from_string(*mos_xml_string: str*)

Construct from an XML string of a MOS document

Parameters

mos_xml_string (*str*) – The XML string of the MOS document

inspect()

Print an outline of the key file contents

merge(*ro:* RunningOrder) → *RunningOrder*

Merge into the *RunningOrder* object provided.

property base_tag: Element

The base tag within the *xml*, as determined by *base_tag_name*

property base_tag_name: str

The name of the base XML tag for this file type

property dict: OrderedDict

Convert XML to dictionary using `xmltodict` library. Useful for testing.

property message_id: int

The MOS file's message ID

property ro_id: str

The running order ID

property stories: List[*Story*]

A list of *Story* objects to be deleted

property xml: Element

The XML element of the MOS file

MetaDataReplace

class mosromgr.mostypes.**MetaDataReplace**

Bases: *MosFile*

A `MetaDataReplace` object is created from a `roMetadataReplace` MOS file and can be constructed using classmethods *from_file()*, *from_string()* or *from_s3()*.

`MetaDataReplace` objects can be merged with a *RunningOrder* by using the + operator. This behaviour is defined in the *merge()* method in this class.

Specification: Replace RO metadata without deleting the RO structure

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

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

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

http://mosprotocol.com/wp-content/MOS-Protocol-Documents/MOSProtocolVersion40/index.html#calibre_link-34

__lt__(*other*) → bool

Sort by *message_id* i.e. `ro < ss` or `sorted([ro, ss])`

__str__()

The XML string of the MOS file

classmethod from_file(*mos_file_path: Union[Path, str]*)

Construct from a path to a MOS file

Parameters
mos_file_path (*Union[pathlib.Path, str]*) – The MOS file path

classmethod from_s3(*bucket_name: str*, *mos_file_key: str*)

Construct from a MOS file in an S3 bucket

Parameters

- **bucket_name** (*str*) – The name of the S3 bucket
- **mos_file_key** (*str*) – A MOS file key within the S3 bucket

classmethod from_string(*mos_xml_string: str*)

Construct from an XML string of a MOS document

Parameters
mos_xml_string (*str*) – The XML string of the MOS document

inspect()

Print an outline of the key file contents

merge(*ro:* RunningOrder) → *RunningOrder*

Merge into the *RunningOrder* object provided.

property base_tag: Element

The base tag within the *xml*, as determined by *base_tag_name*

property base_tag_name: str

The name of the base XML tag for this file type

property dict: OrderedDict

Convert XML to dictionary using `xmltodict` library. Useful for testing.

property message_id: int

The MOS file's message ID

property ro_id: str

The running order ID

property ro_slug: str

The running order slug

property xml: Element

The XML element of the MOS file

ItemDelete

class mosromgr.mostypes.**ItemDelete**

Bases: *MosFile*

An `ItemDelete` object is created from a `roItemDelete` MOS file and can be constructed using classmethods *from_file()*, *from_string()* or *from_s3()*.

`ItemDelete` objects can be merged with a *RunningOrder* by using the + operator. This behaviour is defined in the *merge()* method in this class.

Specification: Delete Items in Story

The roItemDelete message deletes one or more items in a story.

http://mosprotocol.com/wp-content/MOS-Protocol-Documents/MOS_Protocol_Version_2.8.5_Final.htm#roItemDelete

__lt__(*other*) → bool

Sort by *message_id* i.e. `ro < ss` or `sorted([ro, ss])`

__str__()

The XML string of the MOS file

classmethod from_file(*mos_file_path: Union[Path, str]*)

Construct from a path to a MOS file

Parameters
mos_file_path (*Union[pathlib.Path, str]*) – The MOS file path

classmethod from_s3(*bucket_name: str*, *mos_file_key: str*)

Construct from a MOS file in an S3 bucket

Parameters

- **bucket_name** (*str*) – The name of the S3 bucket
- **mos_file_key** (*str*) – A MOS file key within the S3 bucket

classmethod from_string(*mos_xml_string: str*)

Construct from an XML string of a MOS document

Parameters
mos_xml_string (*str*) – The XML string of the MOS document

inspect()

Print an outline of the key file contents

merge(*ro:* RunningOrder) → *RunningOrder*

Merge into the *RunningOrder* object provided.

property base_tag: Element

The base tag within the *xml*, as determined by *base_tag_name*

property base_tag_name: str

The name of the base XML tag for this file type

property dict: OrderedDict

Convert XML to dictionary using `xmltodict` library. Useful for testing.

property items: List[*Item*]

A list of the *Item* objects being deleted

property message_id: int

The MOS file's message ID

property ro_id: str

The running order ID

property story: *Story*

The *Story* object containing the items being deleted

property xml: Element

The XML element of the MOS file

ItemInsert

class mosromgr.mostypes.**ItemInsert**

Bases: *MosFile*

An `ItemInsert` object is created from a `roItemInsert` MOS file and can be constructed using classmethods *from_file()*, *from_string()* or *from_s3()*.

`ItemInsert` objects can be merged with a *RunningOrder* by using the + operator. This behaviour is defined in the *merge()* method in this class.

Specification: Insert Items in Story

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

http://mosprotocol.com/wp-content/MOS-Protocol-Documents/MOS_Protocol_Version_2.8.5_Final.htm#roItemInsert

__lt__(*other*) → bool

Sort by *message_id* i.e. `ro < ss` or `sorted([ro, ss])`

__str__()

The XML string of the MOS file

classmethod from_file(*mos_file_path: Union[Path, str]*)

Construct from a path to a MOS file

Parameters
mos_file_path (*Union[pathlib.Path, str]*) – The MOS file path

classmethod from_s3(*bucket_name: str*, *mos_file_key: str*)

Construct from a MOS file in an S3 bucket

Parameters

- **bucket_name** (*str*) – The name of the S3 bucket
- **mos_file_key** (*str*) – A MOS file key within the S3 bucket

classmethod from_string(*mos_xml_string: str*)

Construct from an XML string of a MOS document

Parameters
mos_xml_string (*str*) – The XML string of the MOS document

inspect()
Print an outline of the key file contents

merge(*ro:* RunningOrder) → *RunningOrder*
Merge into the *RunningOrder* object provided.

property base_tag: Element
The base tag within the *xml*, as determined by *base_tag_name*

property base_tag_name: str
The name of the base XML tag for this file type

property dict: OrderedDict
Convert XML to dictionary using `xmltodict` library. Useful for testing.

property item: *Item*
The *Item* object above which the items are to be inserted

property items: List[*Item*]
A list of *Item* objects to be inserted

property message_id: int
The MOS file's message ID

property ro_id: str
The running order ID

property story: *Story*
The *Story* object into which the items are to be inserted

property xml: Element
The XML element of the MOS file

ItemMoveMultiple

class mosromgr.mostypes.**ItemMoveMultiple**
Bases: *MosFile*

An `ItemMoveMultiple` object is created from a `roItemMoveMultiple` MOS file and can be constructed using classmethods *from_file()*, *from_string()* or *from_s3()*.

`ItemMoveMultiple` objects can be merged with a *RunningOrder* by using the + operator. This behaviour is defined in the *merge()* method in this class.

Specification: Move one or more Items to a specified position within a Story

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

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

http://mosprotocol.com/wp-content/MOS-Protocol-Documents/MOS_Protocol_Version_2.8.5_Final.htm#roItemMoveMultiple

__lt__(*other*) → bool

Sort by *message_id* i.e. `ro < ss` or `sorted([ro, ss])`

__str__()

The XML string of the MOS file

classmethod from_file(*mos_file_path: Union[Path, str]*)

Construct from a path to a MOS file

Parameters
mos_file_path (*Union[pathlib.Path, str]*) – The MOS file path

classmethod from_s3(*bucket_name: str*, *mos_file_key: str*)

Construct from a MOS file in an S3 bucket

Parameters

- **bucket_name** (*str*) – The name of the S3 bucket
- **mos_file_key** (*str*) – A MOS file key within the S3 bucket

classmethod from_string(*mos_xml_string: str*)

Construct from an XML string of a MOS document

Parameters
mos_xml_string (*str*) – The XML string of the MOS document

inspect()

Print an outline of the key file contents

merge(*ro:* RunningOrder) → *RunningOrder*

Merge into the *RunningOrder* object provided.

property base_tag: Element

The base tag within the *xml*, as determined by *base_tag_name*

property base_tag_name: str

The name of the base XML tag for this file type

property dict: OrderedDict

Convert XML to dictionary using `xmltodict` library. Useful for testing.

property item: Optional[*Item*]

The *Item* object above which the items will be moved (if the last itemID tag is not empty)

property items: List[*Item*]

A list of *Item* objects to be moved

property message_id: int

The MOS file's message ID

property ro_id: str

The running order ID

property story: *Story*

The *Story* object containing the items being moved

property xml: Element

The XML element of the MOS file

ItemReplace

class mosromgr.mostypes.**ItemReplace**

Bases: *MosFile*

An `ItemReplace` object is created from a `roItemReplace` MOS file and can be constructed using classmethods *from_file()*, *from_string()* or *from_s3()*.

`ItemReplace` objects can be merged with a *RunningOrder* by using the + operator. This behaviour is defined in the *merge()* method in this class.

Specification: Replace an Item with one or more Items in a Story

The roItemReplace message replaces the referenced item in a story with one or more items.

http://mosprotocol.com/wp-content/MOS-Protocol-Documents/MOS_Protocol_Version_2.8.5_Final.htm#roItemReplace

__lt__(*other*) → bool

Sort by *message_id* i.e. `ro < ss` or `sorted([ro, ss])`

__str__()

The XML string of the MOS file

classmethod from_file(*mos_file_path: Union[Path, str]*)

Construct from a path to a MOS file

Parameters
mos_file_path (*Union[pathlib.Path, str]*) – The MOS file path

classmethod from_s3(*bucket_name: str*, *mos_file_key: str*)

Construct from a MOS file in an S3 bucket

Parameters

- **bucket_name** (*str*) – The name of the S3 bucket
- **mos_file_key** (*str*) – A MOS file key within the S3 bucket

classmethod from_string(*mos_xml_string: str*)

Construct from an XML string of a MOS document

Parameters
mos_xml_string (*str*) – The XML string of the MOS document

inspect()

Print an outline of the key file contents

merge(*ro:* RunningOrder) → *RunningOrder*

Merge into the *RunningOrder* object provided.

property base_tag: Element

The base tag within the *xml*, as determined by *base_tag_name*

property base_tag_name: str

The name of the base XML tag for this file type

property dict: OrderedDict

Convert XML to dictionary using `xmltodict` library. Useful for testing.

property item: *Item*

The *Item* object being replaced

property items: List[*Item*]

A list of replacement *Item* objects

property message_id: int

The MOS file's message ID

property ro_id: str

The running order ID

property story: *Story*

The *Story* object containing the item being replaced

property xml: Element

The XML element of the MOS file

ReadyToAir

class mosromgr.mostypes.**ReadyToAir**

Bases: *MosFile*

A `ReadyToAir` object is created from a `roReadyToAir` MOS file and can be constructed using classmethods *from_file()*, *from_string()* or *from_s3()*.

`ReadyToAir` objects can be merged with a *RunningOrder* by using the + operator. This behaviour is defined in the *merge()* method in this class.

Specification: Identify a Running Order as Ready to Air

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

http://mosprotocol.com/wp-content/MOS-Protocol-Documents/MOSProtocolVersion40/index.html#calibre_link-41

__lt__(*other*) → bool

Sort by *message_id* i.e. `ro < ss` or `sorted([ro, ss])`

__str__()

The XML string of the MOS file

classmethod from_file(*mos_file_path: Union[Path, str]*)

Construct from a path to a MOS file

Parameters
mos_file_path (*Union[pathlib.Path, str]*) – The MOS file path

classmethod from_s3(*bucket_name: str*, *mos_file_key: str*)

Construct from a MOS file in an S3 bucket

Parameters

- **bucket_name** (*str*) – The name of the S3 bucket
- **mos_file_key** (*str*) – A MOS file key within the S3 bucket

classmethod from_string(*mos_xml_string: str*)

Construct from an XML string of a MOS document

Parameters
mos_xml_string (*str*) – The XML string of the MOS document

inspect()

Print an outline of the key file contents

merge(*ro:* RunningOrder) → *RunningOrder*

Merge into the *RunningOrder* object provided.

Currently unimplemented - has no effect on the running order. TODO: #18

property base_tag: Element

The base tag within the *xml*, as determined by *base_tag_name*

property base_tag_name: str

The name of the base XML tag for this file type

property dict: OrderedDict

Convert XML to dictionary using `xmltodict` library. Useful for testing.

property message_id: int

The MOS file's message ID

property ro_id: str

The running order ID

property xml: Element

The XML element of the MOS file

EAStoryReplace

class mosromgr.mostypes.**EAStoryReplace**

Bases: *ElementAction*

An `EAStoryReplace` object is created from a `roElementAction` MOS file containing a story replacement, and can be constructed using classmethods *from_file()*, *from_string()* or *from_s3()*.

`EAStoryReplace` objects can be merged with a *RunningOrder* by using the + operator. This behaviour is defined in the *merge()* method in this class.

Specification: Replacing a story

In element_target: A storyID specifying the story to be replaced

In element_source: One or more stories to put in its place

http://mosprotocol.com/wp-content/MOS-Protocol-Documents/MOSProtocolVersion40/index.html#calibre_link-43

__lt__(*other*) → bool

Sort by *message_id* i.e. `ro < ss` or `sorted([ro, ss])`

__str__()

The XML string of the MOS file

classmethod from_file(*mos_file_path: Union[Path, str]*)

Construct from a path to a MOS file

Parameters
mos_file_path (*Union[pathlib.Path, str]*) – The MOS file path

classmethod from_s3(*bucket_name: str*, *mos_file_key: str*)

Construct from a MOS file in an S3 bucket

Parameters

- **bucket_name** (*str*) – The name of the S3 bucket
- **mos_file_key** (*str*) – A MOS file key within the S3 bucket

classmethod from_string(*mos_xml_string: str*)

Construct from an XML string of a MOS document

Parameters
mos_xml_string (*str*) – The XML string of the MOS document

inspect()

Print an outline of the key file contents

merge(*ro:* RunningOrder) → *RunningOrder*

Merge into the *RunningOrder* object provided.

property base_tag: Element

The base tag within the *xml*, as determined by *base_tag_name*

property base_tag_name: str

The name of the base XML tag for this file type

property dict: OrderedDict

Convert XML to dictionary using `xmltodict` library. Useful for testing.

property message_id: int

The MOS file's message ID

property ro_id: str

The running order ID

property stories: List[Story]

A list of replacement *Story* objects

property story: Story

The *Story* object being replaced

property xml: Element

The XML element of the MOS file

EAItemReplace

class mosromgr.mostypes.**EAItemReplace**

Bases: *ElementAction*

An `EAItemReplace` object is created from a `roElementAction` MOS file containing an item replacement, and can be constructed using classmethods *from_file()*, *from_string()* or *from_s3()*.

`EAItemReplace` objects can be merged with a *RunningOrder* by using the + operator. This behaviour is defined in the *merge()* method in this class.

Specification: Replacing an item

In element_target: A storyID and itemID specifying the item to be replaced

In element_source: One or more items to put in its place

http://mosprotocol.com/wp-content/MOS-Protocol-Documents/MOSProtocolVersion40/index.html#calibre_link-43

__lt__(*other*) → bool

Sort by *message_id* i.e. `ro < ss` or `sorted([ro, ss])`

__str__()

The XML string of the MOS file

classmethod from_file(*mos_file_path: Union[Path, str]*)

Construct from a path to a MOS file

Parameters
mos_file_path (*Union[pathlib.Path, str]*) – The MOS file path

classmethod from_s3(*bucket_name: str*, *mos_file_key: str*)

Construct from a MOS file in an S3 bucket

Parameters

- **bucket_name** (*str*) – The name of the S3 bucket
- **mos_file_key** (*str*) – A MOS file key within the S3 bucket

classmethod from_string(*mos_xml_string: str*)

Construct from an XML string of a MOS document

Parameters
mos_xml_string (*str*) – The XML string of the MOS document

inspect()

Print an outline of the key file contents

merge(*ro:* RunningOrder) → *RunningOrder*

Merge into the *RunningOrder* object provided.

property base_tag: `Element`

The base tag within the *xml*, as determined by *base_tag_name*

property base_tag_name: `str`

The name of the base XML tag for this file type

property dict: `OrderedDict`

Convert XML to dictionary using `xmltodict` library. Useful for testing.

property item: *Item*

The *Item* object being replaced

property items: List[*Item*]

A list of replacement *Item* objects

property message_id: int

The MOS file's message ID

property ro_id: str

The running order ID

property story: *Story*

The *Story* object containing the item being replaced

property xml: Element

The XML element of the MOS file

EAStoryDelete

class mosromgr.mostypes.**EAStoryDelete**

Bases: *ElementAction*

An `EAStoryDelete` object is created from a `roElementAction` MOS file containing a story deletion, and can be constructed using classmethods *from_file()*, *from_string()* or *from_s3()*.

`EAStoryDelete` objects can be merged with a *RunningOrder* by using the + operator. This behaviour is defined in the *merge()* method in this class.

Specification: Deleting stories

In element_target: Not needed, since deletes don't happen relative to another story

In element_source: One or more storyIDs specifying the stories to be deleted

http://mosprotocol.com/wp-content/MOS-Protocol-Documents/MOSProtocolVersion40/index.html#calibre_link-43

__lt__(*other*) → bool

Sort by *message_id* i.e. `ro < ss` or `sorted([ro, ss])`

__str__()

The XML string of the MOS file

classmethod from_file(*mos_file_path: Union[Path, str]*)

Construct from a path to a MOS file

Parameters
mos_file_path (*Union[pathlib.Path, str]*) – The MOS file path

classmethod from_s3(*bucket_name: str*, *mos_file_key: str*)

Construct from a MOS file in an S3 bucket

Parameters

- **bucket_name** (*str*) – The name of the S3 bucket
- **mos_file_key** (*str*) – A MOS file key within the S3 bucket

classmethod from_string(*mos_xml_string: str*)

Construct from an XML string of a MOS document

Parameters
mos_xml_string (*str*) – The XML string of the MOS document

inspect()

Print an outline of the key file contents

merge(*ro:* RunningOrder) → *RunningOrder*

Merge into the *RunningOrder* object provided.

property base_tag: Element

The base tag within the *xml*, as determined by *base_tag_name*

property base_tag_name: str

The name of the base XML tag for this file type

property dict: OrderedDict

Convert XML to dictionary using `xmltodict` library. Useful for testing.

property message_id: int

The MOS file's message ID

property ro_id: str

The running order ID

property stories: List[*Story*]

A list of *Story* objects to be deleted

property xml: Element

The XML element of the MOS file

EAItemDelete

class mosromgr.mostypes.**EAItemDelete**

Bases: *ElementAction*

An `EAItemDelete` object is created from a `roElementAction` MOS file containing an item deletion, and can be constructed using classmethods *from_file()*, *from_string()* or *from_s3()*.

`EAItemDelete` objects can be merged with a *RunningOrder* by using the + operator. This behaviour is defined in the *merge()* method in this class.

Specification: Deleting items

In element_target: A storyID specifying the story containing the items to be deleted

In element_source: One or more itemIDs specifying the items in the story to be deleted

http://mosprotocol.com/wp-content/MOS-Protocol-Documents/MOSProtocolVersion40/index.html#calibre_link-43

__lt__(*other*) → bool

Sort by *message_id* i.e. `ro < ss` or `sorted([ro, ss])`

__str__()

The XML string of the MOS file

classmethod from_file(*mos_file_path: Union[Path, str]*)

Construct from a path to a MOS file

Parameters
mos_file_path (*Union[pathlib.Path, str]*) – The MOS file path

classmethod from_s3(*bucket_name: str*, *mos_file_key: str*)

Construct from a MOS file in an S3 bucket

Parameters

- **bucket_name** (*str*) – The name of the S3 bucket
- **mos_file_key** (*str*) – A MOS file key within the S3 bucket

classmethod from_string(*mos_xml_string: str*)

Construct from an XML string of a MOS document

Parameters
mos_xml_string (*str*) – The XML string of the MOS document

inspect()

Print an outline of the key file contents

merge(*ro:* RunningOrder) → *RunningOrder*

Merge into the *RunningOrder* object provided.

property base_tag: Element

The base tag within the *xml*, as determined by *base_tag_name*

property base_tag_name: str

The name of the base XML tag for this file type

property dict: OrderedDict

Convert XML to dictionary using `xmltodict` library. Useful for testing.

property items: List[*Item*]

A list of *Item* objects being deleted

property message_id: int

The MOS file's message ID

property ro_id: str

The running order ID

property story: *Story*

The *Story* object containing the items being deleted

property xml: Element

The XML element of the MOS file

EAStoryInsert

class mosromgr.mostypes.**EAStoryInsert**

Bases: *ElementAction*

An `EAStoryInsert` object is created from a `roElementAction` MOS file containing a story insertion, and can be constructed using classmethods *from_file()*, *from_string()* or *from_s3()*.

`EAStoryInsert` objects can be merged with a *RunningOrder* by using the + operator. This behaviour is defined in the *merge()* method in this class.

Specification: Inserting stories

In element_target: A storyID specifying the story before which the source stories are inserted

In element_source: One or more stories to insert

http://mosprotocol.com/wp-content/MOS-Protocol-Documents/MOSProtocolVersion40/index.html#calibre_link-43

__lt__(*other*) → bool

Sort by *message_id* i.e. `ro < ss` or `sorted([ro, ss])`

__str__()

The XML string of the MOS file

classmethod from_file(*mos_file_path: Union[Path, str]*)

Construct from a path to a MOS file

Parameters
mos_file_path (*Union[pathlib.Path, str]*) – The MOS file path

classmethod from_s3(*bucket_name: str*, *mos_file_key: str*)

Construct from a MOS file in an S3 bucket

Parameters

- **bucket_name** (*str*) – The name of the S3 bucket
- **mos_file_key** (*str*) – A MOS file key within the S3 bucket

classmethod from_string(*mos_xml_string: str*)

Construct from an XML string of a MOS document

Parameters
mos_xml_string (*str*) – The XML string of the MOS document

inspect()

Print an outline of the key file contents

merge(*ro:* RunningOrder) → *RunningOrder*

Merge into the *RunningOrder* object provided.

property base_tag: `Element`

The base tag within the *xml*, as determined by *base_tag_name*

property base_tag_name: `str`

The name of the base XML tag for this file type

property dict: `OrderedDict`

Convert XML to dictionary using `xmltodict` library. Useful for testing.

property message_id: int
The MOS file's message ID

property ro_id: str
The running order ID

property stories: List[*Story*]
The *Story* objects to be inserted

property story: *Story*
The *Story* object above which the source story will be inserted

property xml: Element
The XML element of the MOS file

EAItemInsert

class mosromgr.mostypes.**EAItemInsert**
Bases: *ElementAction*

An `EAItemInsert` object is created from a `roElementAction` MOS file containing an item insertion, and can be constructed using classmethods *from_file()*, *from_string()* or *from_s3()*.

`EAItemInsert` objects can be merged with a *RunningOrder* by using the + operator. This behaviour is defined in the *merge()* method in this class.

Specification: Inserting items

In element_target: A storyID and itemID specifying the item before which the source items are inserted

In element_source: One or more items to insert

http://mosprotocol.com/wp-content/MOS-Protocol-Documents/MOSProtocolVersion40/index.html#calibre_link-43

__lt__(*other*) → bool
Sort by *message_id* i.e. `ro < ss` or `sorted([ro, ss])`

__str__()
The XML string of the MOS file

classmethod from_file(*mos_file_path: Union[Path, str]*)
Construct from a path to a MOS file

Parameters
mos_file_path (*Union[pathlib.Path, str]*) – The MOS file path

classmethod from_s3(*bucket_name: str*, *mos_file_key: str*)
Construct from a MOS file in an S3 bucket

Parameters

- **bucket_name** (*str*) – The name of the S3 bucket
- **mos_file_key** (*str*) – A MOS file key within the S3 bucket

classmethod from_string(*mos_xml_string: str*)
Construct from an XML string of a MOS document

Parameters
mos_xml_string (*str*) – The XML string of the MOS document

inspect()

Print an outline of the key file contents

merge(*ro:* RunningOrder) → *RunningOrder*

Merge into the *RunningOrder* object provided.

property base_tag: Element

The base tag within the *xml*, as determined by *base_tag_name*

property base_tag_name: str

The name of the base XML tag for this file type

property dict: OrderedDict

Convert XML to dictionary using `xmltodict` library. Useful for testing.

property item: *Item*

The *Item* object above which the source item is to be be inserted

property items: List[*Item*]

A list of *Item* objects to be inserted

property message_id: int

The MOS file's message ID

property ro_id: str

The running order ID

property story: *Story*

The *Story* object into which the item is to be inserted

property xml: Element

The XML element of the MOS file

EAStorySwap

class mosromgr.mostypes.**EAStorySwap**

Bases: *ElementAction*

An `EAStorySwap` object is created from a `roElementAction` MOS file containing a story swap, and can be constructed using classmethods *from_file()*, *from_string()* or *from_s3()*.

`EAStorySwap` objects can be merged with a *RunningOrder* by using the + operator. This behaviour is defined in the *merge()* method in this class.

Specification: Swapping stories

In element_target: An empty storyID tag, or the element_target tag itself is absent

In element_source: Exactly two storyIDs specifying the stories to be swapped

http://mosprotocol.com/wp-content/MOS-Protocol-Documents/MOSProtocolVersion40/index.html#calibre_link-43

__lt__(*other*) → bool

Sort by *message_id* i.e. `ro < ss` or `sorted([ro, ss])`

__str__()

The XML string of the MOS file

classmethod from_file(*mos_file_path: Union[Path, str]*)

Construct from a path to a MOS file

Parameters
mos_file_path (*Union[pathlib.Path, str]*) – The MOS file path

classmethod from_s3(*bucket_name: str*, *mos_file_key: str*)

Construct from a MOS file in an S3 bucket

Parameters

- **bucket_name** (*str*) – The name of the S3 bucket
- **mos_file_key** (*str*) – A MOS file key within the S3 bucket

classmethod from_string(*mos_xml_string: str*)

Construct from an XML string of a MOS document

Parameters
mos_xml_string (*str*) – The XML string of the MOS document

inspect()

Print an outline of the key file contents

merge(*ro:* RunningOrder) → *RunningOrder*

Merge into the *RunningOrder* object provided.

property base_tag: Element

The base tag within the *xml*, as determined by *base_tag_name*

property base_tag_name: str

The name of the base XML tag for this file type

property dict: OrderedDict

Convert XML to dictionary using `xmltodict` library. Useful for testing.

property message_id: int

The MOS file's message ID

property ro_id: str

The running order ID

property stories: Tuple[*Story*]

A tuple of the two *Story* objects to be swapped

property xml: Element

The XML element of the MOS file

EAItemSwap

class mosromgr.mostypes.**EAItemSwap**

Bases: *ElementAction*

An `EAItemSwap` object is created from a `roElementAction` MOS file containing an item swap, and can be constructed using classmethods *from_file()*, *from_string()* or *from_s3()*.

`EAItemSwap` objects can be merged with a *RunningOrder* by using the + operator. This behaviour is defined in the *merge()* method in this class.

Specification: Swapping items

In element_target: A storyID specifying the story containing the items to be swapped

In element_source: Exactly two itemIDs specifying the items to be swapped

http://mosprotocol.com/wp-content/MOS-Protocol-Documents/MOSProtocolVersion40/index.html#calibre_link-43

__lt__(*other*) → bool

Sort by *message_id* i.e. `ro < ss` or `sorted([ro, ss])`

__str__()

The XML string of the MOS file

classmethod from_file(*mos_file_path: Union[Path, str]*)

Construct from a path to a MOS file

Parameters
mos_file_path (*Union[pathlib.Path, str]*) – The MOS file path

classmethod from_s3(*bucket_name: str*, *mos_file_key: str*)

Construct from a MOS file in an S3 bucket

Parameters

- **bucket_name** (*str*) – The name of the S3 bucket
- **mos_file_key** (*str*) – A MOS file key within the S3 bucket

classmethod from_string(*mos_xml_string: str*)

Construct from an XML string of a MOS document

Parameters
mos_xml_string (*str*) – The XML string of the MOS document

inspect()

Print an outline of the key file contents

merge(*ro:* RunningOrder) → *RunningOrder*

Merge into the *RunningOrder* object provided.

property base_tag: Element

The base tag within the *xml*, as determined by *base_tag_name*

property base_tag_name: str

The name of the base XML tag for this file type

property dict: OrderedDict

Convert XML to dictionary using `xmltodict` library. Useful for testing.

property items: Tuple[*Item*]

A tuple of the two *Item* objects to be swapped

property message_id: int

The MOS file's message ID

property ro_id: str

The running order ID

property story: *Story*

The *Story* object containing the items being swapped

property xml: Element

The XML element of the MOS file

EAStoryMove

class mosromgr.mostypes.**EAStoryMove**

Bases: *ElementAction*

An `EAStoryMove` object is created from a `roElementAction` MOS file containing a story move, and can be constructed using classmethods *from_file()*, *from_string()* or *from_s3()*.

`EAStoryMove` objects can be merged with a *RunningOrder* by using the + operator. This behaviour is defined in the *merge()* method in this class.

Specification: Moving stories

In element_target: A storyID specifying the story before which the source stories are moved

In element_source: One or more storyIDs specifying the stories to be moved

http://mosprotocol.com/wp-content/MOS-Protocol-Documents/MOSProtocolVersion40/index.html#calibre_link-43

__lt__(*other*) → bool

Sort by *message_id* i.e. `ro < ss` or `sorted([ro, ss])`

__str__()

The XML string of the MOS file

classmethod from_file(*mos_file_path: Union[Path, str]*)

Construct from a path to a MOS file

Parameters

mos_file_path (*Union[pathlib.Path, str]*) – The MOS file path

classmethod from_s3(*bucket_name: str*, *mos_file_key: str*)

Construct from a MOS file in an S3 bucket

Parameters

- **bucket_name** (*str*) – The name of the S3 bucket
- **mos_file_key** (*str*) – A MOS file key within the S3 bucket

classmethod from_string(*mos_xml_string: str*)

Construct from an XML string of a MOS document

Parameters

mos_xml_string (*str*) – The XML string of the MOS document

inspect()

Print an outline of the key file contents

merge(*ro:* RunningOrder) → *RunningOrder*

Merge into the *RunningOrder* object provided.

property base_tag: Element

The base tag within the *xml*, as determined by *base_tag_name*

property base_tag_name: str

The name of the base XML tag for this file type

property dict: OrderedDict

Convert XML to dictionary using `xmltodict` library. Useful for testing.

property message_id: int

The MOS file's message ID

property ro_id: str

The running order ID

property stories: List[*Story*]

A list of *Story* objects being moved

property story: *Story*

The *Story* object above which the other stories will be moved

property xml: Element

The XML element of the MOS file

EAItemMove

class mosromgr.mostypes.**EAItemMove**

Bases: *ElementAction*

An `EAItemMove` object is created from a `roElementAction` MOS file containing an item move, and can be constructed using classmethods *from_file()*, *from_string()* or *from_s3()*.

`EAItemMove` objects can be merged with a *RunningOrder* by using the + operator. This behaviour is defined in the *merge()* method in this class.

Specification: Moving items

In element_target: A storyID and itemID specifying the item before which the source items are moved

In element_source: One or more itemIDs specifying the items in the story to be moved

http://mosprotocol.com/wp-content/MOS-Protocol-Documents/MOSProtocolVersion40/index.html#calibre_link-43

__lt__(*other*) → bool

Sort by *message_id* i.e. `ro < ss` or `sorted([ro, ss])`

__str__()

The XML string of the MOS file

classmethod from_file(*mos_file_path: Union[Path, str]*)

Construct from a path to a MOS file

Parameters

mos_file_path (*Union[pathlib.Path, str]*) – The MOS file path

classmethod from_s3(*bucket_name: str*, *mos_file_key: str*)

Construct from a MOS file in an S3 bucket

Parameters

- **bucket_name** (*str*) – The name of the S3 bucket
- **mos_file_key** (*str*) – A MOS file key within the S3 bucket

classmethod from_string(*mos_xml_string: str*)

Construct from an XML string of a MOS document

Parameters

mos_xml_string (*str*) – The XML string of the MOS document

inspect()

Print an outline of the key file contents

merge(*ro:* RunningOrder) → *RunningOrder*

Merge into the *RunningOrder* object provided.

property base_tag: Element

The base tag within the *xml*, as determined by *base_tag_name*

property base_tag_name: str

The name of the base XML tag for this file type

property dict: OrderedDict

Convert XML to dictionary using `xmltodict` library. Useful for testing.

property item: *Item*

The *Item* object above which the source items will be moved

property items: List[*Item*]

A list of *Item* objects to be moved

property message_id: int

The MOS file's message ID

property ro_id: str

The running order ID

property story: *Story*

The *Story* object containing the item being replaced

property xml: Element

The XML element of the MOS file

RunningOrderReplace

class mosromgr.mostypes.**RunningOrderReplace**

Bases: *RunningOrder*

An `RunningOrderReplace` object is created from a `roReplace` MOS file and can be constructed using class-methods *from_file()*, *from_string()* or *from_s3()*.

`RunningOrderReplace` objects can be merged with a *RunningOrder* by using the + operator. This behaviour is defined in the *merge()* method in this class.

Specification: Replace Running Order

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

http://mosprotocol.com/wp-content/MOS-Protocol-Documents/MOSProtocolVersion40/index.html#calibre_link-33

__add__(*other:* MosFile)

`RunningOrder` objects can be merged with other MOS files which implement a `merge` method by using the + operator, for example:

```
ro = RunningOrder.from_file('roCreate.mos.xml')
ss = StorySend.from_file('roStorySend.mos.xml')
ro += ss
```

__lt__(*other*) → bool

Sort by *message_id* i.e. `ro < ss` or `sorted([ro, ss])`

__str__()

The XML string of the MOS file

classmethod from_file(*mos_file_path: Union[Path, str]*)

Construct from a path to a MOS file

Parameters
mos_file_path (*Union[pathlib.Path, str]*) – The MOS file path

classmethod from_s3(*bucket_name: str*, *mos_file_key: str*)

Construct from a MOS file in an S3 bucket

Parameters

- **bucket_name** (*str*) – The name of the S3 bucket
- **mos_file_key** (*str*) – A MOS file key within the S3 bucket

classmethod from_string(*mos_xml_string: str*)

Construct from an XML string of a MOS document

Parameters
mos_xml_string (*str*) – The XML string of the MOS document

inspect()

Print an outline of the key file contents

merge(*ro:* RunningOrder) → *RunningOrder*

Merge into the *RunningOrder* object provided.

property base_tag: `Element`

The base tag within the *xml*, as determined by *base_tag_name*

property base_tag_name: `str`

The name of the base XML tag for this file type

property body: `List[Union[Item, str]]`

A list of elements found in the story bodies. Each item in the list is either a string (representing a <p> tag) or an *Item* object (representing an <item> tag). Unlike *script*, this does not exclude empty paragraph tags.

property dict: OrderedDict

Convert XML to dictionary using `xmltodict` library. Useful for testing.

property duration: Optional[float]

Total running order duration in seconds

property end_time: Optional[datetime]

Transmission end time (if present in the XML)

property message_id: int

The MOS file's message ID

property ro_id: str

The running order ID

property ro_slug: str

The running order slug

property script: List[str]

A list of strings found in paragraph tags within the story bodies, excluding any empty paragraphs or technical notes in brackets.

property start_time: Optional[datetime]

Transmission start time (if present in the XML)

property stories: List[*Story*]

A list of *Story* objects within the running order

property xml: Element

The XML element of the MOS file

RunningOrderEnd

class mosromgr.mostypes.**RunningOrderEnd**

Bases: *MosFile*

A `RunningOrderEnd` object is created from a `roDelete` MOS file and can be constructed using classmethods *from_file()*, *from_string()* or *from_s3()*.

`RunningOrderEnd` objects can be merged with a *RunningOrder* by using the + operator. This behaviour is defined in the *merge()* method in this class. Once a `RunningOrderEnd` object has been merged into a *RunningOrder*, the running order is considered "completed" and no further messages can be merged.

Specification: Delete Running Order

http://mosprotocol.com/wp-content/MOS-Protocol-Documents/MOSProtocolVersion40/index.html#calibre_link-35

__lt__(*other*) → bool

Sort by *message_id* i.e. `ro < ss` or `sorted([ro, ss])`

__str__()

The XML string of the MOS file

classmethod from_file(*mos_file_path: Union[Path, str]*)

Construct from a path to a MOS file

Parameters
mos_file_path (*Union[pathlib.Path, str]*) – The MOS file path

classmethod from_s3(*bucket_name: str*, *mos_file_key: str*)

Construct from a MOS file in an S3 bucket

Parameters

- **bucket_name** (*str*) – The name of the S3 bucket
- **mos_file_key** (*str*) – A MOS file key within the S3 bucket

classmethod from_string(*mos_xml_string: str*)

Construct from an XML string of a MOS document

Parameters
mos_xml_string (*str*) – The XML string of the MOS document

inspect()

Print an outline of the key file contents

merge(*ro:* RunningOrder) → *RunningOrder*

Merge into the *RunningOrder* object provided.

Adds a `mosromgrmeta` tag containing the `roDelete` tag from the `roDelete` message to the `roCreate` tag in the running order.

property base_tag: Element

The base tag within the *xml*, as determined by *base_tag_name*

property base_tag_name: str

The name of the base XML tag for this file type

property dict: OrderedDict

Convert XML to dictionary using `xmltodict` library. Useful for testing.

property message_id: int

The MOS file's message ID

property ro_id: str

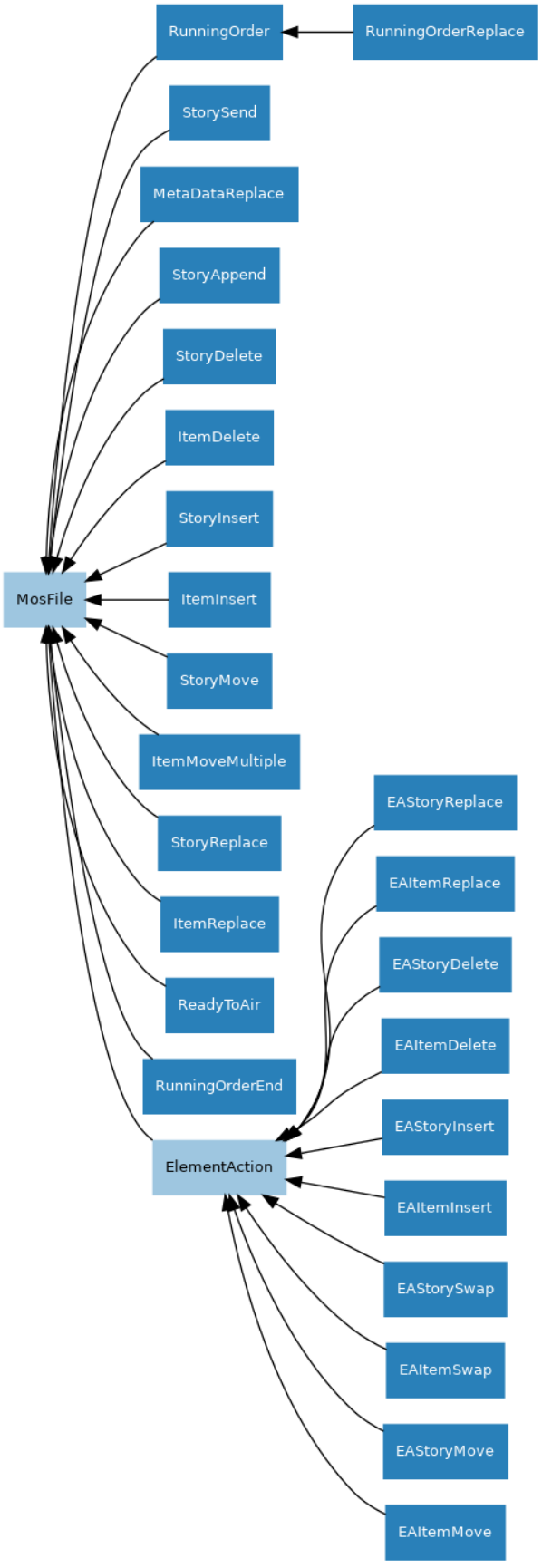
The running order ID

property xml: Element

The XML element of the MOS file

Base classes

Since some logic is shared between MOS file management, some inheritance is used in the implementation:


RunningOrder
RunningOrderReplace
StorySend
MetaDataReplace
StoryAppend
StoryDelete
ItemDelete
StoryInsert
MosFile
ItemInsert
StoryMove
ItemMoveMultiple
StoryReplace
ItemReplace
ReadyToAir
RunningOrderEnd
ElementAction
EAStoryReplace
EAItemReplace
EAStoryDelete
EAItemDelete
EAStoryInsert
EAItemInsert
EAStorySwap
EAItemSwap
EAStoryMove
EAItemMove

MosFile

class mosromgr.mostypes.**MosFile**

Base class for all MOS files

classmethod from_file(*mos_file_path: Union[Path, str]*)

Construct from a path to a MOS file

Parameters
mos_file_path (*Union[pathlib.Path, str]*) – The MOS file path

classmethod from_s3(*bucket_name: str*, *mos_file_key: str*)

Construct from a MOS file in an S3 bucket

Parameters

- **bucket_name** (*str*) – The name of the S3 bucket
- **mos_file_key** (*str*) – A MOS file key within the S3 bucket

classmethod from_string(*mos_xml_string: str*)

Construct from an XML string of a MOS document

Parameters
mos_xml_string (*str*) – The XML string of the MOS document

property base_tag: Element

The base tag within the *xml*, as determined by *base_tag_name*

property base_tag_name

The base tag (xml.etree.ElementTree.Element) within the *xml*, as determined by *base_tag_name*

property dict: OrderedDict

Convert XML to dictionary using `xmltodict` library. Useful for testing.

property message_id: int

The MOS file's message ID

property ro_id: str

The running order ID

property xml: Element

The XML element of the MOS file

ElementAction

class mosromgr.mostypes.**ElementAction**

Base class for various `roElementAction` MOS files.

Specification: Performs specific Action on a Running Order

https://mosprotocol.com/wp-content/MOS-Protocol-Documents/MOSProtocolVersion40/index.html#calibre_link-43

classmethod from_file(*mos_file_path: Union[Path, str]*)

Construct from a path to a MOS file

Parameters
mos_file_path (*Union[pathlib.Path, str]*) – The MOS file path

classmethod from_s3(*bucket_name: str*, *mos_file_key: str*)

Construct from a MOS file in an S3 bucket

Parameters

- **bucket_name** (*str*) – The name of the S3 bucket
- **mos_file_key** (*str*) – A MOS file key within the S3 bucket

classmethod from_string(*mos_xml_string: str*)

Construct from an XML string of a MOS document

Parameters

mos_xml_string (*str*) – The XML string of the MOS document

property base_tag: Element

The base tag within the *xml*, as determined by *base_tag_name*

property base_tag_name: str

The name of the base XML tag for this file type

property dict: OrderedDict

Convert XML to dictionary using `xmltodict` library. Useful for testing.

property message_id: int

The MOS file's message ID

property ro_id: str

The running order ID

property xml: Element

The XML element of the MOS file

2.4.2 MOS Elements

This part of the module provides a collection of classes used to provide easy access to certain elements within a *MosFile* object, such as a list of stories within a running order, and the items within a story.

Although usually not required directly, the MOS Element classes can be imported as follows:

```
from mosromgr.moselements import Story
```

Note: Note that these classes should not normally be constructed by the user, but instances of them can be found within *MosFile* objects, so the following documentation is provided as a reference to how they can be used.

Element classes

Story

class mosromgr.moselements.**Story**

Bases: *MosElement*

This class represents a Story element within any *MosFile* object, providing data relating to the story. The Story ID, Story slug, duration and more are exposed as properties, and the XML element is provided for further introspection.

__str__()

The XML string

property body: List[Union[*Item*, str]]

A list of elements found in the story body. Each item in the list is either a string (representing a <p> tag) or an *Item* object (representing an <item> tag). Unlike *script*, this does not exclude empty paragraph tags, which will be represented by empty strings.

property duration: float

The story duration (the sum of the text time and media time found within mosExternalMetadata->mosPayload), in seconds

property end_time: Optional[datetime]

The transmission end time of the story (if present in the XML)

property id: Optional[str]

The Story ID (if present in the XML)

property items: Optional[List[*Item*]]

List of *Item* elements found within the story (can be None if not available in the XML)

property offset: Optional[float]

The time offset of the story in seconds (if available in the XML)

property script: List[str]

A list of strings found in paragraph tags within the story body, excluding any empty paragraphs or technical notes in brackets.

property slug: Optional[str]

The Story slug (if present in the XML)

property start_time: Optional[datetime]

The transmission start time of the story (if present in the XML)

property xml: Element

The XML element

Item

class mosromgr.moselements.Item

Bases: *MosElement*

This class represents an Item element within any *MosFile* object, providing data relating to an item within a *Story*. The Item ID and Item slug (and more) are exposed as properties, and the XML element is provided for further introspection.

__str__()

The XML string

property id: Optional[str]

The Item ID (if present in the XML)

property mos_id: Optional[str]

The Item's MOS ID (if present in the XML)

property note: Optional[str]
The item note text (if present in the XML)

property object_id: Optional[str]
The Item's Object ID (if present in the XML)

property slug: Optional[str]
The Item slug (if present in the XML)

property type: Optional[str]
The Item's object type (if present in the XML)

property xml: Element
The XML element

Base classes

MosElement

class mosromgr.moselements.**MosElement**
Abstract base class for MOS elements

__str__()
The XML string

property id: Optional[str]
The element ID (if present in the XML)

property slug: Optional[str]
The element slug (if present in the XML)

property xml: Element
The XML element

2.4.3 MOS Collection

This part of the module provides a wrapper class *MosCollection* which stores references to specified MOS files, strings or S3 object keys so the *MosFile* objects can be recreated when needed rather than kept in memory.

Note: Note that creating a *MosCollection* from strings does not benefit from memory efficiency as the strings would still be held in memory.

The *MosCollection* is typically imported like so:

```
from mosromgr.moscollection import MosCollection
```

MOS collections are constructed using one of three classmethods. Either from a list of file paths:

```
mos_files = ['roCreate.mos.xml', 'roStorySend.mos.xml', 'roDelete.mos.xml']
mc = MosCollection.from_files(mos_files)
```

from a list of strings:

```
mos_strings = [roCreate, roStorySend, roDelete]
mc = MosCollection.from_strings(mos_files)
```

or from an S3 bucket:

```
mc = MosCollection.from_s3(bucket_name=bucket_name, prefix=prefix)
```

Note: Your AWS credentials must be configured to construct using the *from_s3()* classmethod. See https://boto3.amazonaws.com/v1/documentation/api/latest/guide/credentials.html

MosCollection

class mosromgr.moscollection.**MosCollection**

Wrapper for a collection of MOS files representing a partial or complete programme

__str__()

The XML string of the collection's running order

classmethod from_files(*mos_file_paths: List[Union[Path, str]]*, ***, *allow_incomplete: bool = False*)

Construct from a list of MOS file paths

Parameters

- **mos_file_paths** (*list*) – A list of paths to MOS files
- **allow_incomplete** (*bool*) – If `False` (the default), the collection is permitted to be constructed without a `roDelete`. If `True`, a *InvalidMosCollection* will be raised if one is not present. (keyword-only argument)

classmethod from_s3(***, *bucket_name: str*, *prefix: str*, *suffix: str = '.mos.xml'*, *allow_incomplete: bool = False*)

Construct from a list of MOS files in an S3 bucket

Parameters

- **bucket_name** (*str*) – The name of the S3 bucket (keyword-only argument)
- **prefix** (*str*) – The prefix of the file keys in the S3 bucket (keyword-only argument)
- **suffix** (*str*) – The suffix of the file keys in the S3 bucket (keyword-only argument). Defaults to '.mos.xml'.
- **allow_incomplete** (*bool*) – If `True`, the collection is permitted to be constructed without a `roDelete`. If `False` (the default), a *InvalidMosCollection* will be raised if one is not present. (keyword-only argument)

classmethod from_strings(*mos_file_strings: List[str]*, ***, *allow_incomplete: bool = False*)

Construct from a list of MOS document XML strings

Parameters

- **mos_file_strings** (*list*) – A list of strings containing MOS file contents
- **allow_incomplete** (*bool*) – If `False` (the default), the collection is permitted to be constructed without a `roDelete`. If `True`, a *InvalidMosCollection* will be raised if one is not present. (keyword-only argument)

merge(*, *strict: bool = True*)

Merge all MOS files into the collection's running order (*ro*). If *strict* is `True` (the default), then merge errors will be fatal. If `False`, then merge errors will be downgraded to warnings.

property completed: bool

Whether or not the running order has had a *RunningOrderEnd* merged (`bool`)

property mos_readers: List[*MosReader*]

A list of *MosReader* objects representing all MOS files in the collection, except the *RunningOrder* (`roCreate`) which is held in *ro*

property ro: *RunningOrder*

The collection's *RunningOrder* object

property ro_id: str

The running order ID

property ro_slug: str

The running order slug

MosReader

The *MosReader* class is internal and is not intended to be constructed by the user. A *MosCollection* object will contain a list of *MosReader* instances, so users may find it useful to refer to its members.

class mosromgr.moscollection.**MosReader**

Internal construct for opening and inspecting a MOS file for the purposes of classifying, sorting and validating a *MosCollection*. Provides the means to reconstruct the *MosFile* instance when needed in order to preserve memory usage.

property message_id: str

The message ID of the MOS file

property mos_object: *MosFile*

Restore the MOS object and return it

property mos_type: *MosFile*

The *MosFile* subclass this object was classified as (returns the class object, not an instance or a string)

property ro_id: str

The MOS file's running order ID

2.4.4 Utilities

This part of the module provides a collection of generic utilities which are largely for internal use.

The various utilities are typically imported like so:

```
from mosromgr.utils import s3
```

Warning: This part of the module should not be considered part of the stable API and is subject to backwards-incompatible changes.

S3

AWS S3 utilities

get_mos_files

mosromgr.utils.s3.**get_mos_files**(*bucket_name: str*, *prefix: Optional[str] = None*, ***, *suffix: str = '.mos.xml'*) → List[str]

Retrieve MOS files from given S3 bucket in location defined by *prefix*. Returns a list of file keys.

get_file_contents

mosromgr.utils.s3.**get_file_contents**(*bucket_name*, *file_key*)

Open the S3 file and return its contents as a string

XML

XML helper functions

remove_node

mosromgr.utils.xml.**remove_node**(*parent: Element*, *node: Element*)

Remove *node* from *parent*.

replace_node

mosromgr.utils.xml.**replace_node**(*parent: Element*, *old_node: Element*, *new_node: Element*, *index: int*)

Replace *old_node* with *new_node* in *parent* at *index*.

insert_node

mosromgr.utils.xml.**insert_node**(*parent: Element*, *node: Element*, *index: int*)

Insert *node* in *parent* at *index*.

find_child

mosromgr.utils.xml.**find_child**(*parent: Element*, *child_tag: str*, *id: Optional[str] = None*) → Tuple[Optional[Element], Optional[int]]

Find an element with *child_tag* in *parent* and return `(child, index)` or `(None, None)` if not found. If *id* is provided, it will be searched for, otherwise the first child will be returned.

2.4.5 Exceptions

The module's exceptions and warnings are typically imported like so:

```
from mosromgr.exc import MosRoMgrException
```

The library's base warning is *MosRoMgrWarning* and others are detailed below.

Errors

MosRoMgrException

exception mosromgr.exc.**MosRoMgrException**

Bases: Exception

Base class for all `mosromgr` exceptions

UnknownMosFileType

exception mosromgr.exc.**UnknownMosFileType**

Bases: *MosRoMgrException*

Exception raised when a MOS file type cannot be determined

MosMergeError

exception mosromgr.exc.**MosMergeError**

Bases: *MosRoMgrException*

Exception raised when MOS merge fails

MosCompletedMergeError

exception mosromgr.exc.**MosCompletedMergeError**

Bases: *MosMergeError*

Exception raised when MOS merge is attempted on a completed *RunningOrder*

InvalidMosCollection

exception mosromgr.exc.**InvalidMosCollection**

Bases: *MosRoMgrException*

Exception raised when MosCollection fails validation

MosInvalidXML

exception mosromgr.exc.**MosInvalidXML**

Bases: *MosRoMgrException*

Exception raised when *MosFile* cannot parse given XML

Warnings

MosRoMgrWarning

exception mosromgr.exc.**MosRoMgrWarning**

Bases: Warning

Base class for all warnings in mosromgr

MosMergeNonStrictWarning

exception mosromgr.exc.**MosMergeNonStrictWarning**

Bases: *MosRoMgrWarning*

Warning raised when a merge error occurs in non-strict mode

ItemNotFoundWarning

exception mosromgr.exc.**ItemNotFoundWarning**

Bases: *MosRoMgrWarning*

Warning raised when an item cannot be found during a *MosFile* merge

StoryNotFoundWarning

exception mosromgr.exc.**StoryNotFoundWarning**

Bases: *MosRoMgrWarning*

Warning raised when a story cannot be found during a *MosFile* merge

DuplicateStoryWarning

exception mosromgr.exc.**DuplicateStoryWarning**

Bases: *MosRoMgrWarning*

Warning raised when a story being added is already found during a *EAStoryInsert* merge

2.5 CLI

This section provides a reference to the command line interface.

2.5.1 mosromgr detect

Detect the MOS type of one or more files

Synopsis

```
mosromgr detect [-h] [-f [files [files ...]]] [-b bucket] [-p prefix] [-s suffix] [-k␣
↪key]
```

Description

-f --files [files ...]
: The MOS files to detect

-b --bucket-name bucket
: The name of the S3 bucket containing the MOS files

-p --prefix prefix
: The prefix for MOS files in the S3 bucket

-s --suffix suffix
: The suffix for MOS files in the S3 bucket

-k --key key
: The file key for a MOS file in the S3 bucket

-h, --help
: Show this help message and exit

Usage

Detect the type of a MOS file:

```
$ mosromgr detect -f 123456-roCreate.mos.xml
123456-roCreate.mos.xml: RunningOrder
```

Multiple files can be provided as arguments:

```
$ mosromgr detect -f 123456-roCreate.mos.xml 123457-roStorySend.mos.xml
123456-roCreate.mos.xml: RunningOrder
123457-roStorySend.mos.xml: StorySend
```

Wildcards can also be used:

```
$ mosromgr detect *
123456-roCreate.mos.xml: RunningOrder
123457-roStorySend.mos.xml: StorySend
...
9148627-roDelete.mos.xml: RunningOrderEnd
bbcProgrammeMetadata.xml: Unknown MOS file type
cricket: Invalid
FINAL.json: Invalid
FINAL.xml: RunningOrder (completed)
```

You can also read files from an S3 bucket. Either a specific file by key:

```
$ mosromgr detect -b my-bucket -k newsnight/20210101/123456-roCreate.mos.xml
OPENMEDIA_NCS.W1.BBC.MOS/OM_10.1253459/5744992-roCreate.mos.xml: RunningOrder
```

Or a whole folder by prefix:

```
$ mosromgr detect -b bbc-newslabs-slicer-mos-message-store -p newsnight/20210101/
newsnight/20210101/123456-roCreate.mos.xml: RunningOrder
newsnight/20210101/123457-roStorySend.mos.xml: StorySend
newsnight/20210101/123458-roStorySend.mos.xml: StorySend
newsnight/20210101/123459-roStorySend.mos.xml: StorySend
...
```

Note: Your AWS credentials must be configured to use the S3 method. See https://boto3.amazonaws.com/v1/documentation/api/latest/guide/credentials.html

2.5.2 mosromgr inspect

View the high-level the contents of a MOS file

Synopsis

```
mosromgr inspect [-h] [-f [files [files ...]]] [-b bucket] [-p prefix] [-s suffix] [-k␣
↪key]
```

Description

-f --files [files ...]
: The MOS files to inspect

-b --bucket-name bucket
: The name of the S3 bucket containing the MOS files

-p --prefix prefix
: The prefix for MOS files in the S3 bucket

-s --suffix suffix
: The suffix for MOS files in the S3 bucket

-k --key key
: The file key for a MOS file in the S3 bucket

-h, **--help**
: Show this help message and exit

Usage

View the contents of a local MOS file:

```
$ mosromgr inspect -f 123456-roCreate.mos.xml
RO: 22:45 NEWSNIGHT 54D CORE Thu, 08.04.2021
STORY: OPENMEDIA_NCS.W1.BBC.MOS;OM_4.15529413;OM_4.15529414,4.15529413.1
STORY: OPENMEDIA_NCS.W1.BBC.MOS;OM_4.15529413;OM_4.15529416,4.15529413.3
STORY: OPENMEDIA_NCS.W1.BBC.MOS;OM_4.15529413;OM_4.15529418,4.15529413.5
...
```

View the contents of a MOS file in S3:

```
$ mosromgr inspect -b my-bucket -k newsnight/20210804/123456-roCreate.mos.xml
RO: 22:45 NEWSNIGHT 54D CORE Thu, 08.04.2021
STORY: OPENMEDIA_NCS.W1.BBC.MOS;OM_4.15529413;OM_4.15529414,4.15529413.1
STORY: OPENMEDIA_NCS.W1.BBC.MOS;OM_4.15529413;OM_4.15529416,4.15529413.3
STORY: OPENMEDIA_NCS.W1.BBC.MOS;OM_4.15529413;OM_4.15529418,4.15529413.5
```

Note: Your AWS credentials must be configured to use the S3 method. See https://boto3.amazonaws.com/v1/documentation/api/latest/guide/credentials.html

2.5.3 mosromgr merge

Merge the provided MOS files

Synopsis

```
mosromgr merge [-h] [-f [files [files ...]]] [-b bucket] [-p prefix] [-s suffix]
               [-o outfile] [-i] [-n]
```

Description

-f --files [files ...]
: The MOS files to merge

-b --bucket-name bucket
: The name of the S3 bucket containing the MOS files

-p --prefix prefix
: The file prefix for MOS files in the S3 bucket

-s --suffix suffix
: The file suffix for MOS files in the S3 bucket

-o --outfile outfile
: Output to a file

-i --incomplete
: Allow an incomplete collection

-n --non-strict
: Downgrade MOS merge errors to warnings

-h, --help
: Show this help message and exit

Usage

Merge local files and store the result in a new file:

```
$ mosromgr merge -f *.mos.xml -o FINAL.xml
...
INFO:mosromgr.moscollection:Merging RunningOrderEnd 123499
INFO:mosromgr.moscollection:Completed merging 99 mos files
Writing merged running order to FINAL.xml
```

Merge files in an S3 bucket folder by prefix and store the result in a new file:

```
$ mosromgr merge -b my-bucket -p newsnight/20210101/ -o FINAL.xml
...
INFO:mosromgr.moscollection:Merging RunningOrderEnd 123499
INFO:mosromgr.moscollection:Completed merging 99 mos files
Writing merged running order to FINAL.xml
```

Note: Your AWS credentials must be configured to use the S3 method. See https://boto3.amazonaws.com/v1/documentation/api/latest/guide/credentials.html

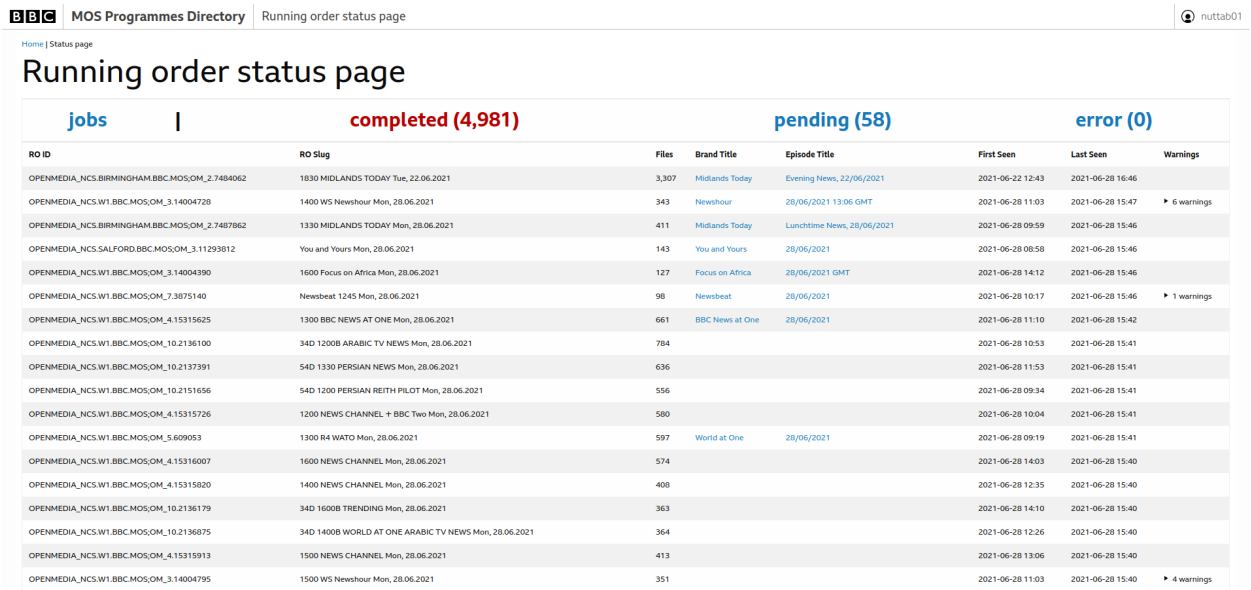
2.6 Uses of mosromgr

This section lists projects which have been known to use the *mosromgr* module. If you have used *mosromgr* in a project and would like to add it to the list, please either edit this file and open a pull request, open an issue, or send an email to bbcnewslabsteam@bbc.co.uk.

2.6.1 BBC News Labs - MOS pipeline

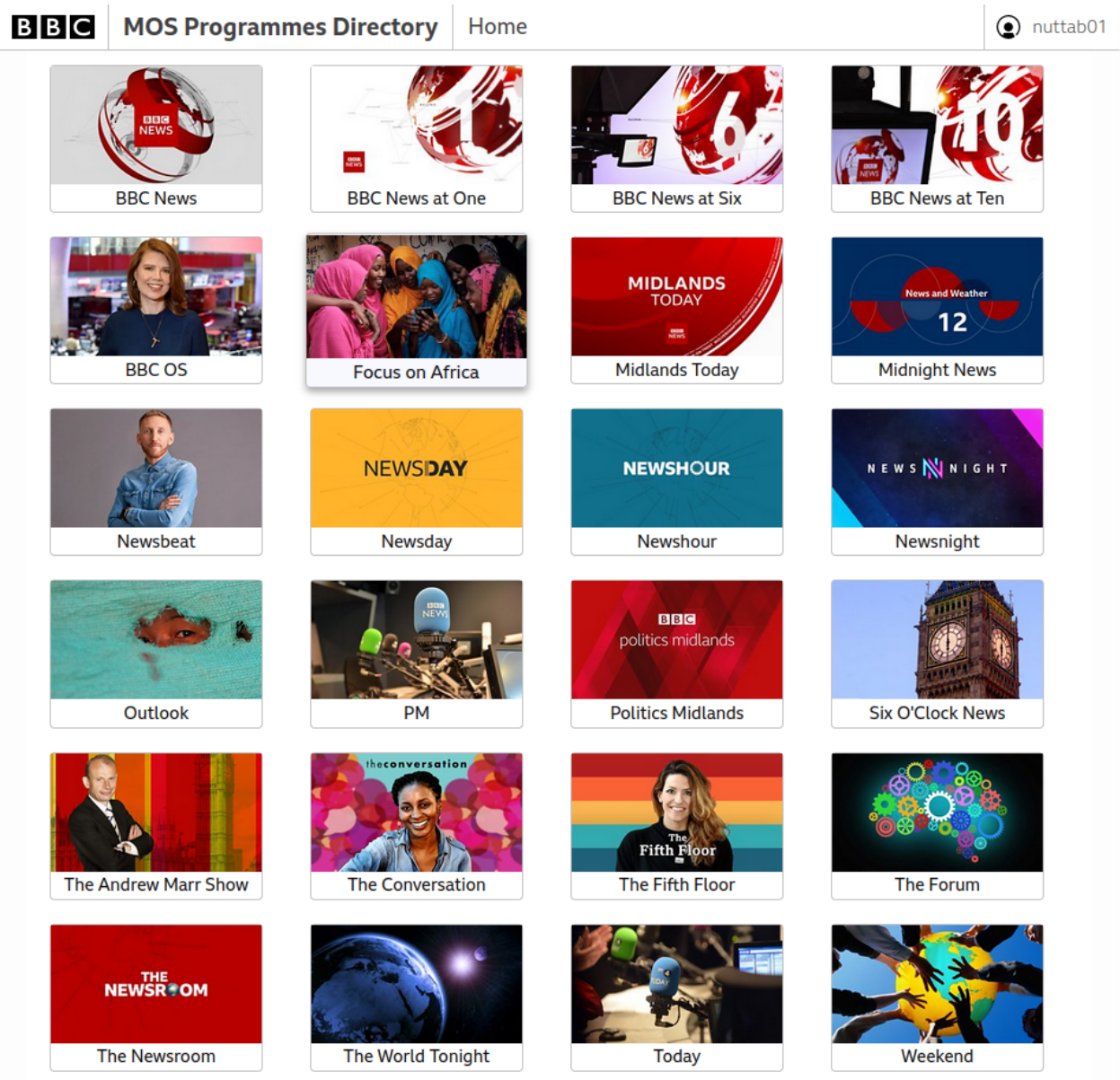
We have a collection of AWS services making up a pipeline which processes MOS messages in real time, updates a status dashboard, publishes completed MOS running orders and JSON summaries to an internal document store, and populates a directory of programmes with new episodes and lists of stories (complete with timing information) as they become available.

Status dashboard:

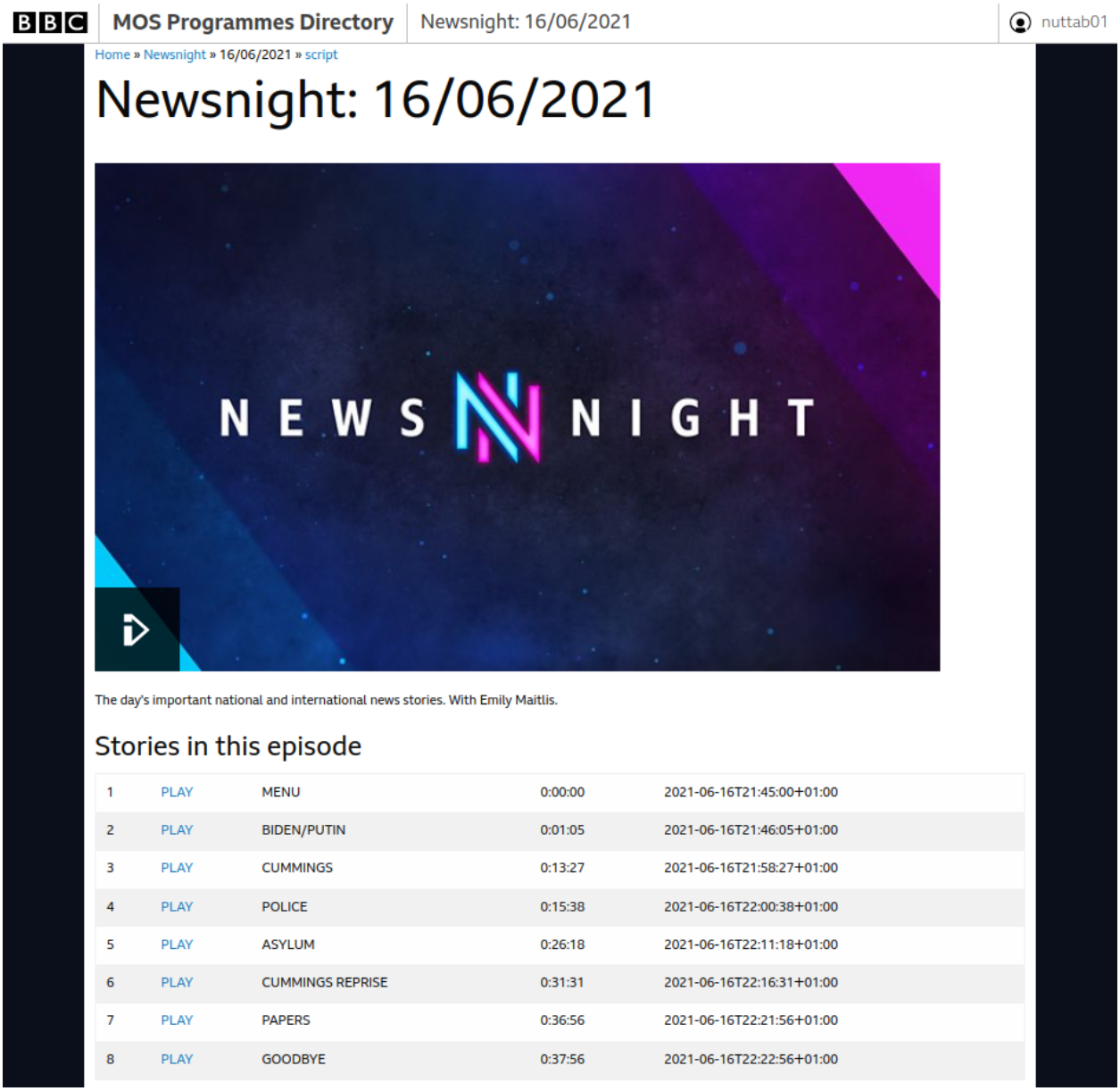


RO ID	RO Slug	Files	Brand Title	Episode Title	First Seen	Last Seen	Warnings
OPENMEDIA_NCS.BIRMINGHAM.BBC.MOS;OM_2.7484062	1830 MIDLANDS TODAY Tue, 22.06.2021	3,307	Midlands Today	Evening News, 22/06/2021	2021-06-22 12:43	2021-06-28 16:46	
OPENMEDIA_NCS.W1.BBC.MOS;OM_3.14004728	1400 WS Newshour Mon, 28.06.2021	343	Newshour	28/06/2021 13:06 GMT	2021-06-28 11:03	2021-06-28 15:47	▸ 6 warnings
OPENMEDIA_NCS.BIRMINGHAM.BBC.MOS;OM_2.7487862	1330 MIDLANDS TODAY Mon, 28.06.2021	411	Midlands Today	Lunchtime News, 28/06/2021	2021-06-28 09:59	2021-06-28 15:46	
OPENMEDIA_NCS.SALFORD.BBC.MOS;OM_3.11293812	You and Yours Mon, 28.06.2021	143	You and Yours	28/06/2021	2021-06-28 08:58	2021-06-28 15:46	
OPENMEDIA_NCS.W1.BBC.MOS;OM_3.14004390	1600 Focus on Africa Mon, 28.06.2021	127	Focus on Africa	28/06/2021 GMT	2021-06-28 14:12	2021-06-28 15:46	
OPENMEDIA_NCS.W1.BBC.MOS;OM_7.3875140	Newsbeat 1245 Mon, 28.06.2021	98	Newsbeat	28/06/2021	2021-06-28 10:17	2021-06-28 15:46	▸ 1 warnings
OPENMEDIA_NCS.W1.BBC.MOS;OM_4.15315625	1300 BBC NEWS AT ONE Mon, 28.06.2021	661	BBC News at One	28/06/2021	2021-06-28 11:10	2021-06-28 15:42	
OPENMEDIA_NCS.W1.BBC.MOS;OM_10.2136100	34D 1200B ARABIC TV NEWS Mon, 28.06.2021	784			2021-06-28 10:53	2021-06-28 15:41	
OPENMEDIA_NCS.W1.BBC.MOS;OM_10.2137391	54D 1330 PERSIAN NEWS Mon, 28.06.2021	636			2021-06-28 11:53	2021-06-28 15:41	
OPENMEDIA_NCS.W1.BBC.MOS;OM_10.2151656	54D 1200 PERSIAN REITH PILOT Mon, 28.06.2021	556			2021-06-28 09:34	2021-06-28 15:41	
OPENMEDIA_NCS.W1.BBC.MOS;OM_4.15315726	1200 NEWS CHANNEL + BBC Two Mon, 28.06.2021	580			2021-06-28 10:04	2021-06-28 15:41	
OPENMEDIA_NCS.W1.BBC.MOS;OM_5.609053	1300 R4 WATO Mon, 28.06.2021	597	World at One	28/06/2021	2021-06-28 09:19	2021-06-28 15:41	
OPENMEDIA_NCS.W1.BBC.MOS;OM_4.15316007	1600 NEWS CHANNEL Mon, 28.06.2021	574			2021-06-28 14:03	2021-06-28 15:40	
OPENMEDIA_NCS.W1.BBC.MOS;OM_4.15315820	1400 NEWS CHANNEL Mon, 28.06.2021	408			2021-06-28 12:35	2021-06-28 15:40	
OPENMEDIA_NCS.W1.BBC.MOS;OM_10.2136179	34D 1600B TRENDING Mon, 28.06.2021	363			2021-06-28 14:10	2021-06-28 15:40	
OPENMEDIA_NCS.W1.BBC.MOS;OM_10.2136875	34D 1400B WORLD AT ONE ARABIC TV NEWS Mon, 28.06.2021	364			2021-06-28 12:26	2021-06-28 15:40	
OPENMEDIA_NCS.W1.BBC.MOS;OM_4.15315913	1500 NEWS CHANNEL Mon, 28.06.2021	413			2021-06-28 13:06	2021-06-28 15:40	
OPENMEDIA_NCS.W1.BBC.MOS;OM_3.14004795	1500 WS Newshour Mon, 28.06.2021	351			2021-06-28 11:03	2021-06-28 15:40	▸ 4 warnings

Programmes directory:

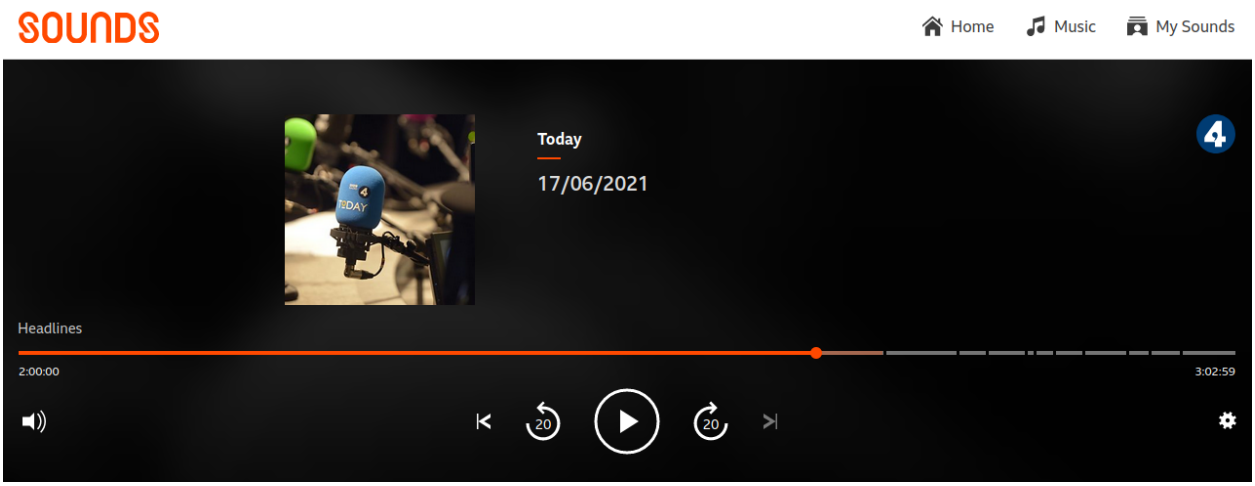


Example chapterised breakdown of an episode of Newsnight:

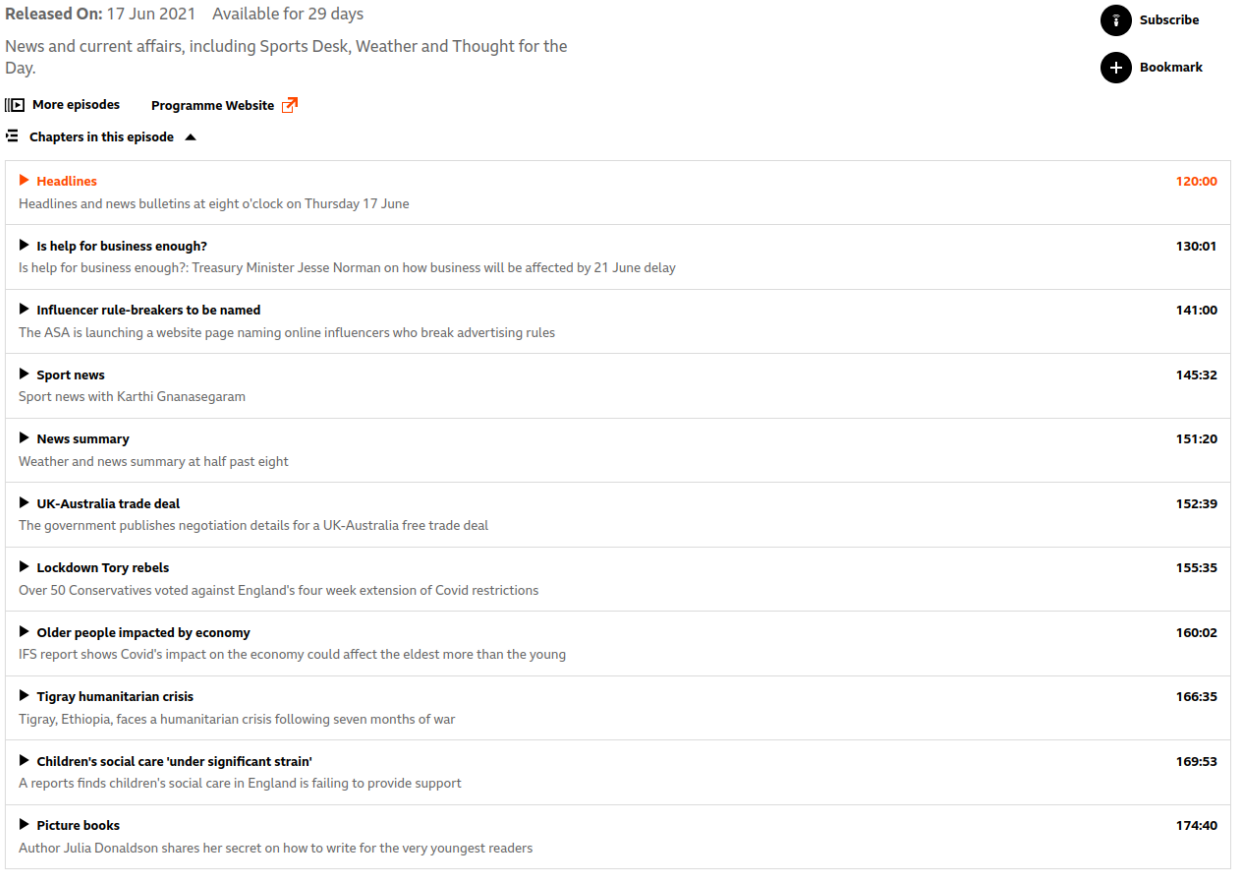


2.6.2 BBC News Labs - Auto chapterisation

We were able to decorate the player timeline with chapter points in certain BBC TV and radio programmes:

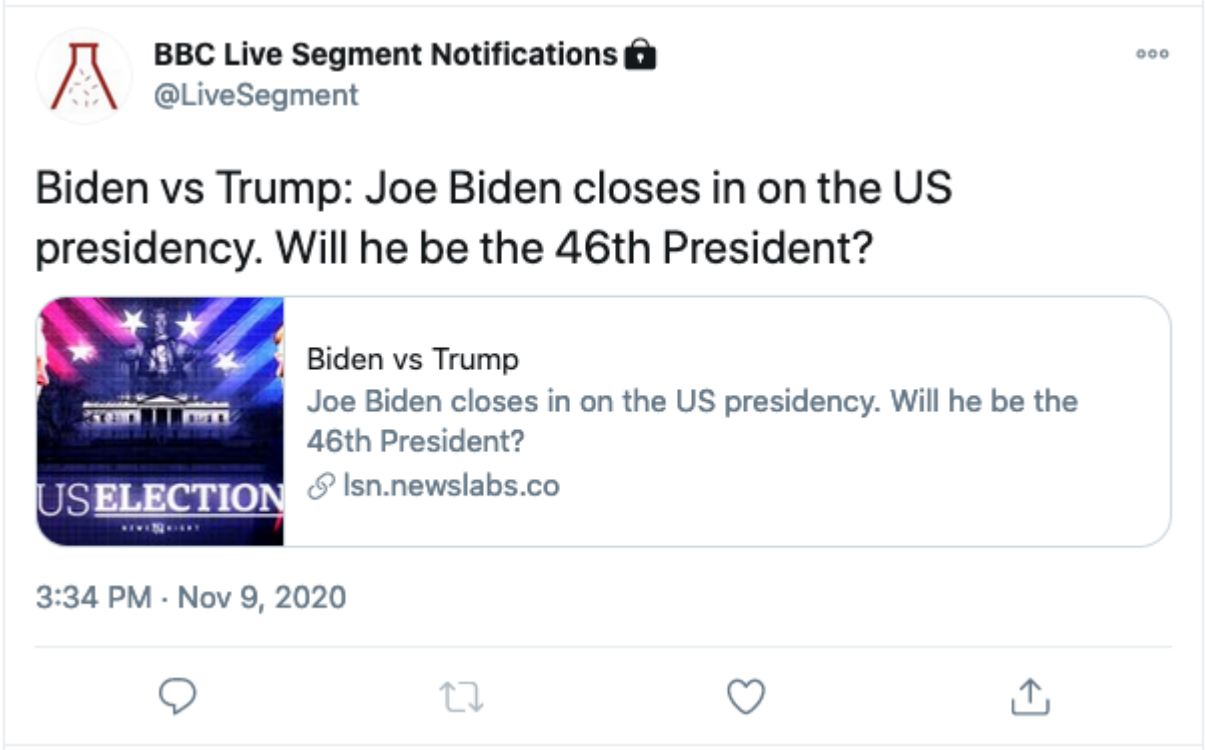


We used the script and story timing information extracted from the running order and aligned it against the transcript.



2.6.3 BBC News Labs - Live Segment Notifications

We developed a proof-of-concept in which a note within a story in a running order could trigger a tweet to alert people of an upcoming story in time to watch live, or link to the clip of the story on-demand:



2.7 Changelog

Warning: Note that the library is currently in beta. The API and CLI are not yet stable and may change. Once the library reaches v1.0, it will be considered stable. Please consider giving *Feedback* to help stabilise the API.

2.7.1 Release 0.10.0 (2022-08-25)

- Add type hints
- Remove `RunningOrderControl` class
- Add support for various edge cases in `merge` methods, fixing several bugs
- Increase test coverage for *XML*, *MOS Collection*, *MOS Elements* and *MOS Types* to 100%

2.7.2 Release 0.9.1 (2021-09-02)

- Add *type*, *object_id*, and *mos_id* properties to the *Item* class

2.7.3 Release 0.9.0 (2021-06-21)

- Updated *mosromgr inspect* CLI command to work for all file types
- Corrected some singular *MosFile MOS element* properties that should have been lists (e.g. `source_story` should have been `source_stories`)
- Improved validation and error handling when merging various *MosFile* objects
- Added *script* and *body* to *Story*
- Added *script* and *body* to *RunningOrder*
- Added non-strict mode to the *MosCollection merge()* method and the *mosromgr merge* CLI command
- Corrected some edge cases in *MosFile* subclass merge implementations (e.g. empty `storyID` tag means move to bottom)

2.7.4 Release 0.8.1 (2021-04-14)

- Fixup release

2.7.5 Release 0.8.0 (2021-04-13)

- Improved validation and error handling when merging various *MosFile* objects
- Added more arguments to CLI commands
- Corrected some singular *MosFile MOS Elements* properties that should have been lists (e.g. `source_story` should have been `source_stories`)

2.7.6 Release 0.7.0 (2021-01-08)

- Ensured exceptions are raised when story IDs are not found when merging
- Ensured tags aren't overwritten when they are empty in *MetaDataReplace*
- Ensured target story is found when merging *StoryInsert* and *StoryReplace*
- Added `RunningOrderControl` class (for `roCtrl` messages)
- Changed `tx_time` to *start_time*

2.7.7 Release 0.6.0 (2020-12-01)

- Added support for <StoryDuration> as an alternative to <MediaTime> and <TextTime>

2.7.8 Release 0.5.0 (2020-11-30)

- Added *ReadyToAir* MOS Type
- Improved error message on invalid *MosCollection*

2.7.9 Release 0.4.0 (2020-11-30)

- Changed `closed` property to *completed*
- Added transmission time and offset to *Story* class
- New *CLI* with separate commands for *mosromgr detect*, *mosromgr inspect* and *mosromgr merge*
- Make *MosCollection* raise exceptions on failure, not just warnings

2.7.10 Release 0.3.0 (2020-11-24)

- Switched from complicated `__init__` constructors to multiple `from_` classmethods e.g. *from_file()*
- Replaced `get_mos_object` function with detection logic in the *MosFile* and *ElementAction* base classes
- Replaced `MosContainer` class with *MosCollection*

2.7.11 Release 0.2.0 (2020-11-24)

- Added *MOS Elements* - a collection of classes used to provide easy access to certain elements within a *MosFile* object

2.7.12 Release 0.1.0 (2020-11-24)

- Implemented most standard MOS message types as *MosFile* subclasses, supporting merging subsequent messages into the original running order
- Implemented a MOS file detection function (`get_mos_object`)
- Added a `MOSContainer` class as a wrapper for a complete programme
- Added a *CLI* for merging MOS files

2.8 Development

This page contains reference material for those interested in developing and contributing to the **mosromgr** module.

The project source code is hosted on GitHub at https://github.com/bbc/mosromgr which also includes the issue tracker.

2.8.1 Setting up for Development

1. Clone the repository and enter the directory:

```
$ git clone https://github.com/bbc/mosromgr
$ cd mosromgr
```

2. Create a virtual environment e.g. using virtualenvwrapper:

```
$ mkvirtualenv mosromgr
```

3. Install the project for development:

```
$ make develop
```

After completing these steps, the library and command line interface will be available to use within your environment. Any modifications made to the source code will be automatically reflected within the environment.

2.8.2 Tests

The test suite uses pytest. Tests are organised mirroring the source code.

Running the tests

To run the test suite and coverage analysis, activate the environment and run:

```
$ make test
```

For more control when running tests, run `pytest` directly, for example `pytest -vvxk story` will run tests with `story` in the name (`-k story`) with verbose output (`-vv`), and stop at the first failure (`-x`).

To run tests on multiple versions of Python, run `tox` which will invoke `make test` for all versions of Python (included in `tox.ini`) that you have installed.

Tests are also automatically run against pull requests using GitHub Actions.

2.8.3 Documentation

The documentation is built using sphinx using the diataxis framework.

Building the documentation

To build the documentation, activate the environment and run:

```
$ make doc
```

This will generate the required diagrams and build the HTML docs which will be located in `docs/build/html`. Serve them with the command:

```
$ make doc-serve
```

You'll now be able to open the docs on your browser at `http://localhost:8000/`.

2.9 Feedback

Before we release v1.0 and stabilise the API, we are seeking other organisations using the MOS protocol to test *mosromgr* on their own MOS files and provide feedback so we can integrate any necessary changes to make sure it works effectively beyond the BBC's use.

If you can help, please test the module on your own MOS files and report back to us using our discussion board or issue tracker on GitHub, or email us at bbcnewslabsteam@bbc.co.uk.

2.10 Indices and tables

- genindex
- modindex
- search

CHAPTER

THREE

ISSUES AND QUESTIONS

Questions can be asked on the discussion board, and issues can be raised on the issue tracker.

CHAPTER

FOUR

CONTRIBUTING

Source code can be found on GitHub at github.com/bbc/mosromgr.

Contributions are welcome. Please refer to the contributing guidelines.

CHAPTER

FIVE

CONTRIBUTORS

- Ben Nuttall
- Owen Tourlamain
- Rob French
- Lucy MacGlashan
- Dave Bevan
- Matthew Sim

CHAPTER

SIX

LICENCE

Licensed under the Apache License, Version 2.0.

CHAPTER

SEVEN

CONTACT

To get in touch with the maintainers, please contact the BBC News Labs team: bbcnewslabsteam@bbc.co.uk

PYTHON MODULE INDEX

m

INDEX

Symbols

B

C

D

E

F

G

I

M

N

O

R

S

T

U

X